

Doc. no. J16/00-0038
WG21 N1261
Date: 08 Sep 2000
Project: Programming Language C++
Reply to: Matt Austern <austern@research.att.com>

C++ Standard Library Defect Report List (Revision 15)

Reference ISO/IEC IS 14882:1998(E)

Also see:

- [Table of Contents](#) for all library issues.
- [Index by Section](#) for all library issues.
- [Index by Status](#) for all library issues.
- [Library Active Issues List](#)
- [Library Closed Issues List](#)

This document contains only library issues which have been closed by the Library Working Group (LWG) after being found to be defects in the standard. That is, issues which have a status of **DR**, **TC**, or **RR**. See "[C++ Standard Library Closed Issues List](#)" for issues closed as non-defects. See the "[C++ Standard Library Active Issues List](#)" for active issues and more information. The introductory material in that document also applies to this document.

Revision History

- R15: pre-Toronto mailing. Added notes to issues 2, 23, and 160 containing committee members' suggestions that those issues be reopened.
- R14: post-Tokyo II mailing; reflects committee actions taken in Tokyo. (00-0021R1/N1244)
- R13 Unchanged from R12.
- R12 added "and paragraph 5" to the proposed resolution of issue [29](#).
- R11 added potential defects from Kona (99-0044/N1220), changed the proposed resolution of issue [4](#) to NAD, and changed the wording of proposed resolution of issue [38](#).

Defect Reports

1. C library linkage editing oversight

Section: 17.4.2.2 [lib.using.linkage](#) **Status:** **DR** **Submitter:** Beman Dawes **Date:** 16 Nov 97

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

Proposed Resolution:

Change [lib.using.linkage](#) paragraph 2 from:

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

to:

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

2. Auto_ptr conversions effects incorrect

Section: 20.4.5.3 [lib.auto_ptr.conv](#) **Status:** [DR](#) **Submitter:** Nathan Myers **Date:** 4 Dec 97

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

Lois suggests that this issue be reopened. Further discussion:

A committee member [*Lois Goldthwaite, 01 Aug 2000*] observed in email:

...but it appears to me that the change to "p.release()" has already been made in the standard. Is it possible this was picked up as a late editorial correction?

Proposed Resolution:

Change [lib.auto_ptr.conv](#) paragraph 1 Effects from "Calls p->release()" to "Calls p.release()".

4. Basic_string size_type and difference_type should be implementation defined

Section: 21.3 [lib.basic.string](#) **Status:** [DR](#) **Submitter:** Beman Dawes **Date:** 16 Nov 97

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

Proposed Resolution:

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

Rationale:

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

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

5. String::compare specification questionable

Section: 21.3.6.8 [lib.string::compare](#) **Status:** [DR](#) **Submitter:** Jack Reeves **Date:** 11 Dec 97

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

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

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

This would imply that what was really intended was two signatures `int compare(size_type pos1, size_type n1, const charT* s)` and `int compare(size_type pos1, size_type n1, const charT* s, size_type n2)` const; each defined in terms of the corresponding constructor.

Proposed Resolution:

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

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

with:

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

Replace the portion of 21.3.6.8 [lib.string::compare](#) paragraphs 5 and 6 which read:

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

with:

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

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

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

Rationale:

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

7. String clause minor problems

Section: 21 [lib.strings](#) **Status:** [DR](#) **Submitter:** Matt Austern **Date:** 15 Dec 97

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

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

(3) `basic_string::push_back` appears in the synopsis, but is never described elsewhere. In the synopsis its argument is `const charT`, which doesn't make much sense; it should probably be `charT`, or possibly `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` is in `[p, p+n)`. These are two orthogonal guarantees, and neither one follows from the other. Both guarantees are necessary if `X::move` is supposed to have the same sort of semantics as `memmove` (which was clearly the intent), and both guarantees are necessary if `X::move` is actually supposed to be useful.

