ISO/IEC JTC 1/SC 22/WG 14 N 1555

Date	11 April 2011
Contributed by	Barry Hedquist and John Benito
Original file name	N1553
Notes	Based on WG 14/N1553 and SC 22/N4578

N1553			Balloted	document:	SC22 N 4578		
				Vote:	Approve, Disapprove, Abstain		
NB (ISO 3166)	**No.**	**Category**	**Clause, Sub-clause**	**Paragraph, Figure, Table**	**Comment and rationale**	**Proposed new text**	**Response**
US	1	E	6.5.1	5	Add a 6th paragraph about generic-selection	A generic selection is a primary expression. It type depends on its form and value, as detailed in 6.5.1.1.	Editorial - Accepted
US	2	TL	7.12.1	7	"... the value corresponding to the error ..." is missing the correspondence.	The correspondence is: "invalid" => EDOM; "divide-by-zero" => ERANGE; "overflow" => ERANGE; "underflow" => ERANGE. It might be better as a table.	The Committee considered the proposed change and concluded the Standard is clear as written.
US	3	TL	6.2.8	4	"nonnegative integral power of two" is ambiguous. Is "nonnegative" referring to the exponent or the final number? Also, it does not include zero.	"Every valid alignment value shall be either zero or a positive integer (which is two to a nonnegative integral power)."	Editorial - the Committee considered the proposed change and found no consensus to adopt the change. The feeling is the document is clear as written.
US	4	TL	6.5.3.4	3	Could add that result is nonnegative.	The result is a nonnegative integer constant.	The Committee considered the proposed change and found no consensus to adopt it.
US	5	TL	G.5.1	8	N1496 was applied to wrong line.	logbw == INFINITY should be isfinite(logbw) and isinf(logbw) should be (logbw == INFINITY)	Editorial - Accepted
US	6	E	6.3.1.4	2, last line	"some" seems wrong. Either remove it or explain which ones.	Results of implicit conversions …	Editorial - the Committee considered the proposed change and found no consensus to adopt the change. The consenus was that the explaination was already in the text of the document.
US	7	E	6.3.1.5	1, last line	"some" seems wrong. Either remove it or explain which ones.	Results of implicit conversions …	Editorial - the Committee considered the proposed change and found no consensus to adopt the change. The consenus is the explaination is already the in the document.

US	8	GE			Both “precision and range” and “range and precision” are used.	Use just “range and precision”	Editorial - Accepted
US	9	GE			Both “precision or range” and “range or precision” are used.	Use just “range or precision”	Editorial - Accepted
US	10	TL	7.3.9.3	3	If the CMPLX macros are not useable in static initialization, then they have little value.	Remove “Recommended practice” and change “should” to “shall”.	Accepted
US	11	TL	F.10.3.5	3	It is ambiguous if ilogb(NaN) is outside the range of the return type. The correct value for ilogb(NaN) is NaN. But, since NaN is not representable in int, “invalid” should be raise and an unspecified value returned. But, 7.12.6.5 specifies FP_ILOGBNAN as the return value (which some people say is in the range of the return type). Same problem applies to zero and infinity.	Add a 3rd paragraph: ilogb(x), for x zero, infinite, or NaN, raises “invalid” and returns the value as specified in 7.12.6.5.	Accepted
US	12	TL	6.10.8.3	1	Need a way to distinguish freestanding from hosted.	__STDC_FREESTANDING__ The integer constant 1, intended to indicate a freestanding environment.	The Committee considered the proposed change and found no consensus to adopt it. Note that `__STDC_HOSTED__` is used for this feature test.
US	13	E	Contents		7.28.4.1 area of table of contents is expanded to four levels, while 6.5.16 is expanded to three levels. Seems like we should be consistent.	Expand all contents to same level (either 3 or 4).	Editorial -the Committee considered the proposed change and found no consensus to adopt the change. The consensus is that current wording is clear.
US	14	E	3.14	4	It would be more obvious if d:8 were on its own line.	Move “:0, d:8;” to their own line.	Editorial - the Committee considered the proposed change and found no consensus to adopt the change. The consensus was proposed change only confuses the issue.
US	15	TL	4.	4	A program that violates C's syntax should not be translated.	Add “Recommended practice – The implementation should not successfully translate a preprocessing translation unit that violates any syntax (has an erroneous program construct).”	The Committee considered the proposed change and found no consensus to adopt it.
US	16	TL	4.	8	It would be nice if a programmer could find out how to invoke an implementation in Standard C conformance mode.	Add to end of sentence: and how to invoke the implementation in conforming mode.	The Committee considered the proposed change and found no consensus to adopt it.

