Doc. no. J16/99-0016R1
WG21 N1193
Date: 21 April 1999
Project: Programming Language C++

C++ Standard Library Issues List (Revision 8)

Reference ISO/IEC IS 14882:1998(E)

Also see:

The purpose of this document is to record the status of issues which have come before the Library Working Group (LWG) of th ANSI (J16) and ISO (WG21) C++ Standards Committee. Issues represent potential defects in the ISO/IEC IS 14882:1998(E) document. Issues are not to be used to request new features or other extensions.

The issues on this list are not necessarily formal ISO Defect Reports (DR's). While some issues will eventually be elevated to Defect Report status, other issues will be disposed of in other ways. See Issue Status.

This document is in an experimental format designed for both viewing via a world-wide web browser and hard-copy printing. It is available as an HTML file for browsing or PDF file for printing.

This issues list exists in two slightly different versions; the Committee Version and the Public Version. The Committee Version is the master copy, while the Public Version is an extract with certain names, email addresses, action items, and internal committee comments removed. A line of text reading "Committee Version" following the title above identifies the Committee Version

For the most current public version of this document seehttp://www.dkuug.dk/jtc1/sc22/wg21. Requests for further information about this document should include the document number above, reference ISO/IEC 14882:1998(E), and be submitted to Information Technology Industry Council (ITI), 1250 Eye Street NW, Washington, DC 20005.

Public information as to how to obtain a copy of the C++ Standard, join the standards committee, submit an issue, or comment on an issue can be found in the C++ FAQ athttp://reality.sgi.com/austern_mti/std-c++/faq.html. Public discussion of C++ Standard related issues occurs onnews:comp.std.c++.

For committee members, files available on the committee's private web site include the HTML version of the Standard itself. HTML hyperlinks from this issues list to those files will only work for committee members who have downloaded them into the same disk directory as the issues list files.

Revision history

- R8: post-Dublin mailing. Updated to reflect LWG and full committee actions in Dublin. (21 Apr 99)
- R7: pre-Dublin updated: Added issues 130, 131, 132, 133, 134, 135, 136, 137, 138, 139 (31 Mar 99)
- R6: pre-Dublin mailing. Added issues 127, 128, and 129. (22 Feb 99)
- R5: update issues 103, 112; added issues 114 to 126. Format revisions to prepare for making list public. (30 Dec 98)

- R4: post-Santa Cruz II updated: Issues 110, 111, 112, 113 added, several issues corrected. (22 Oct 98)
- R3: post-Santa Cruz II: Issues 94 to 109 added, many issues updated to reflect LWG consensus (12 Oct 98)
- R2: pre-Santa Cruz II: Issues 73 to 93 added, issue 17 updated. (29 Sep 98)
- R1: Correction to issue 55 resolution, 60 code format, 64 title. (17 Sep 98)

Issue Status

New - The issue has not yet been reviewed by the LWG. Any **Proposed Resolution** is purely a suggestion from the issue submitter, and should not be construed as the view of LWG.

Open - The LWG has discussed the issue but is not yet ready to move the issue forward. There are several possible reasons for open status:

- Consensus may have not yet have been reached as to how to deal with the issue.
- Informal consensus may have been reached, but the LWG awaits exact **Proposed Resolution** wording for review.
- The LWG wishes to consult additional technical experts before proceeding.
- The issue may require further study.

A **Proposed Resolution** for an open issue is still not be construed as the view of LWG. Comments on the current state of discussions are often given at the end of open issues in an . Such comments are for information only and should not be given undue importance. They do not appear in the public version.

Dup - The LWG has reached consensus that the issue is a duplicate of another issue, and will not be further dealt with. A **Rationale** identities the duplicated issue's issue number.

NAD - The LWG has reached consensus that the issue is not a defect in the Standard, and the issue is ready to forward to the full committee as a proposed record of response. A **Rationale** discusses the LWG's reasoning.

Review - Exact wording of a **Proposed Resolution** is now available for review on an issue for which the LWG previously reached informal consensus.

Ready - The LWG has reached consensus that the issue is a defect in the Standard, the **Proposed Resolution** is correct, and the issue is ready to forward to the full committee for further action as a Defect Report (DR).

DR - (Defect Report) - The full J16 committee has voted to forward the issue to the Project Editor to be processed as a Potential Defect Report. The Project Editor reviews the issue, and then forwards it to the WG21 Convenor, who returns it to the full committee for final disposition. This issues list accords the status of DR to all these Defect Reports regardless of where they are in that process.

TC - (Technical Corrigenda) - The full WG21 committee has voted to accept the Defect Report's Proposed Resolution as a Technical Corrigenda. Action on this issue is thus complete and no further action is possible under ISO rules.

RR - (Record of Response) - The full WG21 committee has determined that this issue is not a defect in the Standard. Action on this issue is thus complete and no further action is possible under ISO rules.

Future - In addition to the regular status, the LWG believes that this issue should be revisited at the next revision of the standard. It is usually paired with NAD.

Issues are always given the status of New when they first appear on the issues list. They may progress to Open or Review while the LWG is actively working on them. When the LWG has reached consensus on the disposition of an issue, the status will then change to Dup, NAD, or Ready as appropriate. Once the full J16 committee votes to forward Ready issues to the Project Editor, they are given the status of Defect Report (DR). These in turn may become the basis for Technical Corrigenda

(TC), or are closed without action other than a Record of Response (RR). The intent of this LWG process is that only issues which are truly defects in the Standard move to the formal ISO DR status.

---

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()".

---

3. Atexit registration during atexit() call is not described

Section: 18.3 lib.support.start.term **Status:** Open **Submitter:** Steve Clamage **Date:** 12 Dec 97 **Msg:** lib-6500

We appear not to have covered all the possibilities of exit processing with respect to atexit registration.

Example 1: (C and C++)

```
#include <stdlib.h>
```

```
void f1() { }
void f2() { atexit(f1); }

int main()
{
    atexit(f2); // the only use of f2
    return 0; // for C compatibility
}
```

At program exit, f2 gets called due to its registration in main. Running f2 causes f1 to be newly registered during the exit processing. Is this a valid program? If so, what are its semantics?

Interestingly, neither the C standard, nor the C++ draft standard nor the forthcoming C9X Committee Draft says directly whether you can register a function with atexit during exit processing.

All 3 standards say that functions are run in reverse order of their registration. Since f1 is registered last, it ought to be run first, but by the time it is registered, it is too late to be first.

If the program is valid, the standards are self-contradictory about its semantics.

Example 2: (C++ only)

```
void F() { static T t; } // type T has a destructor

int main()
{
    atexit(F); // the only use of F
}
```

Function F registered with atexit has a local static variable t, and F is called for the first time during exit processing. A local static object is initialized the first time control flow passes through its definition, and all static objects are destroyed during exit processing. Is the code valid? If so, what are its semantics?

Section 18.3 "Start and termination" says that if a function F is registered with atexit before a static object t is initialized, F will not be called until after t's destructor completes.

In example 2, function F is registered with atexit before its local static object O could possibly be initialized. On that basis, it must not be called by exit processing until after O's destructor completes. But the destructor cannot be run until after F is called, since otherwise the object could not be constructed in the first place.

If the program is valid, the standard is self-contradictory about its semantics.

I plan to submit Example 1 as a public comment on the C9X CD, with a recommendation that the results be undefined. (Alternative: make it unspecified. I don't think it is worthwhile to specify the case where f1 itself registers additional functions, each of which registers still more functions.)

