Doc. no. J16/00-0004
WG21 N1227
Date: 18 February 2000
Project: Programming Language C++
Reply to: *Beman Dawes <beman@esva.net>*

C++ Standard Library Defect Report List (Revision 12)

Committee Version

Reference ISO/IEC IS 14882:1998(E)

Also see:

- Table of Contents for all library issues.
- Index by Section for all library issues.
- Index by Status for all library issues.
- Library Active Issues List
- Library Closed 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 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.

Revision History

- R12 added "and paragraph 5" to the proposed resolution of issue 29.
- R11 added potential defects from Kona (99-0044/N1220), changed the proposed resolution of issue 4 to NAD, and changed the wording of proposed resolution of issue 38.

Defect Reports

---

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:

Not a defect. [Originally classified as a defect, later reclassified. See the rationale.]

Rationale:

basic_string, unlike the other standard library template containers, is severely constrained by its use of char_traits. Those types are dictated by the traits class, and are far from implementation defined.

[Kona: The LWG changed the proposed resolution of this issue for the reason given in the rationale above. The original resolution had been to make the two typedefs implementation defined.]

---

5. String::compare specification questionable

Section: 21.3.6.8 lib.string::compare **Status**: DR **Submitter:** Jack Reeves **Date:** 11 Dec 97

At the very end of the basic_string class definition is the signature: int compare(size_type pos1, size_type n1, const charT* s, size_type n2 = npos) const; In the following text this is defined as: returns basic_string<charT,traits,Allocator>(*this,pos1,n1).compare(basic_string<charT,traits,Allocator>(s,n2);

Since the constructor basic_string(const charT* s, size_type n, const Allocator& a = Allocator()) clearly requires that s != NULL and n < npos and further states that it throws length_error if n == npos, it appears the compare() signature above should always throw length error if invoked like so: str.compare(1, str.size()-1, s); where 's' is some null terminated character array.

This appears to be a typo since the obvious intent is to allow either the call above or something like: str.compare(1, str.size()-1, s, strlen(s)-1);

This would imply that what was really intended was two signatures int compare(size_type pos1, size_type n1, const

charT* s) const int compare(size_type pos1, size_type n1, const charT* s, size_type n2) const; each defined in terms of the corresponding constructor.

Proposed Resolution:

Replace the compare signature in 21.3 lib.basic.string (at the very end of the basic_string synopsis) which reads:

```
int compare(size_type pos1, size_type n1,
            const charT* s, size_type n2 = npos) const;
```

with:

```
int compare(size_type pos1, size_type n1,
            const charT* s) const;
int compare(size_type pos1, size_type n1,
            const charT* s, size_type n2) const;
```

Replace the portion of 21.3.6.8 lib.string::compare paragraphs 5 and 6 which read:

```
int compare(size_type pos, size_type n1,
            charT * s, size_type n2 = npos) const;
Returns:
basic_string<charT,traits,Allocator>(*this, pos, n1).compare(
             basic_string<charT,traits,Allocator>( s, n2))
```

with:

```
int compare(size_type pos, size_type n1,
            const charT * s) const;
Returns:
basic_string<charT,traits,Allocator>(*this, pos, n1).compare(
             basic_string<charT,traits,Allocator>( s ))

int compare(size_type pos, size_type n1,
            const charT * s, size_type n2) const;
Returns:
basic_string<charT,traits,Allocator>(*this, pos, n1).compare(
             basic_string<charT,traits,Allocator>( s, n2))
```

Editors please note that in addition to splitting the signature, the third argument becomes const, matching the existing synopsis.

Rationale:

While the LWG dislikes adding signatures, this is a clear defect in the Standard which must be fixed. The same problem was also identified in issues 7.5 and 87.

---

7. String clause minor problems

Section: 21 lib.strings **Status**: DR **Submitter:** Matt Austern **Date:** 15 Dec 97

(1) In 21.3.5.4 lib.string::insert, the description of template <class InputIterator> insert(iterator, InputIterator, InputIterator) makes no sense. It refers to a member function that doesn't exist. It also talks about the return value of a void function.