US	17	TL	6.7.2.1	paragraphs 8 and 13	Anonymous structures and unions need minor clarification.	Changes along the lines of N 1549 should be adopted.	Accepted
US	18	E	7.25.1	5	Per N1372, thrd_timeout should be thrd_timedout	Replace thrd_timeout with thrd_timedout	Editorial - Accepted
US	19	TL	7.3, 7.15, 7.18		There are headers that define macros "complex", "bool", "alignas" for keywords "_Complex", "_Bool", "_Alignas" etc. But we could not find a header defining the macro "noreturn" for "_Noreturn". Nor could we find a header defining the macro "thread_local" for "_Thread_local". We think there should be.	Add header files along the lines of <stdbool.h> to define the noreturn and thread_local macros.	Accepted in principle - Add a new header file, `<stdnoreturn.h>`, for `_Noreturn`. Put `_Thread_local` in `<threads.h>`.
US	20	TL	6.4.1		Why is "alignof" a new keyword, instead of "_Alignof" with a header to define alignof macro? Seems needlessly inconsistent	Change the “alignof” keyword to “_Alignof” and add a header file along the lines of <stdbool.h> to define the alignof macro.	Accepted in principle - use header `<stdalign.h>`
US	21	GT	7.17.6	paragraph 1	It was never the intention to require that the atomic_* types be defined in terms of the _Atomic keyword, and this paragraph causes major problems with C++ compatibility. The atomic_* types must be implementable as structs so that they can serve as base classes for their atomic<*> counterparts.	For each line in the following table, the atomic type name behaves the same as the corresponding direct type. (NOTE: The atomic type name may be a typedef for the direct type, or it may be a struct.)	Accepted in principle - will adopt along the lines of the proposed solution. A change to require the same representation and alignment rather than requiring the same type.

US	22 GT 7.21		The locking behavior of I/O functions is not specified. This may result in unexpected behavior in multithread contexts and require explicit locking that will be redundant on most implementations.	Require implicit locking or provide for efficient explicit locking.	Accepted in principle - Insert the following paragraphs after 7.21.2p6: Each stream has an associated lock that can is used to prevent data races when multiple threads access a stream, and to restrict the interleaving of stream operations performed by multiple threads. Only one thread may hold this lock at a time. The lock is reentrant: A single thread may hold the lock multiple times at a given time. All functions that read, write, position, or query the position of a stream, except for putc_unlocked, getc_unlocked, putchar_unlocked, and getchar_unlocked, lock the stream, as though with flockfile before accessing it. They release the lock associated with the stream, as though with funlockfile, when the access is complete.

US	23	GT	7		The library section should be examined for threads incompatibilities. Obviously threads-incompatible functions include strtok and rand.	Thread safe versions should be included.	See N1551. The committee considered 5 items relative to this comment. Issue #1: strerror, strtok, rand and asctime. No Consensus to adopt this change. Issue #2: Replace wording for strtok, strerror, rand. Adopted the following wording: The <function> is not required to avoid data races with other calls to <function>. Substitute for <function>, strerror, strtok, and rand, respectively. Issue #3: Same as Issue 2 for rand and srand. Adopted the following words for each function: The rand and srand functions are not required to avoid data races with other calls to rand and srand." Issue #4: setjmp/longjmp Adopt the proposed wording in N1551, as modified by N1566. Issue #5: malloc/free Added the following words following 7.22.3;p1 For purposes of determining the existence of a data race, memory allocation functions behave as though they accessed only memory locations accessible through their arguments and not other static duration storage. These functions may however visibly modify the storage that they allocate or deallocate. A call to free or realloc that deallocates a region p of memory synchronizes with any allocation call that allocates all or part of the region p. This synchronization occurs after any access of p by the deallocating function, and before any such access by the allocating function. behave as though they accessed only
US	24	GT	7		The current mutex API is substantially different from both C++0x and Posix APIs. It is based on an API which currently has few direct clients.	At a minimum, the removal of mtx_try as in N1521 should be reconsidered.	Accepted with Modification. Removed `mtx_try()`. See N1521