I think we should resolve the situation in the whatever way the C committee decides.

For Example 2, I recommend we declare the results undefined.

Proposed Resolution:

---

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
```

---

5. String::compare specification questionable

Section: 21.3.6.8 lib.string::compare **Status:** Ready **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.

---

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.

---

7. String clause minor problems

Section: 21 lib.strings **Status:** Ready **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."

---

8. Locale::global lacks guarantee

Section: 22.1.1.5 lib.locale.statics **Status:** Open **Submitter:** Matt Austern **Date:** 24 Dec 97

It appears there's an important guarantee missing from clause 22. We're told that invoking locale::global(L) sets the C locale if L has a name. However, we're not told whether or not invoking setlocale(s) sets the global C++ locale.

The intent, I think, is that it should not, but I can't find any such words anywhere.

Proposed Resolution:

Add note in 22.1.1.5 lib.locale.statics : "the library shall behave as if no other library function calls locale::global()."

---

9. Operator new(0) calls should not yield the same pointer

Section: 18.4.1 lib.new.delete **Status:** Open **Submitter:** Steve Clamage **Date:** 4 Jan 98

comp.std.c++ posting: I just noticed that section 3.7.3.1 of CD2 seems to allow for the possibility that all calls to operator new(0) yield the same pointer, an implementation technique specifically prohibited by ARM 5.3.3.Was this prohibition really lifted? Does the FDIS agree with CD2 inthe regard? [Issues list maintainer's note: the IS is the same.]

Proposed Resolution:

---

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 issue 19.

11. Bitset minor problems

Section: 23.3.5 lib.template.bitset **Status:** Ready **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[]m 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);`

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].

---

17. Bad bool parsing

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

This section describes the process of parsing a text boolean value from the input stream. It does not say it recognizes either of the sequences "true" or "false" and returns the corresponding bool value; instead, it says it recognizes only one of those sequences, and chooses which according to the received value of a reference argument intended for returning the result, and reports an error if the other sequence is found. (!) Furthermore, it claims to get the names from the ctype<> facet rather than the numpunct<> facet, and it examines the "boolalpha" flag wrongly; it doesn't define the value "loc"; and finally, it computes wrongly whether to use numeric or "alpha" parsing.

I believe the correct algorithm is "as if":

```
// in, err, val, and str are arguments.
err = 0;
const numpunct<charT>& np = use_facet<numpunct<charT> >(str.getloc());
const string_type t = np.truename(), f = np.falsename();
bool tm = true, fm = true;
size_t pos = 0;
while (tm && pos < t.size() || fm && pos < f.size()) {
  if (in == end) { err = str.eofbit; }
  bool matched = false;
  if (tm && pos < t.size()) {
    if (!err && t[pos] == *in) matched = true;
    else tm = false;
  }
  if (fm && pos < f.size()) {
    if (!err && f[pos] == *in) matched = true;
    else fm = false;
  }
  if (matched) { ++in; ++pos; }
```

```
    if (pos > t.size()) tm = false
    if (pos > f.size()) fm = false;
  }
  if (tm == fm || pos == 0) { err |= str.failbit; }
  else                      { val = tm; }
  return in;
```

Notice this works reasonably when the candidate strings are both empty, or equal, or when one is a substring of the other. The proposed text below captures the logic of the code above.

Proposed Resolution:

In 22.2.2.1.2 [lib.facet.num.get.virtuals], in the first line of paragraph 14, change "&&" to "&".

Then, replace paragraphs 15 and 16 as follows:

> Otherwise target sequences are determined "as if" by calling the members _falsename()_ and _truename()_ of the facet obtained by _use_facet >(str.getloc())_. Successive characters in the range _[in,end)_ (see [lib.sequence.reqmts]) are obtained and matched against corresponding positions in the target sequences only as necessary to identify a unique match. The input iterator _in_ is compared to _end_ only when necessary to obtain a character. If and only if a target sequence is uniquely matched, _val_ is set to the corresponding value.
>
> The _in_ iterator is always left pointing one position beyond the last character successfully matched. If _val_ is set, then err is set to _str.goodbit_; or to _str.eofbit_ if, when seeking another character to match, it is found that _(in==end)_. If _val_ is not set, then _err_ is set to _str.failbit_; or to _(str.failbit|str.eofbit)_ if the reason for the failure was that _(in==end)_. [Example: for targets _true_:"a" and _false_:"abb", the input sequence "a" yields _val==true_ and _err==str.eofbit_; the input sequence "abc" yields _err=str.failbit_, with _in_ ending at the 'c' element. For targets _true_:"1" and _false_:"0", the input sequence "1" yields _val==true_ and _err=str.goodbit_. For empty targets (""), any input sequence yields _err==str.failbit_. --end example]

---

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].

---

19. "Noconv" definition too vague

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

In the definitions of codecvt<>::do_out and do_in, they are specified to return noconv if "no conversion is needed". This definition is too vague, and does not say normatively what is done with the buffers.

Proposed Resolution:

Change the entry for noconv in the table under paragraph 4 in section 22.2.1.5.2 [lib.locale.codecvt.virtuals] to read:

> noconv: input sequence is identical to converted sequence.

and change the Note in paragraph 2 to normative text as follows:

> If returns _noconv_, the converted sequence is identical to the input sequence _[from,from_next)_. _to_next_ is set equal to _to_, and the value of _state_ is unchanged.

---

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".

---

21. Codecvt_byname<> instantiations

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

In the second table in the section, captioned "Required instantiations", the instantiations for codecvt_byname<> have been omitted. These are necessary to allow users to construct a locale by name from facets.

Proposed Resolution:

Add in 22.1.1.1.1 [lib.locale.category] to the table captioned "Required instantiations", in the category "ctype" the lines

```
codecvt_byname<char,char,mbstate_t>,
codecvt_byname<wchar_t,char,mbstate_t>
```

---

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()`; ..."

26. Bad sentry example

Section: 27.6.1.1.2 lib.istream::sentry **Status:** Open **Submitter:** Nathan Myers **Date:** 6 Aug 98

In paragraph 6, the code in the example:

```
template <class charT, class traits = char_traits<charT> >
basic_istream<charT,traits>::sentry(
         basic_istream<charT,traits>& is, bool noskipws = false) {
    ...
    int_type c;
    typedef ctype<charT> ctype_type;
    const ctype_type& ctype = use_facet<ctype_type>(is.getloc());
    while ((c = is.rdbuf()->snextc()) != traits::eof()) {
      if (ctype.is(ctype.space,c)==0) {
        is.rdbuf()->sputbackc (c);
        break;
      }
    }
    ...
 }
```

fails to demonstrate correct use of the facilities described. In particular, it fails to use traits operators, and specifies incorrect semantics. (E.g. it specifies skipping over the first character in the sequence without examining it.)

Proposed Resolution:

Replace the example with better code, as follows:

```
template <class charT, class traits>
basic_istream<charT,traits>::sentry::sentry(
```