Proposed Resolution:

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

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

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

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

`void push_back(const charT)`

to

`void push_back(charT)`

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

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

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

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

ITEM 6: In table 37, Replace:

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

with:

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

11. Bitset minor problems

Section: 23.3.5 [lib.template.bitset](#) **Status:** [DR](#) **Submitter:** Matt Austern **Date:** 22 Jan 98

- (1) `bitset<>::operator[]` is mentioned in the class synopsis (23.3.5), but it is not documented in 23.3.5.2.
- (2) The class synopsis only gives a single signature for `bitset<>::operator[]`, `reference operator[](size_t pos)`. This doesn't make much sense. It ought to be overloaded on `const`. `reference operator[](size_t pos); bool operator[](size_t pos) const.`
- (3) Bitset's stream input function (23.3.5.3) ought to skip all whitespace before trying to extract 0s and 1s. The standard doesn't explicitly say that, though. This should go in the Effects clause.

Rationale:

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

Proposed Resolution:

ITEMS 1 AND 2:

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

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

with the two member functions

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

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

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

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

13. Eos refuses to die

Section: 27.6.1.2.3 [lib.istream::extractors](#) **Status:** [DR](#) **Submitter:** William M. Miller **Date:** 3 Mar 98

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

Proposed Resolution:

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

14. Locale::combine should be const

Section: 22.1.1.3 [lib.locale.members](#) **Status:** [DR](#) **Submitter:** Nathan Myers **Date:** 6 Aug 98

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

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

Proposed Resolution:

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

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

15. Locale::name requirement inconsistent

Section: 22.1.1.3 [lib.locale.members](#) **Status:** [DR](#) **Submitter:** Nathan Myers **Date:** 6 Aug 98

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

Proposed Resolution:

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

16. Bad ctype_byname<char> decl

Section: 22.2.1.4 [lib.locale.ctype.byname.special](#) **Status:** [DR](#) **Submitter:** Nathan Myers **Date:** 6 Aug 98

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

Proposed Resolution:

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

17. Bad bool parsing

Section: 22.2.2.1.2 [lib.facet.num.get.virtuals](#) **Status:** [DR](#) **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 `numunct<>` 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](#)].

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.categories](#) **Status:** [DR](#) **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.categories](#)] 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.istream.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.

Dietmar suggests that this issue be reopened. Further discussion:

I don't feel comfortable with the proposed resolution to issue 23: It kind of simplifies the issue too much. Here is what is going on:

Currently, the behavior of numeric overflow is rather counter intuitive and hard to trace, so I will describe it briefly:

- According to 22.2.2.1.2 ([lib.facet.num.get.virtuals](#)) paragraph 11 `failbit` is set if `scanf()` would return an input error; otherwise a value is converted to the rules of `scanf()`.
- `scanf()` is defined in terms of `fscanf()`.
- `fscanf()` returns an input failure if during conversion no character matching the conversion specification could be extracted before reaching EOF. This is the only reason for `fscanf()` to fail due to an input error and clearly does not apply to the case of overflow.
- Thus, the conversion is performed according to the rules of `fscanf()` which basically says that `strtod`, `strtol()`, etc. are to be used for the conversion.
- The `strtod()`, `strtol()`, etc. functions consume as many matching characters as there are and on overflow continue to consume matching characters but also return a value identical to the maximum (or minimum for signed types if there was a leading minus) value of the corresponding type and set `errno` to `ERANGE`.
- Thus, according to the current wording in the standard, overflows can be detected! All what is to be done is to check `errno` after reading an element and, of course, clearing `errno` before trying a conversion. With the current wording, it can be detected whether the overflow was due to a positive or negative number for signed types.

Now the proposed resolution results in not modifying the value passed as last argument if an overflow is encountered but `failbit` is set. Checking `errno` for `ERANGE` still allows for detection of an overflow but not what the sign was.

Actually, my problem is not that much with the sign but this is at least making things worse... My problem is more that it is still necessary to check `errno` for the error description. Thus, I propose the following resolution:

Change paragraph 11 from

-11- Stage 3: The result of stage 2 processing can be one of

- A sequence of `chars` has been accumulated in stage 2 that is converted (according to the rules of `scanf()`) to a value of the type of `val`. This value is stored in `val` and `ios_base::goodbit` is stored in `err`.