US	25	GT	7		The _Atomic type qualifier should be reomoved. It is redundant, and its use needlessly hinders C++ compilation of code.	Remove _Atomic type qualifier.	The Committee considered the proposed change and found no consensus to adopt it.
CA	1	TE	Ge		We believe this function declaration is ambiguous is the current C draft: int func(_Atomic(int)) can mean a func that takes an atomic int or a function that takes a function that returns an atomic int and not a function that takes a function that returns an atomic int	_Atomic should not be a qualifier on the function return. If we remove the second meaning, then C++ can define _Atomic as a macro that expands to our template definition, and take the C++ symbols, and promote them to the global namespace.	The Committee considered the proposed change and found no consensus to adopt it. See US 25
CA	2	TE	B.16		Remove atomic_address in C1x.	We have removed atomic_address in C++0x. This was removed because it was a base class of the pointer specialization, which leads to no type safety.	Accepted
CA	3	TE	Ge		The current draft supports too many compound operations like atomic divide assign, atomic float for arithmetic operations. It is trying to be too general making every compound operators atomic. C++ selectively narrowed the operations based on what current hardware will not have trouble supporting.	Until we specify what they mean, what are the traps, we would prefer that C1x limits it to the same list as C++0x. Additional operations can be added. Original C1x paper implies that these operations can be written as if it is written with a compare exchange loop and that might work, but we need to understand it better. The limited set of operations that C++ supports is listed in Table 1 below these comments.	The Committee considered the proposed change and found no consensus to adopt it.

CA	4	TE	6.10.8.3		There is a current macro that says if you have stdc_no_threads, if that is defined, then you don't need to provide the stdcatomic.h header. These are different things. Specifically, threads belong to the OS and atomics belongs to the hardware. In embedded system, you want hardware support and not have OS come along for the ride.	Separate stdc_no_threads from std_atomic.h	Accepted in principle - split out atomics from `__STDC_NO_THREADS__`, and add `__STDC_NO_ATOMICS__`.
CA	5	TE	5.1.2.4		Remove atomic to atomic assignment.	C++0x has removed it because people may think the assignment is like transactional memory, but it is not.	No Change. The submitter provided more information, saying there was no problem, and essentially withdrew the comment.

CA	6	TE			Align C mutex types with C++ mutex types.	C++ mutex types were designed to make that compatibility possible. It will be embarrassing if we don't have the same mutex type. Originally, they were not placed probably because people did not want to assume a C syntax. Now that there is, this makes this argument moot. C mutex are local objects and while we may put wrapper around that because we require member functions, this will make condition variables fail to work with that. Condition variables only work with the C++ mutex type. If we further export these as inline functions, it also breaks down. We believe the C++ design leads to better performance, especially when we start scaling the system. [Hans and Lawrence may have some personal anecdotes and experience to back this up]. What is supplied by OS facility usually is too slow because it tries to be fair and does not scale well.	The Committee considered the proposed change and found no consensus to adopt it.

RU	1	ED	5.2.4.2.2	Page 30, paragraph 11	The phrase "The values given in the following list shall be replaced by constant expressions with implementation-defined values that are greater or equal in magnitude (absolute value) to those shown, with the same sign:" Should be replaced with something like "The values given in the following list shall be replaced by constant expressions with implementation-defined values: a) greater or equal in magnitude (absolute value) to those shown, with the same sign, if the shown values are greater than 1 in magnitude, or b) less or equal in magnitude (absolute value) to those shown, with the same sign, if the shown values are less than 1 in magnitude:" because constants with values less than 1 in magnitude (FLT_EPSILON, DBL_EPSILON, LDBL_EPSILON, FLT_MIN, DBL_MIN, LDBL_MIN, FLT_TRUE_MIN, DBL_TRUE_MIN, LDBL_TRUE_MIN) can be only decreased in conforming implementations.	To replace the current text "The values given in the following list shall be replaced by constant expressions with implementation-defined values that are greater or equal in magnitude (absolute value) to those shown, with the same sign:" with "The values given in the following list shall be replaced by constant expressions with implementation-defined values: a) greater or equal in magnitude (absolute value) to those shown, with the same sign, if the shown values are greater than 1 in magnitude, or b) less or equal in magnitude (absolute value) to those shown, with the same sign, if the shown values are less than 1 in magnitude"	Editorial - No chance, this is a misreading of the Standard.