(2) Several versions of basic_string::replace don't appear in the class synopsis.

(3) basic_string::push_back appears in the synopsis, but is never described elsewhere. In the synopsis its agument is const charT, which doesn't makes much sense; it should probably be charT, or possible const charT&.

(4) basic_string::pop_back is missing.

(5) int compare(size_type pos, size_type n1, charT* s, size_type n2 = npos) make no sense. First, it's const charT* in the synopsis and charT* in the description. Second, given what it says in RETURNS, leaving out the final argument will always result in an exception getting thrown. This is paragraphs 5 and 6 of 21.3.6.8 lib.string::compare.

(6) In table 37, in section 21.1.1 lib.char.traits.require, there's a note for X::move(s, p, n). It says "Copies correctly even where p is in [s, s+n)". This is correct as far as it goes, but it doesn't go far enough; it should also guarantee that the copy is correct even where s in in [p, p+n). These are two orthogonal guarantees, and neither one follows from the other. Both guarantees are necessary if X::move is supposed to have the same sort of semantics as memmove (which was clearly the intent), and both guarantees are necessary if X::move is actually supposed to be useful.

Proposed Resolution:

ITEM 1: In 21.3.5.4 [lib.string::insert], change paragraph 16 to

EFFECTS: Equivalent to insert(p - begin(), basic_string(first, last)).

ITEM 2: Not a defect; the Standard is clear.. There are ten versions of replace() in the synopsis, and ten versions in 21.3.5.6 [lib.string::replace].

ITEM 3: Change the declaration of push_back in the string synopsis (21.3, [lib.basic.string]) from:

void push_back(const charT)

to

void push_back(charT)

Add the following text immediately after 21.3.5.2 [lib.string::append], paragraph 10.

void basic_string::push_back(charT c);
EFFECTS: Equivalent to append(static_cast<size_type>(1), c);

ITEM 4: Not a defect. The omission appears to have been deliberate.

ITEM 5: Duplicate; see issue 5 (and 87).

ITEM 6: In table 37, Replace:

"Copies correctly even where p is in [s, s+n)."

with:

"Copies correctly even where the ranges [p, p+n) and [s, s+n) overlap."

---

11. Bitset minor problems

Section: 23.3.5 lib.template.bitset **Status**: DR **Submitter:** Matt Austern **Date:** 22 Jan 98

(1) bitset<>::operator[] is mentioned in the class synopsis (23.3.5), but it is not documented in 23.3.5.2.

(2) The class synopsis only gives a single signature for bitset<>::operator[], reference operator[](size_t pos). This doesn't make much sense. It ought to be overloaded on const. reference operator[](size_t pos); bool operator[](size_t pos) const.

(3) Bitset's stream input function (23.3.5.3) ought to skip all whitespace before trying to extract 0s and 1s. The standard doesn't explicitly say that, though. This should go in the Effects clause.

Rationale:

The LWG believes Item 3 is not a defect. "Formatted input" implies the desired semantics. See 27.6.1.2 lib.istream.formatted.

Proposed Resolution:

ITEMS 1 AND 2:

In the bitset synopsis (23.3.5, [lib.template.bitset]), replace the member function

```
reference operator[](size_t pos);
```

with the two member functions

```
bool operator[](size_t pos) const;
reference operator[](size_t pos);
```

Add the following text at the end of 23.3.5.2 [lib.bitset.members], immediately after paragraph 45:

```
bool operator[](size_t pos) const;
```
Requires: pos is valid
Throws: nothing
Returns: `test(pos)`

```
bitset<N>::reference operator[](size_t pos);
```
Requires: pos is valid
Throws: nothing
Returns: An object of type `bitset<N>::reference` such that `(*this)[pos] == this->test(pos)`, and such that `(*this)[pos] = val` is equivalent to `this->set(pos, val);`

---

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:

[R12: modified to include paragraph 5.]

In 27.3.1 [lib.narrow.stream.objects] paragraph 2 and 5, 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 *by Esa Pulkkinen* 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 the public static member `id` as defined in (22.1.1.1.2, [lib.locale.facet]).

