

Doc. No.: X3J16/95-0178  
WG21/ N0778  
Date: September 25, 1995  
Project: Programming Language C++  
Reply To: Richard K. Wilhelm  
Strategic Technology Resources  
rwillhelm@str.com

## Clause 21 (Strings Library) Issues List Revision 8

### Revision History

Version 1 - January 30, 1995: Distributed in pre-Austin mailing.

Version 2 - March 6, 1995: Distributed at Austin meeting.

Version 3 - March 24, 1995: Distributed in post-Austin mailing. Several issues added. Several issues updated to reflect decisions at Austin meeting.

Version 4 - May 19, 1995: Distributed in pre-Monterey mailing.

Version 5 - July 9, 1995: Distributed at the Monterey meeting. Includes many issues added from public comments.

Version 6 - July 11, 1995: Distributed at the Monterey meeting. Added no new issues from previous version. Included issues prepared for formal vote. Added solutions for issues 8, 21,31, 38, 69, 71. Made only changes to reflect the decisions of the string sub-group, correct working paper text and to correct typographical errors.

Version 7 - July 27, 1995: Distributed in the post-Monterey mailing. Reflects the resolutions and discussions of the Monterey meeting.

Version 8 - September 24, 1995: Distributed in the pre-Tokyo mailing. Some new issues added.

### Introduction

This document is a summary of the issues identified in Clause 21. For each issue the status, a short description, and pointers to relevant reflector messages and papers are given. This evolving document will serve as a basis of discussion and historical record for Strings issues and as a foundation of proposals for resolving specific issues.

For clarity, active issues are separated from issues recently closed. Closed issues are retained for one revision of the paper to serve as a record of recent resolutions. Subsequently, they will be removed from the paper for brevity. Any issue which has been removed will include the document number of the final paper in which it was included.

### Active Issues

#### Issue Number: 21-002

Title: Are string\_traits members char\_in() and char\_out() necessary?

Section: 21.1.1.2 [lib.string.char.traits]

Status: active

Description:

In lib-3398, Nathan Myers writes:

Looking at Clause 21, Strings, I find some string\_traits static members:

```
static basic_istream<charT>
    string_char_traits::char_in(basic_istream<charT>& is,
```

```
charT& a)
{ return is >> a; }

static basic_istream<charT>
  string_char_traits::char_out(basic_ostream<charT>& os,
                               charT& a)
{ return os << a; }
```

Are they necessary? If so, shouldn't they be parameterized on ios\_traits? And shouldn't they default to use streambuf put() and get()?

[Note: lib-3398 contained a typo in which char\_in() and char\_out() were incorrectly specified as being members of basic\_string. The slight error is corrected above.]

**Proposed Resolution:**

Remove the members string\_char\_traits::char\_in() and string\_char\_traits::char\_out().

**Requester:** Nathan Myers: myersn@roguewave.com  
**Owner:**  
**Emails:** lib-3398  
**Papers:** (none)

**Issue Number: 21-013**

**Title:** There is no provision for errors caused by implementation limits.  
**Section:** 21.1.1.2 [lib.basic.string]  
**Status:** active  
**Description:**

In private email, John Dlugosz wrote:

"There is no provision for errors caused by implementation limits. The class handles strings up to length NPOS-1, with no specified way to throw an error saying "I can't do that!" for shorter values. In my implementation I'm simulating an out-of-memory error if an operation exceeds a `maxcount' length, since that's what would presumably happen anyway. The maxcount arises due to arithmetic overflow: I'm limited to size\_t-(small constant) \_bytes\_, not elements, and an element may be any size. I can't compute the memory requirements without getting an unreported arithmetic overflow, so I have to check in advance for this instantiation-specific maxcount.

"In order to simulate the out of memory condition, I just call `new' on NPOS bytes. That way I get the "correct" behavior for any installed new\_handler or replacement operator new() that may exist. However, that is not the best solution for a few reasons. First, it will fail if the implementation \_does\_ in fact allocate NPOS bytes without error. Second, an out-of-memory exception might not be the appropriate way for a program to recover from this problem. Third, it is less efficient, since by spec I must test for an argument of NPOS anyway, and take one action and \_then\_ test for the smaller maxcount and take another action. To summarize, I think that a "length error" should be allowed at an implementation defined size limit which is less than or equal to NPOS. There should also be a function available to return this value."