- The sequence of chars accumulated in stage 2 would have caused scanf to report an input failure. `ios_base::failbit` is assigned to `err`.

to become

-11- Stage 3: The result of stage 2 processing can be one of

- A sequence of chars has been accumulated in stage 2 that is converted (according to the rules of `scanf`) to a value of the type of `val`. This value is stored in `val`. If the conversion reported an overflow error for the type of `val` (ie. `errno` would be set to `ERANGE` by the used conversion function) then `ios_base::failbit` is stored in `err`, otherwise `ios_base::goodbit` is stored in `err`.
- The sequence of chars accumulated in stage 2 would have caused scanf to report an input failure. `ios_base::failbit` is assigned to `err`.

With this definition, overflow can be detected easily by storing a value different from the maximum value in `val` and checking whether this value was modified in case `failbit` is set: If it was, there was an overflow error, otherwise some other input error occurred (under the conditions for the second bullet `val` is not changed).

Proposed Resolution:

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

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

to

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

24. "do_convert" doesn't exist

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

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

Proposed Resolution:

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

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

Section: 21.3.7.9 [lib.string.io](#) **Status:** [DR](#) **Submitter:** Nathan Myers **Date:** 6 Aug 98

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

Proposed Resolution:

Change 21.3.7.9 [lib.string.io](#) paragraph 4 from:

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

to:

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

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

Section: 21.3.5.5 [lib.string::erase](#) **Status:** [DR](#) **Submitter:** Nathan Myers **Date:** 6 Aug 98

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

Proposed Resolution:

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

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

to read

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

28. `Ctype<char>::is` ambiguous

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

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

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

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

Proposed Resolution:

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

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

29. `Ios_base::init` doesn't exist

Section: 27.3.1 [lib.narrow.stream.objects](#) **Status:** [DR](#) **Submitter:** Nathan Myers **Date:** 6 Aug 98

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

Proposed Resolution:

[R12: modified to include paragraph 5.]

In 27.3.1 [[lib.narrow.stream.objects](#)] paragraph 2 and 5, change

```
ios_base::init
```

to

```
basic_ios<char>::init
```

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

```
basic_ios<wchar_t>::init
```

30. Wrong header for LC_*

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

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

Proposed Resolution:

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

32. Pbackfail description inconsistent

Section: 27.5.2.4.4 [[lib.streambuf.virt.pback](#)] **Status:** [DR](#) **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(*gp_ptr(),traits::to_char_type( c))"
```

to

```
"traits::eq(traits::to_char_type(c),gp_ptr()[-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.codecvrt.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.codecvrt.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 `truenamename()` 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.truenamename() : 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 synopsis. Similarly for `"nunitbuf"`.

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& nunitbuf(ios_base& str);
```

36. Iword & pword storage lifetime omitted

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

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

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

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

Proposed Resolution:

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

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

with:

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

substituting "iword" or "pword" as appropriate.

37. Leftover "global" reference

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

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

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

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

Proposed Resolution:

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

(or, failing that, in the global locale)

38. Facet definition incomplete

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

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

Proposed Resolution:

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

Get a reference to a facet of a locale.

with:

Requires: `Facet` is a facet class whose definition contains the public static member `id` as defined in (22.1.1.1.2, [[lib.locale.facet](#)]).

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

39. `istreambuf_iterator<>::operator++(int)` definition garbled

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

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

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

Proposed Resolution:

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

40. Meaningless normative paragraph in examples

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

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

Proposed Resolution:

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

41. `ios_base` needs `clear()`, `exceptions()`

Section: 27.4.2 [[lib.ios.base](#)] **Status:** [DR](#) **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 sub-object of a `basic_ios<>` object or sub-object, the effect is equivalent to calling `basic_ios<>::setstate(badbit)` on the derived object (which may throw failure).

[Kona: LWG reviewed wording; `setstate(failbit)` changed to `setstate(badbid)`.]

42. String ctors specify wrong default allocator

Section: 21.3 [[lib.basic.string](#)] **Status:** [DR](#) **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 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);
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()`.

Rationale:

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

The LWG considered two other possible resolutions:

A. 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()`.

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);
```

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

[Kona: issue editing snafu fixed - the proposed resolution now correctly reflects the LWG consensus.]

