Doc. no. J16/99-0031
WG21 N1207
Date: 25 August 1999
Project: Programming Language C++

C++ Standard Library Closed Issues List (Revision 9)

Reference ISO/IEC IS 14882:1998(E)

Also see:

- Table of Contents including both active and closed issues.
- Index by Section including both active and closed issues.
- Index by Status including both active and closed issues.
- Active Issues List

This document contains only library issues which have been closed by the Library Working Group. That is, issues which have a status of Dup, NAD, DR, TC, or RR. See the "C++ Standard Library Active Issues List" for active issues and more information. The introductory material in that document also applies to this document.

Closed Issues

---

1. C library linkage editing oversight

Section: 17.4.2.2 lib.using.linkage **Status**: DR **Submitter:** Beman Dawes **Date:** 16 Nov 97

The change specified in the proposed resolution below did not make it into the Standard. This change was accepted in principle at the London meeting, and the exact wording below was accepted at the Morristown meeting.

Proposed Resolution:

Change lib.using.linkage paragraph 2 from:

> It is unspecified whether a name from the Standard C library declared with external linkage has either extern "C" or extern "C++" linkage.

to:

> Whether a name from the Standard C library declared with external linkage has extern "C" or extern "C++" linkage is implementation defined. It is recommended that an implementation use extern "C++" linkage for this purpose.

---

2. Auto_ptr conversions effects incorrect

Section: 20.4.5.3 lib.auto.ptr.conv **Status**: DR **Submitter:** Nathan Myers **Date:** 4 Dec 97

Paragraph 1 in "Effects", says "Calls p->release()" where it clearly must be "Calls p.release()". (As it is, it seems to require using auto_ptr<>::operator-> to refer to X::release, assuming that exists.)

Proposed Resolution:

Change lib.auto.ptr.conv paragraph 1 Effects from "Calls p->release()" to "Calls p.release()".

---

4. Basic_string size_type and difference_type should be implementation defined

Section: 21.3 lib.basic.string **Status**: DR **Submitter:** Beman Dawes **Date:** 16 Nov 97

In Morristown we changed the size_type and difference_type typedefs for all the other containers to implementation defined with a reference to lib.container.requirements. This should probably also have been done for strings.

Proposed Resolution:

Change lib.basic.string from:

```
typedef typename Allocator::size_type       size_type;
typedef typename Allocator::difference_type difference_type;
```

to:

```
typedef implementation defined size_type; // See lib.container.requirements
typedef implementation defined difference_type; // See lib.container.requirements
```

---

6. File position not an offset unimplementable

Section: 27.4.3 lib.fpos **Status**: NAD **Submitter:** Matt Austern **Date:** 15 Dec 97

Table 88, in I/O, is too strict; it's unimplementable on systems where a file position isn't just an offset. It also never says just what fpos<> is really supposed to be. [Here's my summary. "I think I now know what the class really is, at this point: it's a magic cookie that encapsulates an mbstate_t and a file position (possibly represented as an fpos_t), it has syntactic support for pointer-like arithmetic, and implementors are required to have real, not just syntactic, support for arithmetic." This isn't standardese, of course.]

Rationale:

Not a defect. The LWG believes that the Standard is already clear, and that the above summary is what the Standard in effect says.

---

10. Codecvt<>::do unclear

Section: 22.2.1.5.2 lib.locale.codecvt.virtuals **Status**: Dup **Submitter:** Matt Austern **Date:** 14 Jan 98

Section 22.2.1.5.2 says that codecvt<>::do_in and do_out should return the value noconv if "no conversion was needed". However, I don't see anything anywhere that defines what it means for a conversion to be needed or not needed. I can think of several circumstances where one might plausibly think that a conversion is not "needed", but I don't know which one is intended here.

Rationale:

Duplicate. See issue19.

---

12. Way objects hold allocators unclear

Section: 20.1.5 lib.allocator.requirements **Status**: NAD **Submitter:** Angelika Langer **Date:** 23 Feb 98

I couldn't find a statement in the standard saying whether the allocator object held by a container is held as a copy of the constructor argument or whether a pointer of reference is maintained internal. There is an according statement for compare objects and how they are maintained by the associative containers, but I couldn't find anything regarding allocators.

Did I overlook it? Is it an open issue or known defect? Or is it deliberately left unspecified?

Rationale:

Not a defect. The LWG believes that the Standard is already clear. See 23.1 paragraph 8 [lib.container.requirements].

---

13. Eos refuses to die

Section: 27.6.1.2.3 lib.istream::extractors **Status**: DR **Submitter:** William M. Miller **Date:** 3 Mar 98

In 27.6.1.2.3, there is a reference to "eos", which is the only one in the whole draft (at least using Acrobat search), so it's undefined.

Proposed Resolution:

In 27.6.1.2.3 lib.istream::extractors, replace "eos" with "charT()"

---

14. Locale::combine should be const

Section: 22.1.1.3 lib.locale.members **Status**: DR **Submitter:** Nathan Myers **Date:** 6 Aug 98

locale::combine is the only member function of locale (other than constructors and destructor) that is not const. There is no reason for it not to be const, and good reasons why it should have been const. Furthermore, leaving it non-const conflicts with 22.1.1 paragraph 6: "An instance of a locale is immutable."

History: this member function originally was a constructor. it happened that the interface it specified had no corresponding language syntax, so it was changed to a member function. As constructors are never const, there was no "const" in the interface which was transformed into member "combine". It should have been added at that time, but the omission was not noticed.

Proposed Resolution:

In 22.1.1 [lib.locale] and also in 22.1.1.3 [lib.locale.members], add "const" to the declaration of member combine:

```
template <class Facet> locale combine(const locale& other) const;
```

---

15. Locale::name requirement inconsistent

Section: 22.1.1.3 lib.locale.members **Status**: DR **Submitter:** Nathan Myers **Date:** 6 Aug 98

locale::name() is described as returning a string that can be passed to a locale constructor, but there is no matching constructor.

Proposed Resolution:

In 22.1.1.3 [lib.locale.members], paragraph 5, replace "`locale(name())`" with "`locale(name().c_str())`".

---

16. Bad ctype_byname<char> decl

Section: 22.2.1.4 lib.locale.ctype.byname.special **Status**: DR **Submitter:** Nathan Myers **Date:** 6 Aug 98

The new virtual members ctype_byname<char>::do_widen and do_narrow did not get edited in properly. Instead, the member do_widen appears four times, with wrong argument lists.