**Proposed Resolution:**

**Requester:** John Dlugosz: jdlugosz@objectspace.com  
**Owner:**  
**Emails:** (none)  
**Papers:** (none)

**Issue Number: 21-014**

Title: Argument order for copy() is incorrect.  
Section: 21.1.1.8.7 [lib.string::copy]  
Status: active  
Description:

In private email, John Dlugosz wrote:  
“In copy() the arguments are in a different order than on other functions. I suppose this was to provide for a default on pos. However, if someone does specify both he will be likely to get them backwards and the compiler will not catch this. I feel it is a point of usability that is not worth the default argument. Provide two forms of copy() instead:  
copy (dest, pos, len);  
copy (dest, len);

Note: The current interface to copy is:  
size\_type copy(charT\* s, size\_type n, size\_type pos=0);

Proposed Resolution:

Provide two forms of copy():  
size\_type copy(charT\* s, size\_type pos, size\_type n);  
This function differs from the current copy only in the order of its last two arguments and the lack of a default argument.  
size\_type copy(charT\* s, size\_type n);  
Returns:  
copy(s, 0, n);

.Requester: John Dlugosz: jdlugosz@objectspace.com  
Owner:  
Emails: (none)  
Papers: (none)

**Issue Number: 21-017**

Title: Can reserve() cause construction of characters?.  
Section: 21.1.1.6 [lib.string.capacity]  
Status: active  
Description:

In private email, John Dlugosz wrote:  
“Also, totally unspecified, is the treatment of the `reserve' area with respect to element creation and destruction. I chose to construct elements in the reserve area right away, and then the string grows into the reserve area using assignment semantics. This causes dramatic simplification in several areas, and allows me to implement it without the need for in-place construction and explicit destructor calls (important when targeting cfront-based compilers).”

Proposed Resolution:

No change required.

Requester: John Dlugosz: jdlugosz@objectspace.com  
Owner:  
Emails: (none)  
Papers: (none)

**Issue Number: 21-018**

Title: Specification of traits class is constraining.  
Section: 21.1.1.2 [lib.string.char.traits]  
Status: active

Description:

In private email, John Dlugosz wrote:

“The austerity of the traits class strongly suggests certain implementations and prevents certain optimizations. For a simple example, the `copy()` function does not provide for overlapping copies. Say I have a string "ABr" where A and B represent substrings of some length, and r is unused reserve area. I want to insert "C" into the string, and the length of "ACB" fits into the pre-existing allocation (because C is shorter or equal in size to r). I can't just copy B down to the tail end. Instead, I have to reallocate the whole string and copy the A part also.

“More significantly, the `find()` functions pretty much have to be implemented by a brute-force approach as they are defined-- locate a place where the match occurs. In short, I wish the traits available were richer. It seems inconsistent w.r.t. copy semantics, as explained in [issue 23-017], and it is so simple as to force inefficiencies in the implementation. In addition, it would be nice if additional implementation-specific stuff could be placed in the traits class. This can be done and still allow for user-defined "custom" traits to be created that only have the standard members, by using inheritance.”

Proposed Resolution:

Requester: John Dlugosz: [jdlugosz@objectspace.com](mailto:jdlugosz@objectspace.com)

Owner:

Emails: (none)

Papers: (none)

**Issue Number: 21-024**

Title: Name of traits delimiter function is confusing

Section: 21.1.1.1 [lib.string.char.traits]

Status: active

Description:

The name of the `string_char_traits` function is "is\_del". This has the connotation of "is delete".

Proposed Resolution:

Remove this member. These sorts of traits are the domain of `iostreams`.

Requester: John Hinke: [jhinke@qds.com](mailto:jhinke@qds.com)

Owner:

Emails: (none)

Papers: (none)

**Issue Number: 21-025**

Title: Does `string_char_traits` need a locale?

Section: 21.1.1.2 [lib.string.char.traits.members]

Status: active

Description:

The description of the member `string_char_traits::is_del()` says it returns: `isspace()`. This function is subject to localization. Does this mean that `string_char_traits` is locale sensitive?

Proposed Resolution:

Remove the `is_del()` member. Do not add a locale.