[Kona: strike as overspecification the text "(not inherits)" from the resolution, which read "... whose definition contains (not inherits) the public static member `id` ..."]

---

39. istreambuf_iterator<>::operator++(int) 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.

---

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.

---

48. Use of non-existent exception constructor

Section: 27.4.2.1.1 lib.ios::failure **Status**: DR **Submitter:** Matt Austern **Date:** 21 Jun 98

27.4.2.1.1, paragraph 2, says that class failure initializes the base class, exception, with exception(msg). Class exception (see 18.6.1) has no such constructor.

Proposed Resolution:

Replace 27.4.2.1.1 [lib.ios::failure], paragraph 2, with

> EFFECTS: Constructs an object of class `failure`.

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 *[Matt Austern, lib-6528]* 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 *[Beman Dawes]* comments:

This was US issue CD2-23-011; it was accepted in London *but the change was not made due to an editing oversight*. 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".

---

54. Basic_streambuf's destructor

Section: 27.5.2.1 lib.streambuf.cons **Status**: DR **Submitter:** Matt Austern **Date:**25 Jun 98

The class synopsis for basic_streambuf shows a (virtual) destructor, but the standard doesn't say what that destructor does. My assumption is that it does nothing, but the standard should say so explicitly.

Proposed Resolution:

Add after 27.5.2.1 lib.streambuf.cons paragraph 2:

```
virtual  ~basic_streambuf();
```

Effects: None.

---

55. Invalid stream position is undefined

Section: 27 lib.input.output **Status**: DR **Submitter:** Matt Austern **Date:**26 Jun 98

Several member functions in clause 27 are defined in certain circumstances to return an "invalid stream position", a term that is defined nowhere in the standard. Two places (27.5.2.4.2, paragraph 4, and 27.8.1.4, paragraph 15) contain a cross-reference to a definition in _lib.iostreams.definitions_, a nonexistent section.

I suspect that the invalid stream position is just supposed to be pos_type(-1). Probably best to say explicitly in (for example) 27.5.2.4.2 that the return value is pos_type(-1), rather than to use the term "invalid stream position", define that term somewhere, and then put in a cross-reference.

The phrase "invalid stream position" appears ten times in the C++ Standard. In seven places it refers to a return value, and it should be changed. In three places it refers to an argument, and it should not be changed. Here are the three places where "invalid stream position" should not be changed:

27.7.1.3 [lib.stringbuf.virtuals], paragraph 14
27.8.1.4 [lib.filebuf.virtuals], paragraph 14
D.7.1.3 [depr.strstreambuf.virtuals], paragraph 17

Proposed Resolution:

In 27.5.2.4.2 [lib.streambuf.virt.buffer], paragraph 4, change "Returns an object of class pos_type that stores an invalid stream position (_lib.iostreams.definitions_)" to "Returns `pos_type(off_type(-1))`".

In 27.5.2.4.2 [lib.streambuf.virt.buffer], paragraph 6, change "Returns an object of class pos_type that stores an invalid stream position" to "Returns `pos_type(off_type(-1))`".

In 27.7.1.3 [lib.stringbuf.virtuals], paragraph 13, change "the object stores an invalid stream position" to "the return value is `pos_type(off_type(-1))`".

In 27.8.1.4 [lib.filebuf.virtuals], paragraph 13, change "returns an invalid stream position (27.4.3)" to "returns `pos_type(off_type(-1))`"

In 27.8.1.4 [lib.filebuf.virtuals], paragraph 15, change "Otherwise returns an invalid stream position (_lib.iostreams.definitions_)" to "Otherwise returns `pos_type(off_type(-1))`"

In D.7.1.3 [depr.strstreambuf.virtuals], paragraph 15, change "the object stores an invalid stream position" to "the return value is `pos_type(off_type(-1))`"

In D.7.1.3 [depr.strstreambuf.virtuals], paragraph 18, change "the object stores an invalid stream position" to "the return value is `pos_type(off_type(-1))`"

---

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.

Jerry reports that the 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.