Proposed Resolution:

The correct declarations for the overloaded members `do_narrow` and `do_widen` should be copied from 22.2.1.3, [lib.facet.ctype.special].

---

18. Get(...bool&) omitted

Section: 22.2.2.1.1 lib.facet.num.get.members **Status**: DR **Submitter:** Nathan Myers **Date:** 6 Aug 98

In the list of num_get<> non-virtual members on page 22-23, the member that parses bool values was omitted from the list of definitions of non-virtual members, though it is listed in the class definition and the corresponding virtual is listed everywhere appropriate.

Proposed Resolution:

Add at the beginning of 22.2.2.1.1 [lib.facet.num.get.members] another get member for bool&, copied from the entry in 22.2.2.1 [lib.locale.num.get].

---

20. Thousands_sep returns wrong type

Section: 22.2.3.1.2 lib.facet.numpunct.virtuals **Status**: DR **Submitter:** Nathan Myers **Date:** 6 Aug 98

The synopsis for numpunct<>::do_thousands_sep, and the definition of numpunct<>::thousands_sep which calls it, specify that it returns a value of type char_type. Here it is erroneously described as returning a "string_type".

Proposed Resolution:

In 22.2.3.1.2 [lib.facet.numpunct.virtuals], above paragraph 2, change "string_type" to "char_type".

---

22. Member open vs. flags

Section: 27.8.1.7 lib.ifstream.members **Status**: DR **Submitter:** Nathan Myers **Date:** 6 Aug 98

The description of basic_istream<>::open leaves unanswered questions about how it responds to or changes flags in the error status for the stream. A strict reading indicates that it ignores the bits and does not change them, which confuses users who do not expect eofbit and failbit to remain set after a successful open. There are three reasonable resolutions: 1) status quo 2) fail if fail(), ignore eofbit 3) clear failbit and eofbit on call to open().

Proposed Resolution:

In 27.8.1.7 [lib.ifstream.members] paragraph 3, _and_ in 27.8.1.10 [lib.ofstream.members] paragraph 3, under open() effects, add a footnote:

> A successful open does not change the error state.

---

23. Num_get overflow result

Section: 22.2.2.1.2 lib.facet.num.get.virtuals **Status**: DR **Submitter:** Nathan Myers **Date:** 6 Aug 98

The current description of numeric input does not account for the possibility of overflow. This is an implicit result of changing the description to rely on the definition of scanf() (which fails to report overflow), and conflicts with the documented behavior of traditional and current implementations.

Users expect, when reading a character sequence that results in a value unrepresentable in the specified type, to have an error reported. The standard as written does not permit this.

Proposed Resolution:

In 22.2.2.1.2 [lib.facet.num.get.virtuals], paragraph 11, second bullet item, change

> The sequence of chars accumulated in stage 2 would have caused scanf to report an input failure.

to

> The sequence of chars accumulated in stage 2 would have caused scanf to report an input failure, or the value of the sequence cannot be represented in the type of _val_.

---

24. "do_convert" doesn't exist

Section: 22.2.1.5.2 lib.locale.codecvt.virtuals **Status**: DR **Submitter:** Nathan Myers **Date:** 6 Aug 98

The description of codecvt<>::do_out and do_in mentions a symbol "do_convert" which is not defined in the standard. This is a leftover from an edit, and should be "do_in and do_out".

Proposed Resolution:

In 22.2.1.5 [lib.locale.codecvt], paragraph 3, change "do_convert" to "do_in or do_out". Also, In 22.2.1.5.2 [lib.locale.codecvt.virtuals], change "do_convert()" to "do_in or do_out".

---

25. String operator<< uses width() value wrong

Section: 21.3.7.9 lib.string.io **Status**: DR **Submitter:** Nathan Myers **Date:** 6 Aug 98

In the description of operator<< applied to strings, the standard says that uses the smaller of os.width() and str.size(), to pad "as described in stage 3" elsewhere; but this is inconsistent, as this allows no possibility of space for padding.

Proposed Resolution:

Change 21.3.7.9 lib.string.io paragraph 4 from:

"... where `n` is the smaller of `os.width()` and `str.size()`; ..."

to:

"... where `n` is the larger of `os.width()` and `str.size()`; ..."

---

27. String::erase(range) yields wrong iterator

Section: 21.3.5.5 lib.string::erase **Status**: DR **Submitter:** Nathan Myers **Date:** 6 Aug 98

The string::erase(iterator first, iterator last) is specified to return an element one place beyond the next element after the last one erased. E.g. for the string "abcde", erasing the range ['b'..'d') would yield an iterator for element 'e', while 'd' has not been erased.

Proposed Resolution:

In 21.3.5.5 [lib.string::erase], paragraph 10, change:

> Returns: an iterator which points to the element immediately following _last_ prior to the element being erased.

to read

> Returns: an iterator which points to the element pointed to by _last_ prior to the other elements being erased.

---

28. Ctype<char>is ambiguous

Section: 22.2.1.3.2 [lib.facet.ctype.char.members] **Status**: DR **Submitter:** Nathan Myers **Date:** 6 Aug 98

The description of the vector form of ctype<char>::is can be interpreted to mean something very different from what was intended. Paragraph 4 says

> Effects: The second form, for all *p in the range [low, high), assigns vec[p-low] to table()[(unsigned char)*p].

This is intended to copy the value indexed from table()[] into the place identified in vec[].

Proposed Resolution:

Change 22.2.1.3.2 [lib.facet.ctype.char.members], paragraph 4, to read

> Effects: The second form, for all *p in the range [low, high), assigns into vec[p-low] the value table() [(unsigned char)*p].

---

29. Ios_base::init doesn't exist

Section: 27.3.1 lib.narrow.stream.objects **Status**: DR **Submitter:** Nathan Myers **Date:** 6 Aug 98

Sections 27.3.1 and 27.3.2 [lib.wide.stream.objects] mention a function ios_base::init, which is not defined. Probably it means basic_ios<>::init, defined in 27.4.4.1 [lib.basic.ios.cons], paragraph 3.