Requester: John Hinke: [jhinke@qds.com](mailto:jhinke@qds.com)

Owner:

Emails: (none)  
Papers: (none)

**Issue Number: 21-026**

Title: Description of `string_char_traits::compare()` is expressed in code.  
Section: 21.1.1.2 [lib.string.char.traits.members]  
Status: active  
Description:

The description of the `string_char_traits` member:  

```
static int compare(const char_type* s1, const char_type* s2,  
                  size_t n);
```

is expressed in code as follows:

```
for (size_t i=0; i<n; ++i, ++s1, ++s2)  
    if (ne(*s1, *s2))  
        return (lt(*s1, *s2) ? -1 : 1;  
return 0;
```

It should be expressed in prose.

Proposed Resolution:

Replace the description with the following:

Returns: 0 iff for each  $i$ :  $0 < i < n$  the expression `eq(*(s1+i), *(s2+i))` is true.  
Otherwise, returns -1 given  $i$  and  $j$  such that for  $j$ :  $0 \leq j < n$ , the expression `lt(*(s1+j), *(s2+j))` is true and for each  $i$ :  $0 < i < j$  the expression `eq(*(s1+i), *(s2+i))` is true.  
Otherwise returns 1.

Requester: Rick Wilhelm: rkw@chi.andersen.com  
Owner:  
Emails: (none)  
Papers: (none)

**Issue Number: 21-027**

Title: Description of `string_char_traits::compare()` overspecifies return value.  
Section: 21.1.1.2 [lib.string.char.traits.members]  
Status: active  
Description:

The description of the `string_char_traits` member:  

```
static int compare(const char_type* s1, const char_type* s2,  
                  size_t n);
```

is expressed in code as follows:

```
for (size_t i=0; i<n; ++i, ++s1, ++s2)  
    if (ne(*s1, *s2))  
        return (lt(*s1, *s2) ? -1 : 1;  
return 0;
```

Specifying the exact return values when the comparison returns “less than” or “greater than” is too constraining.

Proposed Resolution:

Replace “-1” with “an integer less than zero” and replace “1” with “an integer greater than 0”

Requester: Rick Wilhelm: rkw@chi.andersen.com  
Owner:  
Emails: (none)  
Papers: (none)

**Issue Number: 21-028**

Title: Description of `string_char_traits::length()` is expressed in code.  
Section: 21.1.1.2 [lib.string.char.traits.members]

Status: active

Description:

The description of the `string_char_traits` member:

```
static int length(const char_type* s);
```

is expressed in code as follows:

```
size_t len = 0;
while (ne(*s++, eos())) ++len;
return len;
```

It should be expressed in prose.

Proposed Resolution:

Replace the description with the following:

Returns: the lowest value of  $i$  such that for  $i, i \geq 0$ , the expression `ne(*(s+i), eos())` returns false and for each  $j, 0 \leq j \leq i$  the expression `ne(*(s+j), eos())` returns true and.

Requester: Rick Wilhelm: rkw@chi.andersen.com

Owner:

Emails: (none)

Papers: (none)

**Issue Number: 21-029**

Title: Description of `string_char_traits::copy()` is overconstraining.

Section: 21.1.1.2 [lib.string.char.traits.members]

Status: active

Description:

The description of the member `string_char_traits::copy()`

```
char_type* s = s1;
for (size_t i=0; i<n; ++i) assign(*s1++, *s2++);
```

This overconstrains implementations, in that there is no particular reason to do the operations in the order specified. (Box 78).

Proposed Resolution:

Replace the description as follows:

Effects: Copies elements. For each non-negative integer  $i < n$ , performs

$*(s1 + i) = *(s2 + i)$ .

Returns:  $s1$ .

Requires:  $s1$  shall not be in the range  $[s2, s2+n)$ .

Requester: Rick Wilhelm: rkw@chi.andersen.com

Owner:

Emails: (none)

Papers: (none)

**Issue Number: 21-030**

Title: Description of `string_char_traits::copy()` is silent on overlapping strings.

Section: 21.1.1.2 [lib.string.char.traits.members]

Status: active

Description:

The description of the member `string_char_traits::copy()`

```
char_type* s = s1;
for (size_t i=0; i<n; ++i) assign(*s1++, *s2++);
```

Doesn't explicitly address the issue of overlapping strings.

Proposed Resolution:

Add the following to the description of `string_char_traits::copy()`:

Requires:  $s1$  shall not be in the range  $[s2, s2+n)$ .

This is similar to the approach followed by `copy()` in 25.2.1 [lib.alg.copy].