NL	1	TE	6.7.5		Comment on Section 6.7.5 - Alignment specifier It would be 'natural', certainly for a language like C, if the alignment specification is part of the type specification and not, as proposed, as part of the declaration specifier. The proposed _Alignas specifier prevents the proper propagation and use of alignment information in the compiler. The argument for the current choice is that the cost of taking the type specifier approach would be very costly for C++; we do not consider this to be a valid argument: in many other places a difference between C and C++ is justified by the reasoning that C and C++ are two different languages, each with their own users and application areas, so why is it so necessary that in the _Alignas case the languages are the same		The Committee considered the proposed change and found no consensus to adopt it at this time.

NL	2	TE	Annex F		Comment on Annex F - IEC 60559 floating-point arithmetic This (normative) section refers to IEC 60669:1989, while there is a new version of this standard by the summer of 2011 (well in advance of adoption of the C1X standard). C1X must refer to the new floating-point standard. Separate question: is it the intention to include the exchange formats (especially binary16 - half float) as a fully required data type once the new new floating-point standard is referenced? If not, should this be added to the Embedded-C specification as there is some interest in this in the embedded C world?		The Committee considered the proposed change and found no consensus to adopt it for this revision However there is a Study Group withing WG14 looking at this issue with plans to create a C binding to the new IEEE Standard as a Technical Specification in the future.

BSI	1	TE	5.1.2.3#5	5	5.1.2.3#5 describes parts of program state when a signal occurs. However, it is defined in terms of objects, which does not cover the floating-point environment. Depending on the operating system, the floating-point environment on receipt of a signal may be set to a default environment or it may be the environment in effect when the signal was delivered; the latter may not be a state that was ever in effect in the abstract machine because code sequences for some operations may change the rounding mode temporarily, then restore it. It seems best to leave the choice explicitly unspecified. (This means signal handlers cannot reliably use floating point; if that is to be permitted, feholdexcept and fesetenv would need to be documented as safe to call from signal handlers.)	In 5.1.2.3, insert ", as is the floating-point environment (7.6)" after "unspecified", and insert ", as does the floating-point environment if it is modified and not restored before exit from the handler" after "undefined".	Accepted

BSI	2	TE			There are some places where alignof needs to be handled similarly to sizeof, for consistency and to reflect existing practice, but with appropriate adjustments for when VLA size expressions are involved	In 6.5.3.4#3, change the second sentence to "Expressions in the operand are not evaluated, and the result is an integer constant.". In 6.6#3, footnote 115, change "sizeof" to "sizeof or alignof". In 6.6#6, insert "alignof expressions," before "sizeof expressions", and change "sizeof operator" to "sizeof or alignof operator". In 6.6#8, change "and sizeof expressions" to "alignof expressions, and sizeof expressions", and change "a sizeof operator" to "an alignof operator or a sizeof operator". In 6.9#3, change "a sizeof operator" to "an alignof operator or a sizeof operator". In 6.9#5, change "a sizeof operator" to "an alignof operator or a sizeof operator".	Accepted

BSI	3	TE	6.7#3	3	6.7#3 says "a typedef name can be redefined to denote the same type as it currently does", and redefining otherwise is a constraint violation, but in the case of VLAs it may not be known until runtime whether the types will in fact be the same. The suggested solution of diagnosing that a violation at runtime is possible should be stated in a footnote.	In 6.7#3, after "same type as it currently does" add a footnote "If identity of the types depends on the values of variable length array size expressions, the implementation may generate a diagnostic that a constraint violation could occur depending on the values at runtime.".	Accepted in principle - 6.7;p3. Change: If an identifier has no linkage, there shall be no more than one declaration of the identifier (in a declarator or type specifier) with the same scope and in the same name space, except that a typedef name can be redefined to denote the same type as it currently does and tags may be redeclared as specified in 6.7.2.3. To: If an identifier has no linkage, there shall be no more than one declaration of the identifier (in a declarator or type specifier) with the same scope and in the same name space, except: * a typedef name can be redefined to denote the same type as it currently does if that type is not a variably modified type * tags may be redeclared as specified in 6.7.2.3.
BSI	4	TE	6.7.#5	5	6.7#5 defines a "definition" of an identifier, saying that for an enumeration constant or typedef name it is "the (only) declaration of the identifier". The "(only)" is no longer accurate now typedef redefinition is allowed; it seems natural to say that the first declaration in such a case is the definition (an alternative would be to say that all are definitions).	In 6.7#5, replace the last bullet point with two bullet points: • for an enumeration constant, is the (only) declaration of the identifier; • for a typedef name, is the first or only declaration of the identifier.	Accepted