Proposed Resolution:

In 27.3.1 [lib.narrow.stream.objects] paragraph 2, change

> ios_base::init

to

> basic_ios<char>::init

Also, make a similar change in 27.3.2 [lib.wide.stream.objects] except it should read

> basic_ios<wchar_t>::init

---

30. Wrong header for LC_*

Section: 22.1.1.1.1 [lib.locale.category] **Status**: DR **Submitter:** Nathan Myers **Date:** 6 Aug 98

Paragraph 2 implies that the C macros LC_CTYPE etc. are defined in <cctype>, where they are in fact defined elsewhere to appear in <clocale>.

Proposed Resolution:

In 22.1.1.1.1 [lib.locale.category], paragraph 2, change "<cctype>" to read "<clocale>".

---

33. Codecvt<> mentions from_type

Section: 22.2.1.5.2 [lib.locale.codecvt.virtuals] **Status**: DR **Submitter:** Nathan Myers **Date:** 6 Aug 98

In the table defining the results from do_out and do_in, the specification for the result _error_ says

> encountered a from_type character it could not convert

but from_type is not defined. This clearly is intended to be an externT for do_in, or an internT for do_out.

Proposed Resolution:

In 22.2.1.5.2 [lib.locale.codecvt.virtuals], paragraph 4, replace the definition in the table for the case of _error_ with

> encountered a character in `[from,from_end)` that it could not convert.

---

34. True/falsename() not in ctype<>

Section: 22.2.2.2.2 [lib.facet.num.get.virtuals] **Status**: DR **Submitter:** Nathan Myers **Date:** 6 Aug 98

In paragraph 19, Effects:, members truename() and falsename are used from facet ctype<charT>, but it has no such members. Note that this is also a problem in 22.2.2.1.2, addressed in (4).

Proposed Resolution:

In 22.2.2.2.2 [lib.facet.num.get.virtuals], paragraph 19, in the Effects: clause for member put(...., bool), replace the initialization of the string_type value s as follows:

```
const numpunct& np = use_facet<numpunct<charT> >(loc);
string_type s = val ? np.truename() : np.falsename();
```

---

35. No manipulator unitbuf in synopsis

Section: 27.4 [lib.iostreams.base] **Status**: DR **Submitter:** Nathan Myers **Date:** 6 Aug 98

In 27.4.5.1, [lib.fmtflags.manip], we have a definition for a manipulator named "unitbuf". Unlike other manipulators, it's not listed in sysopsis. Similarly for "nounitbuf".

Proposed Resolution:

Add to the synopsis for <ios> in 27.4 [lib.iostreams.base], after the entry for "nouppercase", the prototypes:

```
ios_base& unitbuf(ios_base& str);
ios_base& nounitbuf(ios_base& str);
```

---

36. Iword & pword storage lifetime omitted

Section: 27.4.2.5 [lib.ios.base.storage] **Status**: DR **Submitter:** Nathan Myers **Date:** 6 Aug 98

In the definitions for ios_base::iword and pword, the lifetime of the storage is specified badly, so that an implementation which only keeps the last value stored appears to conform. In particular, it says:

The reference returned may become invalid after another call to the object's iword member with a different index ...

This is not idle speculation; at least one implementation was done this way.

Proposed Resolution:

Add in 27.4.2.5 [lib.ios.base.storage], in both paragraph 2 and also in paragraph 4, replace the sentence:

> The reference returned may become invalid after another call to the object's iword [pword] member with a different index, after a call to its copyfmt member, or when the object is destroyed.

with:

> The reference returned is invalid after any other operations on the object. However, the value of the storage referred to is retained, so that until the next call to copyfmt, calling iword [pword] with the same index yields another reference to the same value.

substituting "iword" or "pword" as appropriate.

---

37. Leftover "global" reference

Section: 22.1.1 [lib.locale] **Status**: DR **Submitter:** Nathan Myers **Date:** 6 Aug 98

In the overview of locale semantics, paragraph 4, is the sentence

> If Facet is not present in a locale (or, failing that, in the global locale), it throws the standard exception bad_cast.

This is not supported by the definition of use_facet<>, and represents semantics from an old draft.

Proposed Resolution:

In 22.1.1 [lib.locale], paragraph 4, delete the parenthesized expression

> (or, failing that, in the global locale)

---

38. Facet definition incomplete

Section: 22.1.2 [lib.locale.global.templates] **Status**: DR **Submitter:** Nathan Myers **Date:** 6 Aug 98

It has been noticed that the definition of "facet" is incomplete. In particular, a class derived from another facet, but which does not define a member _id_, cannot safely serve as the argument _F_ to use_facet<F>(loc), because there is no guarantee that a reference to the facet instance stored in _loc_ is safely convertible to _F_.

Proposed Resolution:

In the definition of std::use_facet<>(), replace the text in paragraph 1 which reads:

> Get a reference to a facet of a locale.

with:

> Requires: `Facet` is a facet class whose definition contains (not inherits) the public static member `id` as

defined in (22.1.1.1.2, [lib.locale.facet]).

---

39. Sbufiter ++ definition garbled

Section: 24.5.3.4 [lib.istreambuf.iterator::op++] **Status**: DR **Submitter:** Nathan Myers **Date:** 6 Aug 98

Following the definition of istreambuf_iterator<>::operator++(int) in paragraph 3, the standard contains three lines of garbage text left over from a previous edit.

```
istreambuf_iterator<charT,traits> tmp = *this;
sbuf_->sbumpc();
return(tmp);
```

Proposed Resolution:

In 24.5.3.4 [lib.istreambuf.iterator::op++], delete the three lines of code at the end of paragraph 3.

---

40. Meaningless normative paragraph in examples

Section: 22.2.8 [lib.facets.examples] **Status**: DR **Submitter:** Nathan Myers **Date:** 6 Aug 98

Paragraph 3 of the locale examples is a description of part of an implementation technique that has lost its referent, and doesn't mean anything.

Proposed Resolution:

Delete 22.2.8 [lib.facets.examples] paragraph 3 which begins "This initialization/identification system depends...", or (at the editor's option) replace it with a place-holder to keep the paragraph numbering the same.

---

43. Locale table correction

Section: 22.2.1.5.2 lib.locale.codecvt.virtuals **Status**: Dup **Submitter:** Brendan Kehoe **Date:** 1 Jun 98

Rationale:

Duplicate. See issue 33.