Requester: Rick Wilhelm: rkw@chi.andersen.com

Owner:  
Emails: (none)  
Papers: (none)

**Issue Number: 21-031**

Title: Copy constructor takes extra argument to switch allocator but does not allow allocator to remain the same.  
Section: 21.1.1.4 [lib.string.cons]  
Status: active  
Description:

The copy constructor:

```
basic_string(  
    const basic_string<charT, traits, Allocator>& str,  
    size_type pos = 0, size_type n = npos,  
    Allocator& = Allocator());
```

takes an extra argument, so that it can be used to copy a string while changing its allocator. Is this the best way to do this? (Box 79).

This copy constructor does not allow the user to retain the same allocator as the current string. Additionally, the string class does not provide a member to access a string's allocator.

**Proposed Resolution:**

The solution to this issue exactly mirrors the solution to a general containers issue.

At the Monterey meeting, the following change was approved and inserted into the WP:

In section 21.1.1.9 [lib.string.ops], add the member:

```
const allocator_type& get_allocator() const;
```

Returns: a reference to the string's allocator object.

The resolution to the default Allocator argument is pending the resolution to a similar issue in Clause 23.

Requester: Rick Wilhelm: rkw@chi.andersen.com. See also public comment T21 (p. 108)  
Owner:  
Emails: (none)  
Papers: (none)

**Issue Number: 21-034**

Title: Inconsistency in requirements statements involving npos  
Section: 21.1.1.4 [lib.string.cons] and 21.1.1.6 [lib.string.capacity]  
Status: active  
Description:

In the current draft, the requirements for  
`basic_string(size_type n, charT c, Allocator& = Allocator());`  
read:

Requires:  $n < npos$ .

and the requirements for

```
void resize(size_type n, charT c);
```

read:

Requires:  $n \neq npos$ .

These should be expressed in terms of `max_size()`

**Proposed Resolution:**

Change the description of `resize()`:

Requires: `n <= max_size()`  
Throws: `length_error` if `n > max_size()`

Requester: Rick Wilhelm: [rkw@chi.andersen.com](mailto:rkw@chi.andersen.com) See also public comment T21 (p. 109)  
Owner:  
Emails: (none)  
Papers: (none)

**Issue Number: 21-034a**

Title: Expand ability to throw `length_error`  
Section: 21.1.1.3 [lib.basic.string]  
Status: active  
Description:

The specification carefully dictates that a string should be able to hold the number of entities indexed by a `size_type`. This is evidenced, for example, in the strict specification of when a `length_error` exception is thrown in `basic_string::replace`.

Strictly interpreted, this prevents storage of other information in the same memory block as the data (e.g., reference counts of string lengths). It should be possible to throw a `length_error` when the resulting data size \*plus the size of the overhead information\* exceeds the capacity of a `size_type`.

It may be convenient to specify `length_error` conditions in terms of the `max_size()` value.

Proposed Resolution:

Requester: Judy Ward: [ward@roguewave.com](mailto:ward@roguewave.com)  
Owner:  
Emails: (none)  
Papers: (none)

**Issue Number: 21-037**

Title: Traits needs a `move()` for overlapping copies.  
Section: 21.1.1.4 [lib.string.cons]  
Status: active  
Description:

A `move()` member for overlapping copies would be a useful addition to the `string_char_traits` class.

Proposed Resolution:

Requester: Judy Ward: [ward@roguewave.com](mailto:ward@roguewave.com)  
Owner:  
Emails: (none)  
Papers: (none)

**Issue Number: 21-059**

Title: String traits have no relationship to `iostream` traits.  
Section: 21.1.1.1 [lib.string.char.traits]  
Status: active  
Description:

I would like to propose (whether officially or not) to modify the current CD:

```
template <class charT> struct ios_traits {};  
to  
template <class charT> struct ios_traits :  
    public string_char_traits<charT> {};
```

in order to integrate the closely related traits, 'ios\_traits' and 'string\_char\_traits'.

We can expect the integration of the common features, such as 'eq', 'eos', 'length', and 'copy' which is now inappropriately separated with no explicit reasons.

In lib-3832, Nathan Myers wrote:

“I have been careful to avoid getting too involved with Clause 21, thus far, because I have been quite busy with other chapters. However, it would be my recommendation to eliminate most of the string character traits: eq(), ne(), lt(), assign(), char\_in(), char\_out(), and is\_del(). Also, I would either add a few "speed-up functions" needed to efficiently implement strings without specialization, such as a move() member, or eliminate them all, and let the implementation specialize speedups for types known to it.”