```
    basic_istream<charT,traits>& is, bool noskipws
{
  typedef ctype<charT> ctype_type;
  const ctype_type& ct = use_facet<ctype_type>(is.getloc());
  for (int_type c = is.rdbuf()->sgetc();
       !traits::eq_int_type(c,traits::eof()) && ct.is(ct.space,c);
       c = is.rdbuf()->snextc())
    {}
}
```

---

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>".

---

31. Immutable locale values

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

Paragraph 6, says "An instance of _locale_ is *immutable*; once a facet reference is obtained from it, ...". This has caused some confusion, because locale variables are manifestly assignable.

Proposed Resolution:

In 22.1.1 [lib.locale] replace paragraph 6,

> An instance of locale is immutable; once a facet reference is obtained from it, that reference remains usable as long as the locale value itself exists.

with

> A locale value is immutable. This means that once a facet reference is obtained from a locale object by calling use_facet<>, that reference remains usable, and the results from member functions of it may be cached and re-used, until the locale object is assigned to or destroyed.

---

32. Pbackfail description inconsistent

Section: 27.5.2.4.4 lib.streambuf.virt.pback **Status:** Review **Submitter:** Nathan Myers **Date:** 6 Aug 98

The description of the required state before calling virtual member basic_streambuf<>::pbackfail requirements is inconsistent with the conditions described in 27.5.2.2.4 [lib.streambuf.pub.pback] where member sputbackc calls it. Specifically, the latter says it calls pbackfail if:

traits::eq(c,gptr()[-1]) is false

where pbackfail claims to require:

traits::eq(*gptr(),traits::to_char_type(c)) returns false

It appears that the pbackfail description is wrong.

Proposed Resolution:

In 27.5.2.4.4 [lib.streambuf.virt.pback], paragraph 1, change:

> "traits::eq(*gptr(),traits::to_char_type(c)) "

to

> "traits::eq(traits::to_char_type(c),gptr()[-1]) "

Rationale:

Note deliberate reordering of arguments for clarity in addition to the correction of the argument value.

---

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.

---

41. Ios_base needs clear(), exceptions()

Section: 27.4.2 [lib.ios.base] **Status:** Review **Submitter:** Nathan Myers **Date:** 6 Aug 98

The description of ios_base::iword() and pword() in 27.4.2.4 [lib.ios.members.static], say that if they fail, they "set badbit, which may throw an exception". However, ios_base offers no interface to set or to test badbit; those interfaces are defined in basic_ios<>.

Proposed Resolution:

Change the description in 27.4.2.5 [lib.ios.members.storage] in paragraph 2, and also in paragraph 4, as follows. Replace

> If the function fails it sets badbit, which may throw an exception.

with

> If the function fails, and `*this` is a base subobject of a `basic_ios<>` object or subobject, the effect is equivalent to calling `basic_ios<>::setstate(failbit)` on the derived object (which may throw `failure`).

---

42. String ctors specify wrong default allocator

Section: 21.3 [lib.basic.string] **Status**: Ready **Submitter:** Nathan Myers **Date:** 6 Aug 98

The basic_string<> copy constructor:

```
basic_string(const basic_string& str, size_type pos = 0,
             size_type n = npos, const Allocator& a = Allocator());
```

specifies an Allocator argument default value that is counter-intuitive. The natural choice for a the allocator to copy from is str.get_allocator(). Though this cannot be expressed in default-argument notation, overloading suffices.

Alternatively, the other containers in Clause 23 (deque, list, vector) do not have this form of constructor, so it is inconsistent, and an evident source of confusion, for basic_string<> to have it, so it might better be removed.

Proposed Resolution:

In 21.3 [lib.basic.string], replace the declaration of the copy constructor as follows:

```
basic_string(const basic_string& str, size_type pos = 0,
             size_type n = npos);
basic_string(const basic_string& str, size_type pos,
             size_type n, const Allocator& a);
```

In 21.3.1 [lib.string.cons], replace the copy constructor declaration as above. Add to paragraph 5, Effects:

> When no `Allocator` argument is provided, the string is constructed using the value `str.get_allocator()`.

Rationale:

The LWG believes the constructor is actually broken, rather than just an unfortunate design choice.

The LWG considered two other possible resolutions:

B. In 21.3 [lib.basic.string], and also in 21.3.1 [lib.string.cons], replace the declaration of the copy constructor as follows:

```
basic_string(const basic_string& str, size_type pos = 0,
             size_type n = npos);
```

C. In 21.3 [lib.basic.string], replace the declaration of the copy constructor as follows:

```
basic_string(const basic_string& str);
basic_string(const basic_string& str, size_type pos, size_type n = npos,
             const Allocator& a = Allocator());
```

In 21.3.1 [lib.string.cons], replace the copy constructor declaration as above. Add to paragraph 5, Effects:

In the first form, the Allocator value used is copied from `str.get_allocator()`.

The proposed resolution reflects the original intent of the LWG. It was also noted that this fix "will cause a small amount of existing code to now work correctly."

---

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.

---

44. Iostreams use operator== on int_type values

Section: 27 [lib.input.output] **Status:** Open **Submitter:** Nathan Myers **Date:** 6 Aug 98

Many of the specifications for iostreams specify that character values or their int_type equivalents are compared using operators == or !=, though in other places traits::eq() or traits::eq_int_type is specified to be used throughout. This is an inconsistency; we should change uses of == and != to use the traits members instead.

Proposed Resolution:

---

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.

---

48. Use of non-existent exception constructor

Section: 27.4.2.1.1 lib.ios::failure **Status:** Ready **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`.

---

49. Underspecification of ios_base::sync_with_stdio

Section: 27.4.2.4 lib.ios.members.static **Status:** Open **Submitter:** Matt Austern **Date:** 21 Jun 98

Two problems.

(1) 27.4.2.4 doesn't say what ios_base::sync_with_stdio(f) returns. Does it return f, or does it return the previous synchronization state? My guess is the latter, but the standard doesn't say so.

(2) 27.4.2.4 doesn't say what it means for streams to be synchronized with stdio. Again, of course, I can make some guesses. (And I'm unhappy about the performance implications of those guesses, but that's another matter.)

Proposed Resolution:

Change the following sentenance in 27.4.2.4 lib.ios.members.static returns clause from:

> `true` if the standard iostream objects (27.3) are synchronized and otherwise returns `false`.

to:

> `true` if the previous state of the standard iostream objects (27.3) was synchronized and otherwise returns `false`.

---

50. Copy constructor and assignment operator of ios_base

Section: 27.4.2 lib.ios.base **Status:** Open **Submitter:** Matt Austern **Date:** 21 Jun 98

As written, ios_base has a copy constructor and an assignment operator. (Nothing in the standard says it doesn't have one, and all classes have copy constructors and assingment operators unless you take specific steps to avoid them.) However, nothin in 27.4.2 says what the copy constructor and assignment operator do.

My guess is that this was an oversight, that ios_base is, like basic_ios, not supposed to have a copy constructor or an assignment operator.

A LWG member comments: Yes, its an oversight, but in the opposite sense to what you're suggesting. At one point there was a definite intention that you could copy ios_base. It's an easy way to save the entire state of a stream for future use. As you note, to carry out that intention would have required a explicit description of the semantics (e.g. what happens to the iarray and parray stuff).

Proposed Resolution:

---

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".

---

53. Basic_ios destructor unspecified

Section: 27.4.4.1 lib.basic.ios.cons, 27.4.4.2 lib.basic.ios.members **Status:** Ready **Submitter:** Matt Austern **Date:** 23 Jun 98

There's nothing in 27.4.4 saying what basic_ios's destructor does.

The important question is whether basic_ios::~basic_ios() destroys rdbuf().

Proposed Resolution:

Add after 27.4.4.1 lib.basic.ios.cons paragraph 2:

```
virtual ~basic_ios();
```

Notes: The destructor does not destroy `rdbuf()`.

Add a footnote to 27.4.4.2 lib.basic.ios.members paragraph 6, rdbuf effects, which says:

`rdbuf(0)` does not set `badbit`.

54. Basic_streambuf's destructor

Section: 27.5.2.1 lib.streambuf.cons **Status**: Ready **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**: Ready **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..." .

---

58. Extracting a char from a wide-oriented stream

Section: 27.6.1.2.3 lib.istream::extractors **Status:** Open **Submitter:** Matt Austern **Date:** 1 Jul 98

27.6.1.2.3 has member functions for extraction of signed char and unsigned char, both singly and as strings. However, it doesn't say what it means to extract a `char` from a `basic_streambuf<charT, Traits>`

basic_streambuf, after all, has no members to extract a char, so basic_istream must somehow convert from charT to signed char or unsigned char. The standard doesn't say how it is to perform that conversion.

Proposed Resolution:

`operator>>` should use `narrow` to convert from `charT` to `char`.

---

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.

---

60. What is a formatted input function?

Section: 27.6.1.2.1 lib.istream.formatted.reqmts **Status**: Open **Submitter:** Matt Austern **Date:** 3 Aug 98

Paragraph 1 of 27.6.1.2.1 contains general requirements for all formatted input functions. Some of the functions defined in section 27.6.1.2 explicitly say that those requirements apply ("Behaves like a formatted input member (as described in 27.6.1.2.1)"), but others don't. The question: is 27.6.1.2.1 supposed to apply to everything in 27.6.1.2, or only to those member functions that explicitly say "behaves like a formatted input member"? Or to put it differently: are we to assume that everything that appears in a section called "Formatted input functions" really is a formatted input function? I assume that 27.6.1.2.1 is intended to apply to the arithmetic extractors (27.6.1.2.2), but I assume that it is not intended to apply to extractors like

```
basic_istream& operator>>(basic_istream& (*pf)(basic_istream&));
```

and

```
basic_istream& operator>>(basic_streammbuf*);
```

There is a similar ambiguity for unformatted input, formatted output, and unformatted output.

Comments : It seems like the problem is that the basic_istream and basic_ostream operator <<()'s that are used for the manipulators and streambuf* are in the wrong section and should have their own separate section or be modified to make it clear that the "Common requirements" listed in section 27.6.1.2.1 (for basic_istream) and section 27.6.2.5.1 (for basic_ostream) do not apply to them.

Proposed Resolution:

The three member functions described in paragraphs 1-5 and the one described in paragraph 12-14 of section 27.6.1.2.3 should each have something added (perhaps a Notes clause?) that says: "The common requirements listed in section 27.6.1.2.1 do not apply to this function."

The four member functions described in paragraphs 1-9 of section 27.6.2.5.3 should each have something added (perhaps a Notes clause?) and the one described in section that says: "The common requirements listed in section 27.6.2.5.1 do not apply to this function."

---

61. Ambiguity in iostreams exception policy

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

The introduction to the section on unformatted input (27.6.1.3) says that every unformatted input function catches all exceptions that were thrown during input, sets badbit, and then conditionally rethrows the exception. That seems clear enough. Several of the specific functions, however, such as get() and read(), are documented in some circumstances as setting eofbit and/or failbit. (The standard notes, correctly, that setting eofbit or failbit can sometimes result in an exception being thrown.) The question: if one of these functions throws an exception triggered by setting failbit, is this an exception "thrown during input" and hence covered by 27.6.1.3, or does 27.6.1.3 only refer to a limited class of exceptions? Just to make this concrete, suppose you have the following snippet.

```
char buffer[N];
istream is;
...
is.exceptions(istream::failbit); // Throw on failbit but not on badbit.
is.read(buffer, N);
```

Now suppose we reach EOF before we've read N characters. What iostate bits can we expect to be set, and what exception (if any) will be thrown?

Proposed Resolution:

Clarify that the phrase "thrown during input" refers only to exceptions thrown by streambuf's overridden virtuals, not exceptions thrown as part of istream's error-reporting mechanism.

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`".

63. Exception-handling policy for unformatted output

Section: 27.6.2.6 lib.ostream.unformatted **Status:** Open **Submitter:** Matt Austern **Date:** 11 Aug 98

Clause 27 details an exception-handling policy for formatted input, unformatted input, and formatted output. It says nothing for unformatted output (27.6.2.6). 27.6.2.6 should either include the same kind of exception-handling policy as in the other three places, or else it should have a footnote saying that the omission is deliberate.

Proposed Resolution:

Add an exception-handling policy similar to the one in 27.6.2.5.1 lib.ostream.formatted.reqmts, paragraph 1. The omission seems to have been unintentional.

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.

65. Underspecification of strstreambuf::seekoff

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

The standard says how this member function affects the current stream position. (`gptr` or `pptr`) However, it does not say how this member function affects the beginning and end of the get/put area.

This is an issue when seekoff is used to position the get pointer beyond the end of the current read area. (Which is legal. This is implicit in the definition of *seekhigh* in D.7.1, paragraph 4.)

Proposed Resolution:

---

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.

---

69. Must elements of a vector be contiguous?

Section: 23.2.4 lib.vector **Status**: Ready **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, 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 for resolution

---

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.

---

74. Garbled text for `codecvt::do_max_length`

Section: 22.2.1.5.2 lib.locale.codecvt.virtuals **Status:** Ready **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:** Ready **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.

---

76. Can a `codecvt` facet always convert one internal character at a time?

Section: 22.2.1.5 lib.locale.codecvt **Status:** Open **Submitter:** Matt Austern **Date:** 25 Sep 98

This issue concerns the requirements on classes derived from `codecvt`, including user-defined classes. What are the restrictions on the conversion from external characters (e.g. `char`) to internal characters (e.g. `wchar_t`)? Or, alternatively, what assumptions about `codecvt` facets can the I/O library make?

The question is whether it's possible to convert from internal characters to external characters one internal character at a time, and whether, given a valid sequence of external characters, it's possible to pick off internal characters one at a time. Or, to put it differently: given a sequence of external characters and the corresponding sequence of internal characters, does a position in the internal sequence correspond to some position in the external sequence?

To make this concrete, suppose that `[first, last)` is a sequence of *M* external characters and that `[ifirst, ilast)` is the corresponding sequence of *N* internal characters, where *N > 1*. That is, `my_encoding.in()`, applied to `[first, last)`, yields `[ifirst, ilast)`. Now the question: does there necessarily exist a subsequence of external characters, `[first, last_1)`, such that the corresponding sequence of internal characters is the single character `*ifirst`?

(What a "no" answer would mean is that `my_encoding` translates sequences only as blocks. There's a sequence of *M* external characters that maps to a sequence of *N* internal characters, but that external sequence has no subsequence that maps to *N-1* internal characters.)

Some of the wording in the standard, such as the description of `codecvt::do_max_length` (22.2.1.5.2, paragraph 11) and `basic_filebuf::underflow` (27.8.1.4, paragraph 3) suggests that it must always be possible to pick off internal

characters one at a time from a sequence of external characters. However, this is never explicitly stated one way or the other.

This issue seems (and is) quite technical, but it is important if we expect users to provide their own encoding facets. This is an area where the standard library calls user-supplied code, so a well-defined set of requirements for the user-supplied code is crucial. Users must be aware of the assumptions that the library makes. This issue affects positioning operations on `basic_filebuf`, unbuffered input, and several of`codecvt`'s member functions.

Proposed Resolution:

---

77. Valarray operator[] const returning value

Section: 26.3.2.3 [lib.valarray.access] **Status:** NAD Future **Submitter:** **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.

---

83. String::npos vs. string::max_size()

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

Many string member functions throw if size is getting or exceeding npos. However, I wonder why they don't throw if size is getting or exceeding max_size() instead of npos. May be npos is known at compile time, while max_size() is known at runtime. However, what happens if size exceeds max_size() but not npos, then ? It seems the standard lacks some clarifications here.

Proposed Resolution:

---

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 ambigious, 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.

---

86. String constructors don't describe exceptions

Section: 21.3.1 lib.string.cons **Status:** Open **Submitter:** Nico Josuttis **Date:** 29 Sep 98

The constructor from a range:

```
template<class InputIterator>
        basic_string(InputIterator begin, InputIterator end,
                     const Allocator& a = Allocator());
```

lacks a throw specification. However, I would expect that it throws according to the other constructors if the numbers of characters in the range equals npos (or exceeds max_size(), see above).

Proposed resolution:

---

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.

---

91. Description of operator>> and getline() for string<> might cause endless loop

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

Operator >> and getline() for strings read until eof() in the input stream is true. However, this might never happen, if the stream can't read anymore without reachin EOF. So shouldn't it be changed into that it reads until !good() ?

Proposed resolution:

---

92. Incomplete Algorithm Requirements

Section: 25 lib.algorithms **Status:** Open **Submitter:** Nico Josuttis **Date:** 29 Sep 98

The standard does not state, how often a function object is copied, called, or the order of calls inside an algorithm. This may lead to suprising/buggy behavior. Consider the following example:

```
class Nth {    // function object that returns true for the nth element
  private:
    int nth;     // element to return true for
    int count;   // element counter
  public:
    Nth (int n) : nth(n), count(0) {
    }
    bool operator() (int) {
        return ++count == nth;
    }
};
....
// remove third element
    list<int>::iterator pos;
    pos = remove_if(coll.begin(),coll.end(),  // range
                    Nth(3)),                  // remove criterion
    coll.erase(pos,coll.end());
```

This call, in fact removes the 3rd **AND the 6th** element. This happens because the usual implementation of the algorithm copies the function object internally:

```
template <class ForwIter, class Predicate>
ForwIter std::remove_if(ForwIter beg, ForwIter end, Predicate op)
{
    beg = find_if(beg, end, op);
    if (beg == end) {
        return beg;
    }
    else {
        ForwIter next = beg;
        return remove_copy_if(++next, end, beg, op);
    }
}
```

The algorithm uses find_if() to find the first element that should be removed. However, it then uses a copy of the passed function object to process the resulting elements (if any). Here, Nth is used again and removes also the sixth element. This behavior compromises the advantage of function objects being able to have a state. Without any cost it could be avoided (just impolement it directly instead of calling find_if()).

Proposed resolution:

The standard should specify that this kind of implementation is a bug. Something like "it is guaranteed that an algorithm uses the same object for all calls of passed function objects (however, it may be a copy)".

---

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.

---

94. May library implementors add template parameters to Standard Library classes?

Section: 17.4.4 lib.conforming **Status**: Open **Submitter:** Matt Austern **Date:** 22 Jan 98

Is it a permitted extension for library implementors to add template parameters to standard library classes, provided that those extra parameters have defaults? For example, instead of defining `template <class T, class Alloc = allocator<T> > class vector;` defining it as `template <class T, class Alloc = allocator<T>, int N = 1> class vector;`

The standard may well already allow this (I can't think of any way that this extension could break a conforming program, considering that users are not permitted to forward-declare standard library components), but it ought to be explicitly permitted or forbidden.

Proposed Resolution:

Add a new subclause [presumably 17.4.4.9] following 17.4.4.8 [lib.res.on.exception.handling]:

> 17.4.4.9 Template Parameters
>
> A specialization of a template class described in the C++ Standard Library behaves the same as if the implementation declares no additional template parameters.
>
> Footnote/ Additional template parameters with default values are thus permitted.

Add "template parameters" to the list of subclauses at the end of 17.4.4 paragraph 1 [lib.conforming].

Rationale:

The LWG believes the answer should be "yes, adding template parameters with default values should be permitted." A careful reading of 17.4.4 and its subclauses found no mention of additional template parameters.

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.

---

96. Vector<bool> is not a container

Section: 23.2.5 lib.vector.bool **Status:** Open **Submitter:** AFNOR **Date:** 7 Oct 98

`vector<bool>` is not a container as its reference and pointer types are not references and pointers.

Also it forces everyone to have a space optimization instead of a speed one.

See also: 99-0008 == N1185 Vector<bool> is Nonconforming, Forces Optimization Choice.

Proposed Resolution:

---

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.

---

98. Input iterator requirements are badly written

Section: 24.1.1 lib.input.iterators **Status:** Open **Submitter:** AFNOR **Date:** 7 Oct 98

Table 72 in 24.1.1 (lib.input.iterators) specifies semantics for `*r++` of:

```
{ T tmp = *r; ++r; return tmp; }
```

This does not work for pointers and overconstrains implementors.

Proposed Resolution:

Add for *r++: "To call the copy constructor for the type T is allowed but not required."

---

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
```

---

102. Bug in insert range in associative containers

Section: 23.1.2 lib.associative.reqmts **Status:** Open **Submitter:** AFNOR **Date:** 7 Oct 98

Table 69 of Containers say that a.insert(i,j) is linear if [i, j) is ordered. It seems impossible to implement, as it means that if [i, j) = [x], insert in an associative container is O(1)!

Proposed Resolution:

N+log (size()) if [i,j) is sorted according to value_comp()

---

103. set::iterator is required to be modifiable, but this allows modification of keys

Section: 23.1.2 lib.associative.reqmts, 23.3.3 lib.set, 23.3.4 lib.mutliset **Status:** Open **Submitter:** AFNOR **Date:** 7 Oct 98

Set::iterator is described as implementation-defined with a reference to the container requirement; the container requirement says that const_iterator is an iterator pointing to const T and iterator an iterator pointing to T.

23.1.2 paragraph 2 implies that the keys should not be modified to break the ordering of elements. But that is not clearly specified. Especially considering that the current standard requires that iterator for associative containers be different from const_iterator. Set, for example, has the following:

```
typedef implementation defined iterator;
       // See _lib.container.requirements_
```

23.1 lib.container.requirements actually requires that iterator type pointing to T (table 65). Disallowing user modification of keys by changing the standard to require an iterator for associative container to be the same as const_iterator would be overkill since that will unnecessarily significantly restrict the usage of associative container. A class to be used as elements of set, for example, can no longer be modified easily without either redesigning the class (using mutable on fields that have nothing to do with ordering), or using const_cast, which defeats requiring iterator to be const_iterator. The proposed solution goes in line with trusting user knows what he is doing.

Proposed Resolution:

Option A. In 23.1.2 lib.associative.reqmts , paragraph 2, after first sentence, and before "In addition,...", add one line:

> Modification of keys shall not change their strict weak ordering.

Option B. Add three new sentences to 23.1.2 lib.associative.reqmts :

> At the end of paragraph 5: "Keys in an associative container are immutable." At the end of paragraph 6: "For associative containers where the value type is the same as the key type, both `iterator` and `const_iterator` are constant iterators. It is unspecified whether or not `iterator` and `const_iterator` are the same type."