---

45. Stringstreams read/write pointers initial position unclear

Section: 27.7.3 lib.ostringstream **Status**: NAD **Submitter:** **Date:** 27 May 98

In a a comp.lang.c++.moderated :

"We are not sure how to interpret the CD2 (see [lib.iostream.forward], [lib.ostringstream.cons], [lib.stringbuf.cons]) with respect to the question as to what the correct initial positions of the write and read pointers of a stringstream should be."

"Is it the same to output two strings or to initialize the stringstream with the first and to output the second ?"

Rationale:

The LWG believes the Standard is correct as written. The behavior of stringstreams is consistent with fstreams, and there is a constructor which can be used to obtain the desired effect. This behavior is known to be different from strstreams.

---

46. Minor Annex D errors

Section: D.7 depr.strstreambuf, depr.strstream **Status**: DR **Submitter:** Brendan Kehoe **Date:** 1 Jun 98

See lib-6522, edit- 814.

Proposed Resolution:

Change D.7.1 depr.strstreambuf (since streambuf is a typedef of basic_streambuf<char>) from:

```
virtual streambuf<char>* setbuf(char* s, streamsize n);
```

to:

```
virtual streambuf* setbuf(char* s, streamsize n);
```

In D.7.4 depr.strstream insert the semicolon now missing after int_type:

```
namespace std {
  class strstream
    : public basic_iostream<char> {
  public:
    // Types
    typedef char                                    char_type;
    typedef typename char_traits<char>::int_type int_type
    typedef typename char_traits<char>::pos_type pos_type;
```

---

47. Imbue() and getloc() Returns clauses swapped

Section: 27.4.2.3 lib.ios.base.locales **Status**: DR **Submitter:** Matt Austern **Date:** 21 Jun 98

Section 27.4.2.3 specifies how imbue() and getloc() work. That section has two RETURNS clauses, and they make no sense as stated. They make perfect sense, though, if you swap them. Am I correct in thinking that paragraphs 2 and 4 just got mixed up by accident?

Proposed Resolution:

In 27.4.2.3 lib.ios.base.locales swap paragraphs 2 and 4.

---

51. Requirement to not invalidate iterators missing

Section: 23.1 lib.container.requirements **Status**: DR **Submitter:** David Vandevoorde **Date:** 23 Jun 98

The std::sort algorithm can in general only sort a given sequence by moving around values. The list<>::sort() member on the other hand could move around values or just update internal pointers. Either method can leave iterators into the list<> dereferencable, but they would point to different things.

Does the FDIS mandate anywhere which method should be used for list<>::sort()?

A committee member comments:

I think you've found an omission in the standard.

The library working group discussed this point, and there was supposed to be a general requirement saying that list, set, map, multiset, and multimap may not invalidate iterators, or change the values that iterators point to, except when an operation does it explicitly. So, for example, insert() doesn't invalidate any iterators and erase() and remove() only invalidate iterators pointing to the elements that are being erased.

I looked for that general requirement in the FDIS, and, while I found a limited form of it for the sorted associative containers, I didn't find it for list. It looks like it just got omitted.

The intention, though, is that list<>::sort does not invalidate any iterators and does not change the values that any iterator points to. There would be no reason to have the member function otherwise.

The issues list maintainer comments:

This was US issue CD2-23-011; it was accepted in London . The wording in the proposed resolution below is somewhat updated from CD2-23-011, particularly the addition of the phrase "or change the values of"

Proposed Resolution:

Add a new paragraph at the end of 23.1:

> Unless otherwise specified (either explicitly or by defining a function in terms of other functions), invoking a container member function or passing a container as an argument to a library function shall not invalidate iterators to, or change the values of, objects within that container.

---

52. Small I/O problems

Section: 27.4.3.2 lib.fpos.operations **Status**: DR **Submitter:** Matt Austern **Date:** 23 Jun 98

First, 27.4.4.1 lib.basic.ios.cons table 89. This is pretty obvious: it should be titled "basic_ios<>() effects", not "ios_base () effects".

[The second item is a duplicate; see issue 6 for resolution.]

Second, 27.4.3.2 lib.fpos.operations table 88 . There are a couple different things wrong with it, some of which I've already discussed with Jerry, but the most obvious mechanical sort of error is that it uses expressions like P(i) and p(i), without ever defining what sort of thing "i" is.

(The other problem is that it requires support for streampos arithmetic. This is impossible on some systems, i.e. ones where file position is a complicated structure rather than just a number. Jerry tells me that the intention was to require syntactic support for streampos arithmetic, but that it wasn't actually supposed to do anything meaningful except on

platforms, like Unix, where genuine arithmetic is possible.)

Proposed Resolution:

Change 27.4.4.1 lib.basic.ios.cons table 89 title from "ios_base() effects" to "basic_ios<>() effects".

---

56. Showmanyc's return type

Section: 27.5.2 lib.streambuf **Status**: DR **Submitter:** Matt Austern **Date:**29 Jun 98

The class summary for basic_streambuf<>, in 27.5.2, says that showmanyc has return type int. However, 27.5.2.4.3 says that its return type is streamsize.

Proposed Resolution:

Change `showmanyc`'s return type in the 27.5.2 lib.streambuf class summary to `streamsize`.

---

57. Mistake in char_traits

Section: 21.1.3.2 lib.char.traits.specializations.wchar.t **Status**: DR **Submitter:** Matt Austern **Date:**1 Jul 98

21.1.3.2, paragraph 3, says "The types streampos and wstreampos may be different if the implementation supports no shift encoding in narrow-oriented iostreams but supports one or more shift encodings in wide-oriented streams".

That's wrong: the two are the same type. The <iosfwd> summary in 27.2 says that streampos and wstreampos are, respectively, synonyms for fpos<char_traits<char>::state_type> and fpos<char_traits<wchar_t>::state_type>, and, flipping back to clause 21, we see in 21.1.3.1 and 21.1.3.2 that char_traits<char>::state_type and char_traits<wchar_t>::state_type must both be mbstate_t.

Proposed Resolution:

Remove the sentence in 21.1.3.2 lib.char.traits.specializations.wchar.t paragraph 3 which begins "The types streampos and wstreampos may be different..." .

---

59. Ambiguity in specification of gbump

Section: 27.5.2.3.1 lib.streambuf.get.area **Status**: DR **Submitter:** Matt Austern **Date:**28 Jul 98

27.5.2.3.1 says that basic_streambuf::gbump() "Advances the next pointer for the input sequence by n."