BSI	5	TE	6.7.1		6.7.1 is missing a constraint that _Thread_local may not be used on a function declaration. (This usage makes no sense and disallowing it is existing GNU __thread practice. For function definitions this is already disallowed by 6.9.1#4 but it should also be disallowed for declarations that are not definitions.)	In the Constraints in 6.7.1, add a new paragraph after paragraph 3: "_Thread_local may not be present in the storage class specifiers in a declaration of a function.".	Accepted

BSI	6	TE	6.7.2.1#18		6.7.2.1#18 says "As a special case, the last element of a structure with more than one named member may have an incomplete array type; this is called a flexible array member.". It should be made clear that this allows structures where all previous members are anonymous structures or unions, by virtue of 6.7.2.1#13.	At the end of 6.7.2.1, add a new example:
						Because elements of anonymous structures and unions are considered to be members of the containing structure or union, the following example has more than one named member and is a valid use of a flexible array member:
						struct s
						{
						  struct
						  {
						  int i;
						  };
						  int a[];
						};
							Accepted

BSI	7	TE	6.7.9#15	15	6.7.9#15 says "An array with element type compatible with a qualified or unqualified version of wchar_t may be initialized by a wide string literal, optionally enclosed in braces. Successive wide characters of the wide string literal (including the terminating null wide character if there is room or if the array is of unknown size) initialize the elements of the array.". But 6.4.5 now defines wide string literals to include char16_t and char32_t literals, and the initialization wording needs updating to allow each kind of wide string literal to initialize the associated kind of array.	Change 6.7.9#15 to read "An array with element type compatible with a qualified or unqualified version of wchar_t, char16_t or char32_t may be initialized by a wide string literal, optionally enclosed in braces. The wide string literal must have array element type (as defined in 6.4.5) compatible with the unqualified version of the element type of the array being initialized. Successive elements of the array specified in 6.4.5 for the wide string literal (including the terminating null element if there is room or if the array is of unknown size) initialize the elements of the array."	Accepted
BSI	8	TE	6.10.9#1		6.10.9#1 refers to removal of the L prefix, if present, from a string literal inside _Pragma. This should now handle the new types of string prefixes added in C1X.	In 6.10.9#1, change "L prefix" to "u8, u, U or L prefix	Accepted

BSI	9	TE	7.1.2#4	4	7.1.2#4 says "The program shall not have any macros with names lexically identical to keywords currently defined prior to the inclusion.". There is however a related issue that this does not address: a macro lexically identical to a keyword could be defined after the standard header is included, but with the definition being in effect when a macro defined in the standard header is expanded, and the expansion of the macro in the standard header could use the keyword that is defined as a macro. Thus, either such definitions of keywords as macros should be disallowed whenever a macro from a standard header is expanded, or all macro definitions in standard headers need to use alternative implementation-specific keywords in the reserved namespaces such as __void. In the latter case, examples in the C standard such as the required definition of assert in 7.2#1, the possible definition of the cbrt type-generic macro in 6.5.1.1#5 and the possible definitions of CMPLX, CMPLXF and CMPLXL in 7.3.9.3#5 should not show the use of keywords outside the reserved namespaces. (The Rationale (pages 100 and 101 in version 5.10) discusses uses for defining keyword names as macros, but	In 7.1.2#4, add "or when a macro defined in a standard header is expanded" at end of last sentence.	Accepted
BSI	10	ED	7.19		7.19 has a forward reference to 7.11. This is actually a backward reference. (In C90 it was genuinely a forward reference from 7.1.6 to 7.4.)	At the end of 7.19, remove "Forward references: localization (7.11)."	Editorial - Accepted