A public comment included the following:

“string\_char\_traits is missing three important speed-up functions, the generalizations of memchr, memmove, and memset. Nearly all the mutator functions in basic\_string can be expressed as calls to these three primitives, to good advantage.”

See also issue 21-018.

**Proposed Resolution:**

More detailed work needed on this topic.

**Requester:** Norihiro Kumagai: kuma@slab.tnr.sharp.co.jp.  
See also Public Comment T21 (p. 108).

**Owner:**

**Emails:** lib-3832

**Papers:** (none)

**Issue Number: 21-060**

**Title:** string\_char\_traits::ne not needed

**Section:** 21.1.1.1 [lib.string.char.traits]

**Status:** active

**Description:**

A public comment included:

“string\_char\_traits::ne is hardly needed given the member eq. It should be removed.

**Proposed Resolution:**

Remove the member string\_char\_traits::ne().

**Requester:** Public comment T21 (p. 107)

**Owner:**

**Emails:** (none)

**Papers:** (none)

**Issue Number: 21-061**

**Title:** Missing explanation of traits specialization

**Section:** 21.1.1.2 [lib.string.char.traits.members]

**Status:** active

**Description:**

A public comment noted:

“No explanation is given for why the descriptions of the members of template class string\_char\_traits are “default definitions.” If it is meant to suggest that the

program can supply an explicit specialization, provided the specialization satisfies the semantics of the class, then the text should say so (here and several other places as well).

Proposed Resolution:

Requester: None.  
Owner: Public comment T21 (p. 108).  
Emails: (none)  
Papers: (none)

**Issue Number: 21-062**

Title: Missing explanation of requirements on charT.  
Section: 21.1.1.3 [lib.basic.string]  
Status: active  
Description:

A public comment noted:  
Paragraph 1 doesn't say enough about the properties of a "char-like object." It should say that it doesn't need to be constructed or destroyed (otherwise, the primitives in string\_char\_traits are woefully inadequate).  
string\_char\_traits::assign (and copy) must suffice either to copy or initialize a char-like element. The definition should also say that an allocator must have the same definitions for the types size\_type, difference\_type, pointer, const\_pointer, reference, and const\_reference as class allocator::types<charT> (again because string\_char\_traits has no provision for funny address types).

Proposed Resolution:

Requester: Public comment T21 (p. 108).  
Owner:  
Emails: (none)  
Papers: (none)

**Issue Number: 21-063**

Title: No constraints on constructor parameter.  
Section: 21.1.1.4 [lib.string.cons]  
Status: active  
Description:

The description of the constructor  
`basic_string(const charT* s, size_type n, Allocator&);`  
Doesn't constrain the size\_type parameter.

Proposed Resolution:

Modify the description of the constructor as follows:  
Requires: s shall not be a null pointer and n != npos.  
Throws: length\_error if n == npos

Requester: Public comment T21 (p. 108)  
Owner:  
Emails: (none)  
Papers: (none)

**Issue Number: 21-067**

Title: Traits specializations are over-constrained for eos() member  
Section: 21.1.1.2 [lib.string.char.traits.members]  
Status: active  
Description:

The current description is:

Returns: The null character, `char_type()`

However, if the traits are specialized, the specialization should not be required to return the result of the default constructor.

Proposed Resolution:

Change the description to be:

Returns: The null character.

Requester: Public comment T21 (p. 108).

Owner:

Emails: (none)

Papers: (none)

**Issue Number: 21-068**

Title: What is the proper role of the “Notes” section in Clause 21.

Section: 21.1.1.6 [lib.string.capacity] (and several other sections in the clause)

Status: active

Description:

Clause 21 currently contains several sections which include the text:

Notes:

The draft already says that notes are non-normative. However, the contents of these sections are often normative. Should the contents of these sections be moved into other sections.

Also, the Notes sections currently give information on the use of some traits. The Japanese delegation would like to see information on the use of traits expanded to give the user more information about the impact of traits on the string template. However, one public comment described these sorts of notes on traits as over-specification.

Proposed Resolution:

Change all instances of “Notes” sections to conform to the draft convention for notes as specified in [intro.compliance], with the exception of the following instances:

- 21.1.1.6 [lib.string.capacity], notes on `reserve()` which discuss the invalidation of references and guarantees on reallocation.
- 21.1.1.7 [lib.string.access], notes on `operator[]` which discuss the validity of references.
- 21.1.1.8.7 [lib.string::copy], notes on `copy()` which discuss the absence of a null object at the end of the string.