The straightforward interpretation is that it is just gptr() += n. An alternative interpretation, though, is that it behaves as if it calls sbumpc n times. (The issue, of course, is whether it might ever call underflow.) There is a similar ambiguity in the case of pbump.

AT&T implementation used the former interpretation.

Proposed Resolution:

Change 27.5.2.3.1 lib.streambuf.get.area paragraph 4 gbump effects from:

Effects: Advances the next pointer for the input sequence by n.

to:

Effects: Adds n to the next pointer for the input sequence.

Make the same change to 27.5.2.3.2 lib.streambuf.put.area paragraph 4 pbump effects.

---

62. Sync's return value

Section: 27.6.1.3 lib.istream.unformatted **Status**: DR **Submitter:** Matt Austern **Date:**6 Aug 98

The Effects clause for sync() (27.6.1.3, paragraph 36) says that it "calls rdbuf()->pubsync() and, if that function returns -1 ... returns traits::eof()."

That looks suspicious, because traits::eof() is of type traits::int_type while the return type of sync() is int.

Proposed Resolution:

In 27.6.1.3 lib.istream.unformatted, paragraph 36, change "returns `traits::eof()`" to "returns `-1`".

---

64. Exception handling in `basic_istream::operator>>(basic_streambuf*)`

Section: 27.6.1.2.3 lib.istream::extractors **Status**: DR **Submitter:** Matt Austern **Date:**11 Aug 98

27.6.1.2.3, paragraph 13, is ambiguous. It can be interpreted two different ways, depending on whether the second sentence is read as an elaboration of the first.

Proposed Resolution:

Replace 27.6.1.2.3 lib.istream::extractors, paragraph 13, which begins "If the function inserts no characters ..." with:

> If the function inserts no characters, it calls `setstate(failbit)`, which may throw `ios_base::failure` (27.4.4.3). If it inserted no characters because it caught an exception thrown while extracting characters from `sb` and `failbit` is on in `exceptions()` (27.4.4.3), then the caught exception is rethrown.

---

66. Strstreambuf::setbuf

Section: D.7.1.3 depr.strstreambuf.virtuals **Status**: DR **Submitter:** Matt Austern **Date:**18 Aug 98

D.7.1.3, paragraph 19, says that strstreambuf::setbuf "Performs an operation that is defined separately for each class derived from strstreambuf". This is obviously an incorrect cut-and-paste from basic_streambuf. There are no classes derived from strstreambuf.

Proposed Resolution:

D.7.1.3 depr.strstreambuf.virtuals, paragraph 19, replace the setbuf effects clause which currently says "Performs an operation that is defined separately for each class derived from strstreambuf" with:

> **Effects**: implementation defined, except that `setbuf(0,0)` has no effect.

---

67. Setw useless for strings

Section: 21.3.7.9 lib.string.io **Status**: Dup **Submitter:** Steve Clamage **Date:** 9 Jul 98

In a comp.std.c++ posting : What should be output by :

```
string text("Hello");
cout << '[' << setw(10) << right << text << ']';
```

Shouldn't it be:

```
[     Hello]
```

Another person replied: Actually, according to the FDIS, the width of the field should be the minimum of width and the length of the string, so the output shouldn't have any padding. I think that this is a typo, however, and that what is wanted is the maximum of the two. (As written, setw is useless for strings. If that had been the intent, one wouldn't expect them to have mentioned using its value.)

It's worth pointing out that this is a recent correction anyway; IIRC, earlier versions of the draft forgot to mention formatting parameters what soever.

Rationale:

Duplicate. See issue 25.

---

68. Extractors for char* should store null at end

Section: 27.6.1.2.3 lib.istream::extractors **Status**: DR **Submitter:** Angelika Langer **Date:** 14 Jul 98

Extractors for char* (27.6.1.2.3) do not store a null character after the extracted character sequence whereas the unformatted functions like get() do. Why is this?

Proposed Resolution:

27.6.1.2.3 lib.istream::extractors, paragraph 7, change the last list item from:

> A null byte (charT()) in the next position, which may be the first position if no characters were extracted.

to become a new paragraph which reads:

> Operator>> then stores a null byte (charT()) in the next position, which may be the first position if no characters were extracted.

---

70. Uncaught_exception() missing throw() specification

Section: 18.6 lib.support.exception, 18.6.4 lib.uncaught **Status**: DR **Submitter:** Steve Clamage **Date:**

In article 3E04@pratique.fr, writes:

uncaught_exception() doesn't have a throw specification.

It is intentionnal ? Does it means that one should be prepared to handle exceptions thrown from uncaught_exception() ?

uncaught_exception() is called in exception handling contexts where exception safety is very important. >

Proposed Resolution:

In 18.6 lib.support.exception and 18.6.4 lib.uncaught add "throw()" to uncaught_exception().

---

71. Do_get_monthname synopsis missing argument

Section: 22.2.5.1 [lib.locale.time.get] **Status**: DR **Submitter:** Nathan Myers **Date:** 13 Aug 98

The locale facet member `time_get<>::do_get_monthname` is described in 22.2.5.1.2 [lib.locale.time.get.virtuals] with five arguments, consistent with do_get_weekday and with its specified use by member get_monthname. However, in the synopsis, it is specified instead with four arguments. The missing argument is the "end" iterator value.

Proposed Resolution:

In 22.2.5.1 [lib.locale.time.get], add an "end" argument to the declaration of member do_monthname as follows:

```
virtual iter_type do_get_monthname(iter_type s, iter_type end, ios_base&,
                                   ios_base::iostate& err, tm* t) const;
```

---

72. Do_convert phantom member function

Section: 22.2.1.5 lib.locale.codecvt **Status**: Dup **Submitter:** Nathan Myers **Date:** 24 Aug 98

In 22.2.1.5 par 3 lib.locale.codecvt, and in 22.2.1.5.2 par 8 lib.locale.codecvt.virtuals, a nonexistent member function "do_convert" is mentioned. This member was replaced with "do_in" and "do_out", the proper referents in the contexts above.

Proposed Resolution:

Duplicate: see issue 24.