Option C: To 23.1.2 lib.associative.reqmts , paragraph 3, which currently reads:

> The phrase ``equivalence of keys'' means the equivalence relation imposed by the comparison and not the operator== on keys. That is, two keys k1 and k2 in the same container are considered to be equivalent if for the comparison object comp, comp(k1, k2) == false && comp(k2, k1) == false.

add the following:

> For any two keys k1 and k2 in the same container, comp(k1, k2) shall return the same value whenever it is evaluated. [Note: If k2 is removed from the container and later reinserted, comp(k1, k2) must still return a consisent value but this value may be different than it was the first time k1 and k2 were in the same container. This is intended to allow usage like a string key that contains a filename, where comp compares file contents; if k2 is removed, the file is changed, and the same k2 (filename) is reinserted, comp(k1, k2) must again return a consistent value but this value may be different than it was the previous time k2 was in the container.]

Rationale:

Simply requiring that keys be immutable is not sufficient, because the comparison object may indirectly (via pointers) operate on values outside of the keys. Furthermore, requiring that keys be immutable places undue restrictions on `set` for structures where only a portion of the structure participates in the comparison.

---

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.

---

108. Lifetime of exception::what() return unspecified

Section: 18.6.1 lib.exception para 8, 9 **Status:** Review **Submitter:** AFNOR **Date:** 7 Oct 98

The lifetime of the return value of exception::what() is left unspecified. This issue has implications with exception safety of exception handling: some exceptions should not throw bad_alloc.

Proposed Resolution:

Add to 18.6.1 lib.exception paragraph 9 (exception::what notes clause) the sentence:

> The return value remains valid until the exception object from which it is obtained is destroyed or a non-const member function of the exception object is called.

---

109. Missing binders for non-const sequence elements

Section: 20.3.6 lib.binders **Status:** Open **Submitter:** Bjarne Stroustrup **Date:** 7 Oct 98

There are no versions of binders that apply to non-const elements of a sequence. This makes examples like for_each() using bind2nd() on page 521 of "The C++ Programming Language (3rd)" non-conforming. Suitable versions of the binders need to be added.

Proposed Resolution:

---

110. istreambuf_iterator::equal not const

Section: 24.5.3 [lib.istreambuf.iterator], 24.5.3.5 [lib.istreambuf.iterator::equal] **Status:** Ready **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;
```

---

111. istreambuf_iterator::equal overspecified, inefficient

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

The member istreambuf_iterator<>::equal is specified to be unnecessarily inefficient. While this does not affect the efficiency of conforming implementations of iostreams, because they can "reach into" the iterators and bypass this function, it does affect users who use istreambuf_iterators.

The inefficiency results from a too-scrupulous definition, which requires a "true" result if neither iterator is at eof. In practice these iterators can only usefully be compared with the "eof" value, so the extra test implied provides no benefit, but slows down users' code.

The solution is to weaken the requirement on the function to return true only if both iterators are at eof.

Proposed Resolution:

Replace 24.5.3.5 [lib.istreambuf.iterator::equal], paragraph 1,

> -1- Returns: true if and only if both iterators are at end-of-stream, or neither is at end-of-stream, regardless of what streambuf object they use.

with

> -1- Returns: true if and only if both iterators are at end-of-stream, regardless of what streambuf object they use.

---

112. Minor typo in `ostreambuf_iterator` constructor

Section: 24.5.4.1 lib.ostreambuf.iter.cons **Status**: Review **Submitter:** Matt Austern **Date:** 20 Oct 98

The **requires** clause for `ostreambuf_iterator`'s constructor from an `ostream_type` (24.5.4.1, paragraph 1) reads "*s* is not null". However, *s* is a reference, and references can't be null.

Proposed Resolution:

In 24.5.4.1 lib.ostreambuf.iter.cons:

Move the current paragraph 1, which reads "Requires: s is not null.", from the first constructor to the second constructor.

Insert a new paragraph 1 Requires clause for the first constructor reading:

> **Requires**: `s.rdbuf()` is not null.

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.

---

114. Placement forms example in error twice

Section: 18.4.1.3 [lib.new.delete.placement] **Status:** Review **Submitter:** Steve Clamage **Date:** 28 Oct 1998

Section 18.4.1.3 contains the following example:

```
[Example: This can be useful for constructing an object at a known address:
        char place[sizeof(Something)];
        Something* p = new (place) Something();
 -end example]
```

First code line: "place" need not have any special alignment, and the following constructor could fail due to misaligned data.

Second code line: Aren't the parens on Something() incorrect? [Dublin: the LWG believes the () are correct.]

Examples are not normative, but nevertheless should not show code that is invalid or likey to fail.

Proposed Resolution:

Replace the first line of code in the example in 18.4.1.3 [lib.new.delete.placement] with:

```
void* place = operator new(sizeof(Something));
```

---

115. Typo in strstream constructors

Section: D.7.4.1 [depr.strstream.cons] **Status:** Review **Submitter:** Steve Clamage **Date:** 2 Nov 1998

D.7.4.1 strstream constructors paragraph 2 says:

> Effects: Constructs an object of class strstream, initializing the base class with iostream(& sb) and initializing sbwith one of the two constructors:
>
> - If mode&app==0, then s shall designate the first element of an array of n elements. The constructor is strstreambuf(s, n, s).
>
> - If mode&app==0, then s shall designate the first element of an array of n elements that contains an NTBS whose first element is designated by s. The constructor is strstreambuf(s, n, s+std::strlen(s)).

Notice the second condition is the same as the first. I think the second condition should be "If mode&app==app", or "mode&app!=0", meaning that the append bit is set.

Proposed Resolution:

In D.7.3.1 [depr.ostrstream.cons] paragraph 2 and D.7.4.1 [depr.strstream.cons] paragraph 2, change the first condition to `(mode&app)==0` and the second condition to `(mode&app)!=0`.

---

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.

.

---

117. `basic_ostream` uses nonexistent `num_put` member functions

Section: 27.6.2.5.2 lib.ostream.inserters.arithmetic **Status:** Review **Submitter:** Matt Austern **Date:** 20 Nov 98

The **effects** clause for numeric inserters says that insertion of a value `x`, whose type is either `bool`, `short`, `unsigned short`, `int`, `unsigned int`, `long`, `unsigned long`, `float`, `double`, `long double`, or `const void*`, is delegated to `num_put`, and that insertion is performed as if through the following code fragment:

```
bool failed = use_facet<
   num_put<charT,ostreambuf_iterator<charT,traits> >
   >(getloc()).put(*this, *this, fill(), val). failed();
```

This doesn't work, because `num_put<>::put` is only overloaded for the types `bool`, `long`, `unsigned long`, `double`, `long double`, and `const void*`. That is, the code fragment in the standard is incorrect (it is diagnosed as ambiguous at compile time) for the types `short`, `unsigned short`, `int`, `unsigned int`, and `float`.

We must either add new member functions to `num_put`, or else change the description in `ostream` so that it only calls functions that are actually there. I prefer the latter.

Proposed Resolution:

Replace 27.6.2.5.2, paragraph 1 with the following:

The classes `num_get<>` and `num_put<>` handle locale-dependent numeric formatting and parsing. These inserter functions use the imbued `locale` value to perform numeric formatting. When `val` is of type `bool`, `long`, `unsigned long`, `double`, `long double`, or `const void*`, the formatting conversion occurs as if it performed the following code fragment:

```
bool failed = use_facet<
   num_put<charT,ostreambuf_iterator<charT,traits> >
   >(getloc()).put(*this, *this, fill(), val). failed();
```

When `val` is of type `short` or `int` the formatting conversion occurs as if it performed the following code fragment:

```
bool failed = use_facet<
   num_put<charT,ostreambuf_iterator<charT,traits> >
   >(getloc()).put(*this, *this, fill(), static_cast<long>(val)). failed();
```

When `val` is of type `unsigned short` or `unsigned int` the formatting conversion occurs as if it performed the following code fragment:

```
bool failed = use_facet<
   num_put<charT,ostreambuf_iterator<charT,traits> >
   >(getloc()).put(*this, *this, fill(), static_cast<unsigned long>(val)). f
```

When `val` is of type `float` the formatting conversion occurs as if it performed the following code fragment:

```
bool failed = use_facet<
   num_put<charT,ostreambuf_iterator<charT,traits> >
   >(getloc()).put(*this, *this, fill(), static_cast<double>(val)). failed();
```

---

118. `basic_istream` uses nonexistent `num_get` member functions

Section: 27.6.1.2.2 lib.istream.formatted.arithmetic **Status:** Open **Submitter:** Matt Austern **Date:** 20 Nov 98

Formatted input is defined for the types `short`, `unsigned short`, `int`, `unsigned int`, `long`, `unsigned long`, `float`, `double`, `long double`, `bool`, and `void*`. According to section 27.6.1.2.2, formatted input of a value `x` is done as if by the following code fragment:

```
typedef num_get< charT,istreambuf_iterator<charT,traits> > numget;
iostate err = 0;
use_facet< numget >(loc).get(*this, 0, *this, err, val);
setstate(err);
```

According to section 22.2.2.1.1 lib.facet.num.get.members, however, `num_get<>::get()` is only overloaded for the types `bool`, `long`, `unsigned short`, `unsigned int`, `unsigned long`, `unsigned long`, `float`, `double`, `long double`, and `void*`. Comparing the lists from the two sections, we find that 27.6.1.2.2 is using a nonexistent function for types `short` and `int`.

Proposed Resolution:

Add `short` and `int` overloads for `num_get<>::get()`

119. Should virtual functions be allowed to strengthen the exception specification?

Section: 17.4.4.8 lib.res.on.exception.handling **Status:** Ready **Submitter:** Judy Ward **Date:** 15 Dec 1998

Section 17.4.4.8 lib.res.on.exception.handling states:

"An implementation may strengthen the exception-specification for a function by removing listed exceptions."

The problem is that if an implementation is allowed to do this for virtual functions, then a library user cannot write a class that portably derives from that class.

For example, this would not compile if ios_base::failure::~failure had an empty exception specification:

```
#include <ios>
#include <string>

class D : public std::ios_base::failure {
public:
        D(const std::string&);
        ~D(); // error - exception specification must be compatible with
              // overridden virtual function ios_base::failure::~failure()
};
```

Proposed Resolution:

Change Section 17.4.4.8 lib.res.on.exception.handling from:

"may strengthen the exception-speciification for a function"

to:

"may strengthen the exception-specification for a non-virtual function".

120. Can an implementor add specializations?

Section: 17.4.3.1 lib.reserved.names **Status:** Open **Submitter:** Judy Ward **Date:** 15 Dec 1998

Section 17.4.3.1 says:

> It is undefined for a C++ program to add declarations or definitions to namespace std or namespaces within namespace std unless otherwise specified. A program may add template specializations for any standard library template to namespace std. Such a specialization (complete or partial) of a standard library template results in undefined behavior unless the declaration depends on a user-defined name of external linkage and unless the specialization meets the standard library requirements for the original template...

This implies that it is ok for library users to add specializations, but not implementors. A user program can actually detect this, for example, the following manual instantiation will not compile if the implementor has made ctype<wchar_t> a specialization:

```
#include <locale>
#include <wchar.h>

template class std::ctype<wchar_t>; // can't be specialization
```

Proposed Resolution:

Add to 17.4.4 lib.conforming a section called Specializations with wording:

An implementation can define additional specializations for any of the template classes or functions in the standard library if a use of any of these classes or functions behaves as if the implementation did not define them.

---

121. Detailed definition for ctype<wchar_t> specialization missing

Section: 22.1.1.1.1 lib.locale.category **Status:** Open **Submitter:** Judy Ward **Date:** 15 Dec 1998

Section 22.1.1.1.1 has the following listed in Table 51: ctype<char> , ctype<wchar_t>.

Also Section 22.2.1.1 lib.locale.ctype says:

> The instantiations required in Table 51 (22.1.1.1.1) namely ctype<char> and ctype<wchar_t> , implement character classing appropriate to the implementation's native character set.

However, Section 22.2.1.3 lib.facet.ctype.special only has a detailed description of the ctype<char> specialization, not the ctype<wchar_t> specialization.

Proposed Resolution:

Add the ctype<wchar_t> detailed class description to Section 22.2.1.3 lib.facet.ctype.special .

---

122. streambuf/wstreambuf description should not say they are specializations

Section: 27.5.2 lib.streambuf **Status:** Open **Submitter:** Judy Ward **Date:** 15 Dec 1998

Section 27.5.2 describes the streambuf classes this way:

The class streambuf is a specialization of the template class basic_streambuf specialized for the type char.

The class wstreambuf is a specialization of the template class basic_streambuf specialized for the type wchar_t.

This implies that these classes must be template specializations, not typedefs.

It doesn't seem this was intended, since Section 27.5 has them declared as typedefs.

Proposed Resolution:

Remove the two sentences above, since the streambuf synopsis already has a declaration for the typedefs.

---

123. Should valarray helper arrays fill functions be const?

Section: 26.3.5.4 lib.slice.arr.fill , 26.3.7.4 lib.gslice.array.fill , 26.3.8.4 lib.mask.array.fill , 26.3.9.4 lib.indirect.array..fill
Status: Open **Submitter:** Judy Ward **Date:** 15 Dec 1998

One of the operator= in the valarray helper arrays is const and one is not. For example, look at slice_array. This operator= in Section 26.3.5.2 lib.slice.arr.assign is const:

```
void operator=(const valarray<T>&) const;
```

but this one in Section 26.3.5.4 lib.slice.arr.fill , is not:

```
void operator=(const T&);
```

The description of the semantics for these two functions is similar.

Proposed Resolution:

Make the `operator=(const T&)` versions of slice_array, gslice_array, indirect_array, and mask_array `const` member functions.

---

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:** Ready **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:** Ready **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:** Ready **Submitter:** Judy Ward **Date:** 15 Dec 1998

In Section 22.2.1.1.2 lib.locale.ctype.virtuals the following typos need to be fixed:

```
      do_widen(do_narrow(c),0) == c
should be:
      do_widen(do_narrow(c,0)) == c

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

Proposed Resolution:

Fix as suggested above

---

127. auto_ptr<> conversion issues

Section: 20.4.5 lib.auto.ptr **Status:** Ready **Submitter:** Greg Colvin **Date:** 17 Feb 99

There are two problems with the current `auto_ptr` wording in the standard:

First, the `auto_ptr_ref` definition cannot be nested because `auto_ptr<Derived>::auto_ptr_ref` is unrelated to `auto_ptr<Base>::auto_ptr_ref`

Second, there is no `auto_ptr` assignment operator taking an `auto_ptr_ref` argument.