---

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?

Jerry Schwarz: There is apparently an editing glitch. You'll notice that the last item of the list of what stops extraction doesn't make any sense. It was supposed to be the line that said a null is stored.

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.

---

69. Must elements of a vector be contiguous?

Section: 23.2.4 lib.vector **Status**: DR **Submitter:** Andrew Koenig **Date:** 29 Jul 1998

The issue is this:

> Must the elements of a vector be in contiguous memory?

(Please note that this is entirely separate from the question of whether a vector iterator is required to be a pointer; the answer to that question is clearly "no," as it would rule out debugging implementations)

Proposed Resolution:

Add the following text to the end of 23.2.4 [lib.vector], paragraph 1.

> The elements of a vector are stored contiguously, meaning that if v is a `vector<T, Allocator>` where T is some type other than `bool`, then it obeys the identity `&v[n] == &v[0] + n` for all `0 <= n < v.size()`.

Rationale:

The LWG feels that as a practical matter the answer is clearly "yes". There was considerable discussion as to the best way to express the concept of "contiguous", which is not directly defined in the standard. Discussion included:

- An operational definition similar to the above proposed resolution is already used for valarray (26.3.2.3).
- There is no need to explicitly consider a user-defined operator& because elements must be copyconstructible (23.1 para 3) and copyconstructible (20.1.3) specifies requirements for operator&.
- There is no issue of one-past-the-end because of language rules.

---

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, *Valentin Bonnard* 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;
```

---

74. Garbled text for `codecvt::do_max_length`

Section: 22.2.1.5.2 lib.locale.codecvt.virtuals **Status**: DR **Submitter:** Matt Austern **Date:** 18 Sep 98

The text of `codecvt::do_max_length`'s "Returns" clause (22.2.1.5.2, paragraph 11) is garbled. It has unbalanced parentheses and a spurious **n**.

Proposed Resolution:

Replace 22.2.1.5.2 lib.locale.codecvt.virtuals paragraph 11 with the following:

> Returns: The maximum value that `do_length(state, from, from_end, 1)` can return for any valid range `[from, from_end)` and `stateT` value state. The specialization `codecvt<char, char, mbstate_t>::do_max_length()` returns 1.

---

75. Contradiction in `codecvt::length`'s argument types

Section: 22.2.1.5 lib.locale.codecvt **Status**: DR **Submitter:** Matt Austern **Date:** 18 Sep 98

The class synopses for classes `codecvt<>` (22.2.1.5) and `codecvt_byname<>` (22.2.1.6) say that the first parameter of the member functions `length` and `do_length` is of type `const stateT&`. The member function descriptions, however (22.2.1.5.1, paragraph 6; 22.2.1.5.2, paragraph 9) say that the type is `stateT&`. Either the synopsis or the summary must be changed.

If (as I believe) the member function descriptions are correct, then we must also add text saying how `do_length` changes its `stateT` argument.

Proposed Resolution:

In 22.2.1.5 [lib.locale.codecvt], and also in 22.2.1.6 [lib.locale.codecvt_byname], change the `stateT` argument type on both member `length()` and member `do_length()` from

```
const stateT&
```

to

```
stateT&
```

In 22.2.1.5.2 [lib.locale.codecvt.virtuals], add to the definition for member `do_length` a paragraph:

> Effects: The effect on the `state` argument is ``as if'' it called `do_in(state, from, from_end, from, to, to+max, to)` for `to` pointing to a buffer of at least `max` elements.

---

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 redundancy according to the general style of the standard.

---

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.

---

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

---

110. istreambuf_iterator::equal not const

Section: 24.5.3 [lib.istreambuf.iterator], 24.5.3.5 [lib.istreambuf.iterator::equal] **Status**: DR **Submitter:** Nathan Myers **Date:** 15 Oct 98

Member istreambuf_iterator<>::equal is not declared "const", yet 24.5.3.6 [lib.istreambuf.iterator::op==] says that operator==, which is const, calls it. This is contradictory.

Proposed Resolution:

In 24.5.3 [lib.istreambuf.iterator] and also in 24.5.3.5 [lib.istreambuf.iterator::equal], replace:

```
bool equal(istreambuf_iterator& b);
```

with:

```
bool equal(const istreambuf_iterator& b) const;
```

---

124. ctype_byname<charT>::do_scan_is & do_scan_not return type should be const charT*

Section: 22.2.1.2 lib.locale.ctype.byame **Status:** DR **Submitter:** Judy Ward **Date:** 15 Dec 1998

In Section 22.2.1.2 lib.locale.ctype.byame ctype_byname<charT>::do_scan_is() and do_scan_not() are declared to return a const char* not a const charT*.

Proposed Resolution:

Change Section 22.2.1.2 lib.locale.ctype.byame `do_scan_is()` and `do_scan_not()` to return a `const charT*`.

---

125. valarray<T>::operator!() return type is inconsistent

Section: 26.3.2 lib.template.valarray **Status:** DR **Submitter:** Judy Ward **Date:** 15 Dec 1998

In Section 26.3.2 lib.template.valarray valarray<T>::operator!() is declared to return a valarray<T>, but in Section 26.3.2.5 lib.valarray.unary it is declared to return a valarray<bool>. The latter appears to be correct.

Proposed Resolution:

Change in Section 26.3.2 lib.template.valarray the declaration of `operator!()` so that the return type is `valarray<bool>`.

---

126. typos in Effects clause of ctype::do_narrow()

Section: 22.2.1.1.2 lib.locale.ctype.virtuals **Status:** DR **Submitter:** Judy Ward **Date:** 15 Dec 1998

Proposed Resolution:

In Section 22.2.1.1.2 lib.locale.ctype.virtuals change:

```
do_widen(do_narrow(c),0) == c
```

to:

```
do_widen(do_narrow(c,0)) == c
```

and change:

```
(is(M,c) || !ctc.is(M, do_narrow(c),dfault) )
```

to:

```
(is(M,c) || !ctc.is(M, do_narrow(c,dfault)) )
```

---

132. list::resize description uses random access iterators

Section: 23.2.2.2 lib.list.capacity **Status:** DR **Submitter:** Howard Hinnant **Date:** 6 Mar 99

The description reads:

-1- Effects:

```
if (sz > size())
  insert(end(), sz-size(), c);
else if (sz < size())
  erase(begin()+sz, end());
else
  ;                           //  do nothing
```

Obviously list::resize should not be specified in terms of random access iterators.

Proposed Resolution:

Change 23.2.2.2 paragraph 1 to:

Effects:

```
if (sz > size())
  insert(end(), sz-size(), c);
else if (sz < size())
{
  iterator i = begin();
  advance(i, sz);
  erase(i, end());
}
```

[Dublin: The LWG asked Howard to discuss exception safety offline with David Abrahams. They had a discussion and believe there is no issue of exception safety with the proposed resolution.]

---

133. map missing get_allocator()

Section: 23.3.1 lib.map **Status:** DR **Submitter:** Howard Hinnant **Date:** 6 Mar 99

The title says it all.

Proposed Resolution:

Insert in 23.3.1 [lib.map], paragraph 2, after operator= in the map declaration:

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

---

139. Optional sequence operation table description unclear

Section: 23.1.1 lib.sequence.reqmts **Status**: DR **Submitter:** Andrew Koenig **Date:** 30 Mar 99

The sentence introducing the Optional sequence operation table (23.1.1 paragraph 12) has two problems:

A. It says ``The operations in table 68 are provided only for the containers for which they take constant time."

That could be interpreted in two ways, one of them being ``Even though table 68 shows particular operations as being provided, implementations are free to omit them if they cannot implement them in constant time."

B. That paragraph says nothing about amortized constant time, and it should.

Proposed Resolution:

Replace the wording in 23.1.1 paragraph 12 which begins ``The operations in table 68 are provided only..." with:

> Table 68 lists sequence operations that are provided for some types of sequential containers but not others. An implementation shall provide these operations for all container types shown in the ``container" column, and shall implement them so as to take amortized constant time.

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