---

73. `is_open` should be const

Section: 27.8.1 lib.file.streams **Status**: NAD **Submitter:** Matt Austern **Date:** 27 Aug 98

Classes `basic_ifstream`, `basic_ofstream`, and `basic_fstream` all have a member function `is_open`. It should be a `const` member function, since it does nothing but call one of `basic_filebuf`'s const member functions.

Rationale:

Not a defect. This is a deliberate feature; const streams would be meaningless.

---

77. Valarray operator[] const returning value

Section: 26.3.2.3 [lib.valarray.access] **Status**: NAD Future **Submitter:** Levente Farkas **Date:** 9 Sep 98

valarray:

```
T operator[] (size_t) const;
```

why not

```
const T& operator[] (size_t) const;
```

as in vector ???

One can't copy even from a const valarray eg:

```
memcpy(ptr, &v[0], v.size() * sizeof(double));
```

[I] find this bug in valarray is very difficult.

Rationale:

The LWG believes that the interface was deliberately designed that way. That is what valarray was designed to do; that's where the "value array" name comes from. LWG members further comment that "we don't want valarray to be a full STL container." 26.3.2.3 lib.valarray.access specifies properties that indicate "an absence of aliasing" for non-constant arrays; this allows optimizations, including special hardware optimizations, that are not otherwise possible.

---

78. Typo: event_call_back

Section: 27.4.2 lib.ios.base **Status**: DR **Submitter:** Nico Josuttis **Date:** 29 Sep 98

typo: event_call_back should be event_callback

Proposed Resolution:

In the 27.4.2 lib.ios.base synopsis change "event_call_back" to "event_callback".

---

79. Inconsistent declaration of polar()

Section: 26.2.1 lib.complex.synopsis, 26.2.7 lib.complex.value.ops **Status**: DR **Submitter:** Nico Josuttis **Date:** 29 Sep 98

In 26.2.1 lib.complex.synopsis polar is declared as follows:

```
template<class T> complex<T> polar(const T&, const T&);
```

In 26.2.7 lib.complex.value.ops it is declared as follows:

```
template<class T> complex<T> polar(const T& rho, const T& theta = 0);
```

Thus whether the second parameter is optional is not clear.

Proposed Resolution:

In 26.2.1 lib.complex.synopsis change:

```
template<class T> complex<T> polar(const T&, const T&);
```

to:

```
template<class T> complex<T> polar(const T& rho, const T& theta = 0);
```

---

80. Global Operators of complex declared twice

Section: 26.2.1 lib.complex.synopsis, 26.2.2 lib.complex **Status**: DR **Submitter:** Nico Josuttis **Date:** 29 Sep 98

Both 26.2.1 and 26.2.2 contain declarations of global operators for class complex. This redundancy should be removed.

Proposed Resolution:

Reduce redundance according to the general style of the standard.

---

81. Wrong declaration of slice operations

Section: 26.3.5 lib.template.slice.array, 26.3.7 lib.template.gslice.array, 26.3.8, 26.3.9 **Status**: NAD **Submitter:** Nico Josuttis **Date:** 29 Sep 98

Isn't the definition of copy constructor and assignment operators wrong? Instead of

```
slice_array(const slice_array&);
slice_array& operator=(const slice_array&);
```

IMHO they have to be

```
slice_array(const slice_array<T>&);
slice_array& operator=(const slice_array<T>&);
```

Same for gslice_array.

Rationale:

Not a defect. The Standard is correct as written.

---

82. Missing constant for set elements

Section: 23.1.2 lib.associative.reqmts **Status**: NAD **Submitter:** Nico Josuttis **Date:** 29 Sep 98

Paragraph 5 specifies:

For set and multiset the value type is the same as the key type. For map and multimap it is equal to pair<const Key, T>.

Strictly speaking, this is not correct because for set and multiset the value type is the same as the **constant** key type.

Rationale:

Not a defect. The Standard is correct as written; it uses a different mechanism (const &) for `set` and `multiset`. See issue 103 for a related issue.

---

84. Ambiguity with string::insert()

Section: 21.3.5 lib.string.modifiers **Status**: NAD Future **Submitter:** Nico Josuttis **Date:** 29 Sep 98

If I try

```
s.insert(0,1,' ');
```

I get an nasty ambiguity. It might be

```
s.insert((size_type)0,(size_type)1,(charT)' ');
```

which inserts 1 space character at position 0, or

```
s.insert((char*)0,(size_type)1,(charT)' ')
```

which inserts 1 space character at iterator/address 0 (bingo!), or

```
s.insert((char*)0, (InputIterator)1, (InputIterator)' ')
```

which normally inserts characters from iterator 1 to iterator ' '. But according to 23.1.1.9 (the "do the right thing" fix) it is equivalent to the second. However, it is still ambiguous, because of course I mean the first!

Rationale:

Not a defect. The LWG believes this is a "genetic misfortune" inherent in the design of string and thus not a defect in the Standard as such .

---

85. String char types

Section: 21 lib.strings **Status**: NAD **Submitter:** Nico Josuttis **Date:** 29 Sep 98

The standard seems not to require that charT is equivalent to traits::char_type. So, what happens if charT is not equivalent to traits::char_type ?

Rationale:

There is already wording in 21.1 paragraph 3 (lib.char.traits) that requires them to be the same.

---

87. Error in description of string::compare()

Section: 21.3.6.8 lib.string::compare **Status**: Dup **Submitter:** Nico Josuttis **Date:** 29 Sep 98

The following compare() description is obviously a bug:

```
int compare(size_type pos, size_type n1,
         charT *s, size_type n2 = npos) const;
```

because without passing n2 it should compare up to the end of the string instead of comparing npos characters (which throws an exception)

Rationale:

Duplicate; see issue 5.

---

88. Inconsistency between string::insert() and string::append()

Section: 21.3.5.4 lib.string::insert, 21.3.5.2 lib.string::append **Status**: NAD Future **Submitter:** Nico Josuttis **Date:** 29 Sep 98

Why does

```
template<class InputIterator>
     basic_string& append(InputIterator first, InputIterator last);
```

return a string, while

```
template<class InputIterator>
     void insert(iterator p, InputIterator first, InputIterator last);
```

returns nothing ?

Rationale:

The LWG believes this inconsistency is not sufficiently serious to constitute a defect.