In these three cases, the text should be moved to the “Required Behavior” section. Note: this solution implies that all notes which deal with the use of traits members are non-normative.

Requester: Public comment T21 (p. 108).

Owner:

Emails: (none)

Papers: (none)

**Issue Number: 21-074**

Title: Should `basic_string` have a member semantically equivalent to `strlen()`

Section: 21.1.1.6 [lib.string.capacity])

Status: active

Description:

The `basic_string` template contains two member functions which return the number of characters in the string: `size()` and `length()`. Issue 21-054 proposed

changing the semantics of `length()` to return the number of characters in the string which are positioned before the first `traits::eos()` character.

In discussions in Monterey, the LWG rejected the notion of changing the semantics of `length()`, but agreed to discuss adding a new member which is semantically equivalent to C's `strlen()`.

In lib-3973, Jerry Schwarz (jss@declarative.com) spoke against the idea:

“The string class is already large (at least IMO) and adding new functions should be done only if there is a real justification. `c_strlen` does not have any such justification. Firstly, it is inconsistent with the abstraction that string provides in which `traits::eos()` is not special. And secondly, `string::find` can be used to determine the locations of `traits::eos()`. So it provides no extra functionality.”

In lib-3997, John Max Skaller suggested that a template function be added to the library to provide this functionality. In lib-4003, Nathan Myers refined this idea into:

```
template <class charT, class Traits, class Allocator>
    typename basic_string<charT,Traits,Allocator>::size_type
    strlen(const basic_string<charT,Traits,Allocator>& s)
```

Returns: `s.find(Traits::eos())`, if that succeeds, or 0 if it fails.

Note: Result identical to `strlen(s.c_str())` for `basic_string<char>`.

Notice that this is not quite the same as `find('\0')`.

**Proposed Resolution:**

Add the following member to 21.1.1.6 [lib.string.capacity]

```
size_type c_strlen() const;
```

Returns: the minimum of `length()` and the number of char-like objects currently in the string which appear before the first `traits::eos()` character.

An alternative solution is provided above.

Requester:

LWG

Owner:

Emails:

lib-3967, lib-3968, lib-3972, lib-3973, lib-3979, lib-3983, lib-3993, lib-3995, lib-3997, lib-3999, lib-4001, lib-4003, lib-4005

Papers:

(none)

**Issue Number: 21-076**

Title:

Inconsistent pattern of arguments in `basic_string` overloads

Section:

21.1.1.3 [lib.template.string]

Status:

active

Description:

During discussions at the Monterey meeting, the LWG determined that the pattern of arguments and overloads used in member functions is often inconsistent and confusing.

Most of these inconsistencies relate to `size_type` parameters referring either to the lvalue (this) or the rvalue (a parameter passed to the member function).

**Proposed Resolution:**

Paper N0767=95-0167 (pre-Tokyo mailing) contains the proposed resolution for this issue.

Requester:

LWG

Owner:  
Emails: (none)  
Papers: (none)

**Issue Number: 21-077**

Title: basic\_string not identified as a Sequence.  
Section: 21.1.1.3 [lib.template.string]  
Status: active  
Description:

Although basic\_string has been modified to conform to the requirements for Sequences specified in Clause 23, no language in the WP specifically states that basic\_string is a Sequence.

**Proposed Resolution:**

Add the following to 21.1.1.3 [lib.basic.string]:

The template class basic\_string conforms to the requirements of a Sequence, as specified in 23.1.1 [lib.sequence.reqmts]. Additionally, because the iterators supported by basic\_string are random access iterators [lib.random.access.iterators], basic\_string conforms to the the requirements of a Reversible Container, as specified in 23.1.

Requester: LWG  
Owner:  
Emails: (none)  
Papers: (none)

**Issue Number: 21-078**

Title: Possible problem with reference counting and strings.  
Section: 21.1.1.7 [lib.string.access]  
Status: active  
Description:

In lib-4097, Uwe Steinmuller wrote:

```
string s = "abc"; //1  
char& r = s[0]; //2  
string cs = s; //3  
r = x; //4
```

Problem: If an implementation prevents (using some flag) that after processing line //2 this representation cannot be shared (copy is getting its own representation), then there is no problem.

I doubt many implementations will do so (including my own). If in line //3 cs shares the representation with s then line //4 will modify both strings. The user did nothing wrong if he looks at the standard. The reference r should be valid until a non const operation is performed on s and there is no such operation.