BSI	11	TE	7.19		It seems clear from the standard text that the scanf %% format is required to skip white-space in the input stream: that %% acts differently from single ordinary characters in the format string and you need to use %1[%] to match just a single % without white-space. However, implementations differ in this regard, so it would be useful to add an example to make this clearer to implementors.	In 7.21.6.2, add another example:
						"The program
						#include <stdio.h>
						
						int main (void)
						{
						  int dummy;
						  return sscanf ("foo \t %bar1",
						"foo%%bar%d", &dummy);
						}
						returns status 1, not 0, because input white-space is skipped when matching %%."
							Accepted

BSI	12	TE	7.25.1#4 and 7.26.1#4		7.25.1#4 says that xtime "holds a time specified in seconds and nanoseconds", and has members "time_t sec;" and "long nsec;". But time_t is not required to count in seconds; 7.26.1#4 says "The range and precision of times representable in clock_t and time_t are implementation-defined.". time_t may count in units other than seconds; it may be a floating-point type, so if it counts in seconds it may have subsecond resolution; it may not bear a linear relation to elapsed time. The xtime type is used by cnd_timedwait, mtx_timedlock and thrd_sleep, and set by xtime_get. xtime_get creates a valid xtime value, which apparently is to be interpreted in accordance with the base argument; the other functions use such a value, and do not have any base parameter to describe the interpretation. The description of the base argument refers to TIME_UTC, but the list of macros defined in this header does not include TIME_UTC. I don't believe it makes sense to have the base argument to xtime_get, given that the semantics of time_t values (which must be the basis for those of xtime values) do not depend on any such value, and the only way to modify a time_t value to get a valid future time, and so a valid future xtime value,	Proposed change 1: In 7.25.1#4, change "holds a time specified in seconds and nanoseconds" to "holds a time specified as a nanosecond offset from a time_t value", with a footnote "Although the time_t value is given as time_t sec;, time_t does not necessarily count in seconds.". Proposed change 2: In 7.25.7.1#1, remove the "int base" argument. In 7.25.7.1#2, remove "based on the time base base". In 7.25.7.1#3, change "the nonzero value base, which must be TIME_UTC" to "a nonzero value". In Annex B.24, remove the "int base" argument to xtime_get.	Accepted with Modification. See N1564
BSI	13	TE	7.26.1#3		It appears 7.26.1#3 allows time_t and clock_t to be complex types. I see no good reason for this to be permitted.	In 7.26.1#3, change "arithmetic types" to "real types".	Accepted

BSI	14	TE	3.7.3 and 7.28.1		When are wide string library functions required to handle values of type wchar_t that do not represent any value in the execution character set, and when does using such values with a library function result in undefined behavior? This issue was raised directly and through the Austin Group; the Batavia minutes say "We are taking no action here" (N1541 6.31 item 1) but this still leaves the standard unclear. The definition of "wide character" in 3.7.3 is "bit representation that fits in an object of type wchar_t, capable of representing any character in the current locale". I interpret the part after the comma as being descriptive of the type wchar_t, rather than constraining the definition of "wide character". That is, "wide character" includes all bit representations that fit in type wchar_t, whether or not they represent valid members of the execution character set. The first problem here would seem to be the possible inclusion of trap representations; it seems better for only representations that represent values of type wchar_t to count as wide characters, and for only the integer value to be significant. That is, a wide character should be a value of type wchar_t, not a bit representation. In turn, 7.1.1#4 defines a "wide string" to include all null-terminated sequences	Proposed change 1: In 3.7.3, change "bit representation that fits in" to "value representable by". Proposed change 2: In 7.28.1, add a new paragraph before paragraph 5: "Arguments to the functions in this subclause may point to arrays containing wchar_t values that do not correspond to members of the extended character set. Such values shall be processed according to the specified semantics, provided that it is unspecified whether an encoding error occurs if such a value occurs in the format string for a function in 7.28.2 or 7.28.5 and the specified semantics do not include passing the wide character through wcrtomb."	Accepted