I have discussed these problems with my proposal coauthor, Bill Gibbons, and with some compiler and library implementers, and we believe that these problems are not desired or desirable implications of the standard.

Proposed Resolution:

In 20.4.5 lib.auto.ptr, paragraph 2, move the `auto_ptr_ref` definition to namespace scope.

In 20.4.5 lib.auto.ptr, paragraph 2, add an assignment operator to the `auto_ptr` definition:

```
auto_ptr& operator=(auto_ptr_ref<X> r) throw();
```

Also add the assignment operator to 20.4.5.3 lib.auto.ptr.conv:

```
auto_ptr& operator=(auto_ptr_ref<X> r) throw()

Effects: Calls reset(p) for the auto_ptr p that r holds.
Returns: *this.
```

---

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.

---

129. Need error indication from seekp() and seekg()

Section: 27.6.1.3 lib.istream.unformatted and 27.6.2.4 lib.istream.seeks **Status:** Review **Submitter:** Angelika Langer **Date:** February 22, 1999

Currently, the standard does not specify how seekg() and seekp() indicate failure. They are not required to set failbit, and they can't return an error indication because they must return *this, i.e. the stream. Hence, it is undefined what happens if they fail. And they _can_ fail, for instance, when a file stream is disconnected from the underlying file (is_open()==false) or when a wide charaacter file stream must perform a state-dependent code conversion, etc.

The stream functions seekg() and seekp() should set failbit in the stream state in case of failure.

Proposed Resolution:

Add to the Effects: clause of seekg() in 27.6.1.3 lib.istream.unformatted and to the Effects: clause of seekp() in 27.6.2.4 lib.istream.seeks:

> In case of failure, the function calls `setstate(failbit)` (which may throw `ios_base::failure`).

---

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().

---

132. list::resize description uses random access iterators

Section: 23.2.2.2 lib.list.capacity **Status:** Ready **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());
}
```

---

133. map missing get_allocator()

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

The title says it all.

Proposed Resolution:

Insert:

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

after operator= in 23.3.1, paragraph 2, in the map declaration.

---

134. vector and deque constructors over specified

Section: 23.2.4.1 lib.vector.cons **Status:** Open **Submitter:** Howard Hinnant **Date:** 6 Mar 99

The complexity description says: "It does at most 2N calls to the copy constructor of T and logN reallocations if they are just input iterators ...".

This appears to be overly restrictive, dictating the precise memory/performance tradeoff for the implementor.

Proposed Resolution:

Change 23.2.1.1, paragraph 6 to:

-6- Complexity: If the iterators first and last are forward iterators, bidirectional iterators, or random access iterators the constructor makes only N calls to the copy constructor, and performs no reallocations, where N is last - first. It makes order N calls to the copy constructor of T and order log N reallocations if they are input iterators.*

And change 23.2.4.1, paragraph 1 to:

-1- Complexity: The constructor template <class InputIterator> vector(InputIterator first, InputIterator last) makes only N calls to the copy constructor of T (where N is the distance between first and last) and no reallocations if iterators first and last are of forward, bidirectional, or random access categories. It makes order N calls to the copy constructor of T and order logN reallocations if they are just input iterators, since it is impossible to determine the distance between first and last and then do copying.

---

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.

---

136. seekp, seekg setting wrong streams?

Section: 27.6.1.3 lib.istream.unformatted **Status:** Open **Submitter:** Howard Hinnant **Date:** 6 Mar 99

I may be misunderstanding the intent, but should not seekg set only the input stream and seekp set only the output stream? The description seems to say that each should set both input and output streams. If that's really the intent, I withdraw this proposal.

Proposed Resolution:

In section 27.6.1.3 change:

```
basic_istream<charT,traits>& seekg(pos_type pos);
Effects: If fail() != true, executes rdbuf()->pubseekpos(pos).
```

To:

```
basic_istream<charT,traits>& seekg(pos_type pos);
Effects: If fail() != true, executes rdbuf()->pubseekpos(pos, ios_base::in).
```

In section 27.6.1.3 change:

```
basic_istream<charT,traits>& seekg(off_type& off, ios_base::seekdir dir);
Effects: If fail() != true, executes rdbuf()->pubseekoff(off, dir).
```

To:

```
basic_istream<charT,traits>& seekg(off_type& off, ios_base::seekdir dir);
Effects: If fail() != true, executes rdbuf()->pubseekoff(off, dir, ios_base::in).
```

In section 27.6.2.4, paragraph 2 change:

```
-2- Effects: If fail() != true, executes rdbuf()->pubseekpos(pos).
```

To:

```
-2- Effects: If fail() != true, executes rdbuf()->pubseekpos(pos, ios_base::out).
```

In section 27.6.2.4, paragraph 4 change:

```
-4- Effects: If fail() != true, executes rdbuf()->pubseekoff(off, dir).
```

To:

```
-4- Effects: If fail() != true, executes rdbuf()->pubseekoff(off, dir, ios_base::
```

---

137. Do use_facet and has_facet look in the global locale?

Section: 22.1.1 lib.locale **Status:** Open **Submitter:** Angelika Langer **Date:** March 17, 1999

Section 22.1.1 lib.locale says:

-4- In the call to use_facet<Facet>(loc), the type argument chooses a facet, making available all members of the named type. If Facet is not present in a locale (or, failing that, in the global locale), it throws the standard exception bad_cast. A C++ program can check if a locale implements a particular facet with the template function has_facet<Facet>().

This contradicts the specification given in section 22.1.2 lib.locale.global.templates :

template <class Facet> const Facet& use_facet(const locale& loc);

-1- Get a reference to a facet of a locale.
-2- Returns: a reference to the corresponding facet of loc, if present.
-3- Throws: bad_cast if has_facet<Facet>(loc) is false.
-4- Notes: The reference returned remains valid at least as long as any copy of loc exists

Proposed Resolution:

If there's consensus that section 22.1.2 reflects the intent, then the phrase:

(or, failing that, in the global locale)

should be removed from section 22.1.1.

---

138. Class ctype_byname<char> redundant and misleading

Section: 22.2.1.4 lib.locale.ctype.byname.special **Status:** Open **Submitter:** Angelika Langer **Date:** March 18, 1999

Section 22.2.1.4 lib.locale.ctype.byname.special specifies that ctype_byname<char> must be a specialization of the ctype_byname template.

It is common practice in the standard that specializations of class templates are only mentioned where the interface of the specialization deviates from the interface of the template that it is a specialization of. Otherwise, the fact whether or not a required instantiation is an actual instantiation or a specialization is left open as an implementation detail.

Clause 22.2.1.4 deviates from that practice and for that reason is misleading. The fact, that ctype_byname<char> is specified as a specialization suggests that there must be something "special" about it, but it has the exact same interface as the ctype_byname template. Clause 22.2.1.4 does not have any explanatory value, is at best redundant, at worst misleading - unless I am missing anything.

Naturally, an implementation will most likely implement ctype_byname<char> as a specialization, because the base class ctype<char> is a specialization with an interface different from the ctype template, but that's an implementation detail and need not be mentioned in the standard.

Proposed Resolution:

Delete section 22.2.1.4 lib.locale.ctype.byname.special

---

139. Optional sequence operation table description unclear

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

The sentence introducting 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 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 -----