---

89. Missing throw specification for string::insert() and string::replace()

Section: 21.3.5.4 lib.string::insert, 21.3.5.6 lib.string::replace **Status**: Dup **Submitter:** Nico Josuttis **Date:** 29 Sep 98

All insert() and replace() members for strings with an iterator as first argument lack a throw specification. The throw specification should probably be: length_error if size exceeds maximum.

Rationale:

Considered a duplicate because it will be solved by the resolution of issue 83.

---

90. Incorrect description of operator >> for strings

Section: 21.3.7.9 lib.string.io **Status**: DR **Submitter:** Nico Josuttis **Date:** 29 Sep 98

The effect of operator >> for strings containe the following item:

`isspace(c,getloc())` is true for the next available input character c.

Here `getloc()` has to be replaced by `is.getloc()`.

Proposed resolution:

In 21.3.7.9 lib.string.io paragraph 1 Effects clause replace:

> `isspace(c,getloc())` is true for the next available input character c.

with:

> `isspace(c,is.getloc())` is true for the next available input character c.

---

93. Incomplete Valarray Subset Definitions

Section: 26.3 lib.numarray **Status**: NAD Future **Submitter:** Nico Josuttis **Date:** 29 Sep 98

You can easily create subsets, but you can't easily combine them with other subsets. Unfortunately, you almost always needs an explicit type conversion to valarray. This is because the standard does not specify that valarray subsets provide the same operations as valarrays.

For example, to multiply two subsets and assign the result to a third subset, you can't write the following:

```
va[slice(0,4,3)] = va[slice(1,4,3)] * va[slice(2,4,3)];
```

Instead, you have to code as follows:

```
va[slice(0,4,3)] = static_cast<valarray<double> >(va[slice(1,4,3)]) *
                   static_cast<valarray<double> >(va[slice(2,4,3)]);
```

This is tedious and error-prone. Even worse, it costs performance because each cast creates a temporary objects, which could be avoided without the cast.

Proposed resolution:

Extend all valarray subset types so that they offer all valarray operations.

Ratinale:

This is not a defect in the Standard; it is a request for an extension.

---

95. Members added by the implementation

Section: 17.4.4.4 lib.member.functions **Status**: NAD. **Submitter:** AFNOR **Date:** 7 Oct 98

In 17.3.4.4/2 vs 17.3.4.7/0 there is a hole; an implementation could add virtual members a base class and break user derived classes.

Example:

```
// implementation code:
struct _Base { // _Base is in the implementer namespace
        virtual void foo ();
};
class vector : _Base // deriving from a class is allowed
{ ... };

// user code:
class vector_checking : public vector
{
        void foo (); // don't want to override _Base::foo () as the
                     // user doesn't know about _Base::foo ()
};
```

Proposed Resolution:

Clarify the wording to make the example illegal.

Rationale:

This is not a defect in the Standard. The example is already illegal. See 17.4.4.4 lib.member.functions paragraph 2.

---

97. Insert inconsistent definition

Section: 23 lib.containers **Status**: NAD Future **Submitter:** AFNOR **Date:** 7 Oct 98

`insert(iterator, const value_type&)` is defined both on sequences and on set, with unrelated semantics: insert here (in sequences), and insert with hint (in associative containers). They should have different names (B.S. says: do not abuse overloading).

Rationale:

This is not a defect in the Standard. It is a genetic misfortune of the design, for better or for worse.

---

99. Reverse_iterator comparisons completely wrong

Section: 24.4.1.3.13 lib.reverse.iter.op<, etc. **Status**: NAD **Submitter:** AFNOR **Date:** 7 Oct 98

The <, >, <=, >= comparison operator are wrong: they return the opposite of what they should.

Note: same problem in CD2, these were not even defined in CD1
SGI STL code is correct; this problem is known since the Morristown meeting but there it was too late

Rationale:

This is not a defect in the Standard. A careful reading shows the Standard is correct as written.

---

100. Insert iterators/ostream_iterators overconstrained

Section: 24.4.2 lib.insert.iterators, 24.5.4 lib.ostreambuf.iterator **Status**: NAD **Submitter:** AFNOR **Date:** 7 Oct 98

Overspecified For an insert iterator it, the expression *it is required to return a reference to it. This is a simple possible implementation, but as the SGI STL documentation says, not the only one, and the user should not assume that this is the case.

Rationale:

The LWG believes this causes no harm and is not a defect in the standard.

---

101. No way to free storage for vector and deque

Section: 23.2.4 lib.vector, 23.2.1 lib.deque **Status**: NAD **Submitter:** AFNOR **Date:** 7 Oct 98

Reserve can not free storage, unlike string::reserve

Rationale:

This is not a defect in the Standard. The LWG has considered this issue in the past and sees no need to change the Standard. Deque has no reserve() member function. For vector, shrink-to-fit can be expressed in a single line of code (where `v` is `vector<T>`):

```
vector<T>(v).swap(v);  // shrink-to-fit v
```

---

104. Description of basic_string::operator[] is unclear

Section: 21.3.4 lib.string.access **Status**: NAD **Submitter:** AFNOR **Date:** 7 Oct 98

It is not clear that undefined behavior applies when pos == size () for the non const version.

Proposed Resolution:

Rewrite as: Otherwise, if pos > size () or pos == size () and the non-const version is used, then the behavior is undefined.

Rationale:

The Standard is correct. The proposed resolution already appears in the Standard.

---

105. fstream ctors argument types desired

Section: 27.8 lib.file.streams **Status**: NAD Future **Submitter:** AFNOR **Date:** 7 Oct 98

fstream ctors take a const char* instead of string.
fstream ctors can't take wchar_t

An extension to add a const wchar_t* to fstream would make the implementation non conforming.

Rationale:

This is not a defect in the Standard. It might be an interesting extension for the next Standard.

---

106. Numeric library private members are implementation defined

Section: 26.3.5 lib.template.slice.array, etc. **Status**: DR **Submitter:** AFNOR **Date:** 7 Oct 98

This is the only place in the whole standard where the implementation has to document something private.

Proposed Resolution:

Remove the comment which says "// remainder implementation defined" from:

- 26.3.5 lib.template.slice.array
- 26.3.7 lib.template.gslice.array
- 26.3.8 lib.template.mask.array
- 26.3.9 lib.template.indirect.array

---

107. Valarray constructor is strange

Section: 26.3.2 lib.template.valarray **Status**: NAD **Submitter:** AFNOR **Date:** 7 Oct 98

The order of the arguments is (elem, size) instead of the normal (size, elem) in the rest of the library. Since elem often has an integral or floating point type, both types are convertible to each other and reversing them leads to a well formed program.