Solutions: We require the implementation (which is implicitly done by the current draft) to handle this case. This requires an extra flag and overhead to check for it. A restriction for the guarantee of r would be also a solution but could get quite complicated.

In lib-4102, Steven Kearns wrote:

One solution is to have operator[](int index) return a helper class:

```
class StringHelper {  
    int index;
```

```
String& s;  
StringHelper(String& s0, int index0) : index(index0), s(s0) {}  
operator=(char c) { s.SetAt(index, c); }  
};
```

Unfortunately, this makes the most common idiom:

```
String s;  
s[0] = 'a';
```

much more inefficient than before. So the only practical solution is to come up with a suitable restriction on the lifetime of the reference returned. How about the obvious one of saying that the reference returned is only valid until the next non-const operation on the string, or until the string is copied or assigned from.

Proposed Resolution:

Requester: Uwe Steinmuller (Uwe.Steinmueller@zfe.siemens.de)  
Owner:  
Emails: lib-4097, lib-4101, lib-4102, lib-4105, lib-4107  
Papers: (none)

## Closed Issues

Issues which have been recently closed are included in their entirety. Issues which have appeared in a previous version of the issues list as “closed” have the bulk of their content deleted for brevity. The document number of the paper in which they last appeared is included for reference.

### Issue Number: 21-001

Title: Should basic\_string have a getline() function?  
Last Doc.: N0721=95-0121

### Issue Number: 21-003

Title: Character-oriented assign function has incorrect signature  
Last Doc.: N0721=95-0121

### Issue Number: 21-004

Title: Character-oriented replace function has incorrect signature  
Last Doc.: N0759=95-0159

### Issue Number: 21-005

Title: How come the string class does not have a prepend() function?  
Last Doc.: N0759=95-0159

### Issue Number: 21-006

Title: Should the Allocator be the last template argument to basic\_string?  
Last Doc.: N0721=95-0121

### Issue Number: 21-007

Title: Should the string\_char\_traits speed-up functions be specified as inline?  
Last Doc.: N0759=95-0159

### Issue Number: 21-008

Title: Should an ostream inserter and extractor be specified for basic\_string?  
Last Doc.: N0759=95-0159

**Issue Number: 21-009**

Title: Why are character parameters passed as “const charT”?  
Last Doc.: N0721=95-0121

**Issue Number: 21-010**

Title: Should member parameters passed as “const\_pointer”?  
Last Doc.: N0721=95-0121

**Issue Number: 21-011**

Title: Why are character parameters to the string traits functions passed by reference?  
Last Doc.: N0721=95-0121

**Issue Number: 21-012**

Title: Why are character parameters to the string functions passed by value?  
Section: 21.1.1.2 [lib.string.char.traits]  
Status: closed  
Description:

In the string functions, character parameters are specified as being passed by “charT”. In the past, the LWG had decided that char-like types should be considered cheap enough to pass by value.

However during discussions at the Austin meeting, the LWG formed the consensus that characters should be passed by reference. The rationale was: for most character types, on most architectures, it was as efficient for characters to be passed by references instead of by value. The importance of reference parameters arrived in atypical character types which might be considerably larger than ASCII characters

Resolution:

All character parameters to all string functions will be passed by const reference.

Requester: Rick Wilhelm: rkw@chi.andersen.com

Owner:

Emails: (none)

Papers: (none)

**Issue Number: 21-015**

Title: The copy() member should be const.  
Last Doc.: N0759=95-0159

**Issue Number: 21-016**

Title: The error conditions are not well-specified for the find() and rfind() functions.  
Last Doc.: N0759=95-0159

**Issue Number: 21-019**

Title: The Allocator template parameter is not reflected in a member typedef.  
Last Doc.: N0759=95-0159

**Issue Number: 21-020**

Title: Header for Table 42 is incorrect.  
Last Doc.: N0759=95-0159

**Issue Number: 21-021**

Title: compare() has unexpected results

Last Doc.: N0759=95-0159

**Issue Number: 21-022**

Title: s.append('c') appends 99 nulls.

Last Doc.: N0759=95-0159

**Issue Number: 21-023**

Title: Non-conforming default Allocator arguments

Last Doc.: N0759=95-0159

**Issue Number: 21-032**

Title: Description for operator+() is incorrect

Last Doc.: N0759=95-0159

**Issue Number: 21-033**

Title: Requirements for const charT\* arguments not specified

Last Doc.: N0759=95-0159

**Issue Number: 21-035**

Title: Character replacement does not change length.