BSI	15	TE	7.30		There are what appear to be namespaces (explicitly reserved or otherwise) used by various headers that are not listed in 7.30 but should be.	Proposed change 1: Between 7.30.3 and 7.30.4, add a subclause for <fenv.h>: "Macros that begin with FE_ and an uppercase letter may be added to the definitions in the <fenv.h> header.". Add footnotes referencing this new subclause to the sentences referring to such macros in 7.6#6, 7.6#8 and 7.6#10. Proposed change 2: Between 7.30.6 and 7.30.7, add a subclause for <stdatomic.h>: "Macros, function names, typedef names and enumeration values that begin with ATOMIC_, atomic_ or memory_ may be added to the <stdatomic.h> header.". Proposed change 3: Between 7.30.11 and 7.30.12, add a subclause for <threads.h>: "Function names, typedef names and enumeration values that begin with cnd_, mtx_, thrd_ or tss_ may be added to the <threads.h> header.".	Accepted

BSI	16	TE	J2		J.2 lists (bottom of page 563) "The number of characters transmitted by a formatted output function is greater than INT_MAX (7.21.6.1, 7.21.6.3, 7.21.6.8, 7.21.6.10).". This is missing the wide character functions and the functions that output to strings instead of files; all of these have the same issue that there may be return values specified by the semantics that cannot be represented in the int return type. (A similar issue applies to functions in Annex K. I have not tried to propose a fix there, though making the overflow cases into runtime-constraint violations may make sense. The asprintf-family functions in TR 24731-2 also have this problem.)	In the last item on page 563, change "characters" to "characters or wide characters" and change "transmitted" to "transmitted, written to a string, or that would be written to a string has the array size parameter been large enough". Add 7.21.6.5, 7.21.6.6, 7.21.6.12, 7.21.6.13, 7.28.2.1, 7.28.2.3, 7.28.2.5, 7.28.2.7, 7.28.2.9, 7.28.2.11 to the list of subclauses in that item.	Accepted
BSI	17	TE	J2		scanf-family functions may have a format string with more than INT_MAX conversion specifiers. J.2 should list undefined behavior if one of these functions would need to return a value greater than INT_MAX. (A similar issue applies to functions in Annex K. I have not tried to propose a fix there, though making the overflow cases into runtime-constraint violations may make sense.)	Add to J.2 an item "The number of input items assigned by a formatted input function is greater than INT_MAX (7.21.6.2, 7.21.6.4, 7.21.6.7, 7.21.6.9, 7.21.6.11, 7.21.6.14, 7.28.2.2, 7.28.2.4, 7.28.2.6, 7.28.2.8, 7.28.2.10, 7.28.2.12).".	Accepted
BSI	18	TE	J.5.6#1	1	In view of the binary16 format in IEEE 754-2008, J.5.6 should explicitly note the possibility of additional floating types having less range and precision than float.	In J.5.6#1, add "or less range and precision than float" after "long double".	Accepted

BSI	19	TE			There are several instances of undefined behavior that are intrinsically hard for implementations to bound, by reason of the ABIs in use in practice or the limitations of hardware. These should be added to the list of critical undefined behavior in L.3#2. Specifically: • Modifying constant objects should be considered equivalent to operations using invalid pointers. • The problems with invalid arguments to library functions also apply to symbols such as va_arg specified to be macros. • Incompatible types, where not constraint violations, generally cannot be diagnosed without information often not available at link time, and if (say) one translation unit declares an object with a type occupying more memory than another translation unit defining the object, accesses from the first translation unit vill be like using invalid pointers.	Proposed change 1: In the list in L.3#2, add "The program attempts to modify a string literal (6.4.5).". Proposed change 2: In the list in L.3#2, add "An attempt is made to modify an object defined with a const-qualified type through use of an lvalue with non-const-qualified type (6.7.3).". Proposed change 3: In the list in L.3#2, change "library function" to "library function or macro". Proposed change 4: "Two declarations of the same object or function specify types that are not compatible (6.2.7).".	Accepted with Modification, See N1568. Proposed change 1: Accepted as written. Proposed change 2: Accepted as written. Proposed change 3: In L.3#2, change "An argument to a library function . . ." to "An argument to a function or macro defined in the standard library . . ." Proposed change 4: Part A: Change "(6.3.2.3)" to "(6.3.2.3, and see 6.2.7)". Part B: "A store is performed to an object that has two incompatible declarations (6.2.7)."