Rationale:

The LWG believes that while the order of arguments is unfortunate, it does not constitute a defect in the standard.

---

113. Missing/extra iostream sync semantics

Section: 27.6.1.1 lib.istream, 27.6.1.3 lib.istream.unformatted, para 36 **Status**: NAD **Submitter:** Steve Clamage **Date:** 13 Oct 98

In 27.6.1.1, class basic_istream has a member function sync, described in 27.6.1.3, paragraph 36.

Following the chain of definitions, I find that the various sync functions have defined semantics for output streams, but no semantics for input streams. On the other hand, basic_ostream has no sync function.

The sync function should at minimum be added to basic_ostream, for internal consistency.

A larger question is whether sync should have assigned semantics for input streams.

Classic iostreams said streambuf::sync flushes pending output and attempts to return unread input characters to the source. It is a protected member function. The filebuf version (which is public) has that behavior (it backs up the read pointer). Class strstreambuf does not override streambuf::sync, and so sync can't be called on a strstream.

If we can add corresponding semantics to the various sync functions, we should. If not, we should remove sync from basic_istream.

Rationale:

A sync function is not needed in basic_ostream because the flush function provides the desired functionality.

As for the other points, the LWG finds the standard correct as written.

---

116. bitset cannot be constructed with a const char*

Section: 23.3.5 lib.template.bitset **Status**: NAD Future **Submitter:** Judy Ward **Date:** 6 Nov 1998

The following code does not compile:

```
#include <bitset>
using namespace std;
bitset<32> b("111111111");
```

If you cast the ctor argument to a string, i.e.:

```
bitset<32> b(string("111111111"));
```

then it will compile. The reason is that bitset has the following templatized constructor:

```
template <class charT, class traits, class Allocator>
explicit bitset (const basic_string<charT, traits, Allocator>& str, ...);
```

According to the compiler vendor, the user cannot pass this template constructor a `const char*` and expect a conversion to `basic_string`. The reason is "When you have a template constructor, it can get used in contexts where type deduction can be done. Type deduction basically comes up with exact matches, not ones involving conversions."

I don't think the intention when this constructor became templatized was for construction from a `const char*` to no longer work.

Proposed Resolution:

Add to 23.3.5 lib.template.bitset a bitset constructor declaration

```
explicit bitset(const char*);
```

and in Section 23.3.5.1 lib.bitset.cons add:

```
explicit bitset(const char* str);
```

Effects:
Calls `bitset((string) str, 0, string::npos);`

Rationale:

Although the problem is real, the standard is designed that way so it is not a defect. Education is the immediate workaround. A future standard may wish to consider the Proposed Resolution as an extension.

.

---

128. Need open_mode() function for file stream, string streams, file buffers, and string buffers

Section: 27.7 lib.string.streams and 27.8 lib.file.streams **Status**: NAD Future **Submitter:** Angelika Langer **Date:** February 22, 1999

The following question came from Thorsten Herlemann:

> You can set a mode when constructing or opening a file-stream or filebuf, e.g. ios::in, ios::out, ios::binary, ... But how can I get that mode later on, e.g. in my own operator << or operator >> or when I want to check whether a file-stream or file-buffer object passed as parameter is opened for input or output or binary? Is there no possibility? Is this a design-error in the standard C++ library?

It is indeed impossible to find out what a stream's or stream buffer's open mode is, and without that knowledge you don't know how certain operations behave. Just think of the append mode.

Both streams and stream buffers should have a `mode()` function that returns the current open mode setting.

Proposed Resolution:

For stream buffers, add a function to the base class as a non-virtual function qualified as const to 27.5.2 lib.streambuf

```
openmode mode() const;
```

Returns the current open mode.

With streams, I'm not sure what to suggest. In principle, the mode could already be returned by `ios_base`, but the mode is only initialized for file and string stream objects, unless I'm overlooking anything. For this reason it should be added to the most derived stream classes. Alternatively, it could be added to `basic_ios` and would be default initialized in `basic_ios<>::init()`.

Rationale:

This might be an interesting extension for some future, but it is not a defect in the current standard. The Proposed

Resolution is retained for future reference.

---

130. Return type of container::erase(iterator) differs for associative containers

Section: 23.1.2 lib.associative.reqmts, 23.1.1 lib.sequence.reqmts **Status**: NAD Future **Submitter:** Andrew Koenig **Date:** 2 Mar 99

Table 67 (23.1.1) says that container::erase(iterator) returns an iterator. Table 69 (23.1.2) says that in addition to this requirement, associative containers also say that container::erase(iterator) returns void.

That's not an addition; it's a change to the requirements, which has the effect of making associative containers fail to meet the requirements for containers.

Rationale:

The LWG believes this was an explicit design decision by Alex Stepanov driven by complexity considerations. It has been previously discussed and reafirmed, so this is not a defect in the current standard. A future standard may wish to reconsider this issue.

---

131. list::splice throws nothing

Section: 23.2.2.4 lib.list.ops **Status:** NAD **Submitter:** Howard Hinnant **Date:** 6 Mar 99

What happens if a splice operation causes the size() of a list to grow beyond max_size()?

Rationale:

Size() cannot grow beyond max_size().

---

135. basic_iostream doubly initialized

Section: 27.6.1.5.1 lib.iostream.cons **Status:** NAD **Submitter:** Howard Hinnant **Date:** 6 Mar 99

-1- Effects Constructs an object of class basic_iostream, assigning initial values to the base classes by calling basic_istream<charT,traits>(sb) (lib.istream) and basic_ostream<charT,traits>(sb) (lib.ostream)

The called for basic_istream and basic_ostream constructors call init(sb). This means that the basic_iostream's virtual base class is initialized twice.

Proposed Resolution:

Change 27.6.1.5.1, paragraph 1 to:

-1- Effects Constructs an object of class basic_iostream, assigning initial values to the base classes by calling basic_istream<charT,traits>(sb) (lib.istream).

Rationale:

The LWG agreed that the init function is called twice, but said that this is harmless and so not a defect in the standard.

----- End of document -----