Last Doc.: N0759=95-0159

**Issue Number: 21-036**

Title: Character case disregarded during common operations.

Last Doc.: N0759=95-0159

**Issue Number: 21-038**

Title: Operator < clashes cause ambiguity

Last Doc.: N0759=95-0159

**Issue Number: 21-039**

Title: Iterator parameters can get confused with size\_type parameters.

Last Doc.: N0759=95-0159

**Issue Number: 21-040**

Title: Repetition parameter non-intuitive

Last Doc.: N0759=95-0159

**Issue Number: 21-041**

Title: Assignment operator defined in terms of itself

Last Doc.: N0759=95-0159

**Issue Number: 21-042**

Title: Character assignment defined in terms of non-existent constructor

Last Doc.: N0759=95-0159

**Issue Number: 21-043**

Title: Character append operator defined in terms of non-existent constructor  
Last Doc.: N0759=95-0159

**Issue Number: 21-044**

Title: Character modifiers defined in terms of non-existent constructor  
Last Doc.: N0759=95-0159

**Issue Number: 21-045**

Title: Iterator typenamees overspecified  
Last Doc.: N0759=95-0159

**Issue Number: 21-046**

Title: basic\_string type syntactically incorrect in some descriptions  
Last Doc.: N0759=95-0159

**Issue Number: 21-047**

Title: Error in description of replace() member  
Last Doc.: N0759=95-0159

**Issue Number: 21-048**

Title: Inconsistency in const-ness of compare() declarations  
Last Doc.: N0759=95-0159

**Issue Number: 21-049**

Title: Inconsistency constructor effects and semantics of data()  
Last Doc.: N0759=95-0159

**Issue Number: 21-050**

Title: Incorrect semantics for operator+()  
Last Doc.: N0759=95-0159

**Issue Number: 21-051**

Title: Incorrect return type for insert() member  
Last Doc.: N0759=95-0159

**Issue Number: 21-052**

Title: Unconstrained position arguments for find members.  
Last Doc.: N0759=95-0159

**Issue Number: 21-053**

Title: Semantics of size() prevents null characters in string  
Last Doc.: N0759=95-0159

**Issue Number: 21-054**

Title: Change the semantics of length()  
Last Doc.: N0759=95-0159

**Issue Number: 21-055**

Title: append(), assign() have incorrect requirements  
Last Doc.: N0759=95-0159

**Issue Number: 21-056**

Title: Requirements for insert() are too weak.  
Last Doc.: N0759=95-0159

**Issue Number: 21-057**

Title: replace has incorrect requirements  
Last Doc.: N0759=95-0159

**Issue Number: 21-058**

Title: Description of data() is over-constraining.  
Last Doc.: N0759=95-0159

**Issue Number: 21-064**

Title: Miscellaneous errors in resize(size\_type n)  
Last Doc.: N0759=95-0159

**Issue Number: 21-065**

Title: Incorrect return value for insert()  
Last Doc.: N0759=95-0159

**Issue Number: 21-066**

Title: Description of remove() is over-specific  
Last Doc.: N0759=95-0159

**Issue Number: 21-069**

Title: Swap complexity underspecified.  
Last Doc.: N0759=95-0159

**Issue Number: 21-070**

Title: operator>= described incorrectly  
Last Doc.: N0759=95-0159

**Issue Number: 21-071**

Title: Does getline() have the correct semantics?  
Last Doc.: N0759=95-0159

**Issue Number: 21-072**

Title: Incorrect use of size\_type in third table in section  
Last Doc.: N0759=95-0159

**Issue Number: 21-073**

Title: Add overloads to functions that take default character object.  
Last Doc.: N0759=95-0159

**Issue Number: 21-075**

Title: Incomplete specification for assignment operator  
Section: 21.1.1.4 [lib.string.cons]  
Status: closed  
Description:

The current description of the basic\_string assignment operator does not handle the case of a string being assigned to itself.

Resolution:

**Clause 21 (Strings Library) Issues List: Rev. 8 - 95-178=N0778**

In the `basic_string` assignment operator's Effects description, add the following after the table:

If `*this` and `str` are the same object, no effect.

Requester: LWG  
Owner:  
Emails: (none)  
Papers: (none)