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_istream<char> {
  public:
    // Types
    typedef char                               char_type;
    typedef typename char_traits<char>::int_type int_type
    typedef typename char_traits<char>::pos_type pos_type;
```

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

Section: 27.4.2.3 [lib.ios.base.locales](#) **Status:** [DR](#) **Submitter:** Matt Austern **Date:** 21 Jun 98

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

Proposed Resolution:

In 27.4.2.3 [lib.ios.base.locales](#) swap paragraphs 2 and 4.

48. Use of non-existent exception constructor

Section: 27.4.2.1.1 [lib.ios::failure](#) **Status:** [DR](#) **Submitter:** Matt Austern **Date:** 21 Jun 98

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

Proposed Resolution:

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

EFFECTS: Constructs an object of class `failure`.

50. Copy constructor and assignment operator of ios_base

Section: 27.4.2 [lib.ios.base](#) **Status:** [DR](#) **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 assignment operators unless you take specific steps to avoid them.) However, nothing 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 [*Jerry Schwarz*] 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). *So I guess [remainder of comment disappeared into the black hole of email].*

Proposed Resolution:

In 27.4.2 [lib.ios.base](#), class ios_base, specify the copy constructor and operator= members as being private.

Rationale:

The LWG believes the difficulty of specifying correct semantics outweighs any benefit of allowing ios_base objects to be copyable.

51. Requirement to not invalidate iterators missing

Section: 23.1 [lib.container.requirements](#) **Status:** [DR](#) **Submitter:** David Vandevoorde **Date:** 23 Jun 98

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

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

A committee member [*Matt Austern, lib-6528*] comments:

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

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

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

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

The issues list maintainer [*Beman Dawes*] comments:

This was US issue CD2-23-011; it was accepted in London *but the change was not made due to an editing oversight*. The wording in the proposed resolution below is somewhat updated from CD2-23-011, particularly the addition of the phrase "or change the values of"

Proposed Resolution:

Add a new paragraph at the end of 23.1:

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

52. Small I/O problems

Section: 27.4.3.2 [lib.fpos.operations](#) **Status:** [DR](#) **Submitter:** Matt Austern **Date:** 23 Jun 98

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

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

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

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

Proposed Resolution:

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

53. Basic_ios destructor unspecified

Section: 27.4.4.1 [lib.basic.ios.cons](#), 27.4.4.2 [lib.basic.ios.members](#) **Status:** [DR](#) **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().

Rationale:

The LWG reviewed the additional question of whether or not rdbuf(0) may set badbit. The answer is clearly yes; it may be set via clear(). See 27.4.4.2 [lib.basic.ios.members](#), paragraph 6.

[Kona: reviewed at length by the LWG, which removed from the original proposed resolution a footnote which incorrectly said "rdbuf(0) does not set badbit".]

54. Basic_streambuf's destructor

Section: 27.5.2.1 [lib.streambuf.cons](#) **Status:** [DR](#) **Submitter:** Matt Austern **Date:** 25 Jun 98

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

Proposed Resolution:

Add after 27.5.2.1 [lib.streambuf.cons](#) paragraph 2:

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

Effects: None.

55. Invalid stream position is undefined

Section: 27 [lib.input.output](#) **Status:** [DR](#) **Submitter:** Matt Austern **Date:** 26 Jun 98

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

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

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

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

Proposed Resolution:

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

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

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

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

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

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

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

56. Showmanyc's return type

Section: 27.5.2 [lib.streambuf](#) **Status:** [DR](#) **Submitter:** Matt Austern **Date:** 29 Jun 98

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

Proposed Resolution:

Change `showmanyc`'s return type in the 27.5.2 [lib.streambuf](#) class summary to `streamsize`.

57. Mistake in char_traits

Section: 21.1.3.2 [lib.char_traits.specializations.wchar_t](#) **Status:** DR **Submitter:** Matt Austern **Date:** 1 Jul 98

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

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

Proposed Resolution:

Remove the sentence in 21.1.3.2 [lib.char_traits.specializations.wchar_t](#) paragraph 3 which begins "The types streampos and wstreampos may be different..." .

59. Ambiguity in specification of gbump

Section: 27.5.2.3.1 [lib.streambuf.get.area](#) **Status:** DR **Submitter:** Matt Austern **Date:** 28 Jul 98

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

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

[A LWG member reports that the AT&T implementation used the former interpretation.]

Proposed Resolution:

Change 27.5.2.3.1 [lib.streambuf.get.area](#) paragraph 4 gbump effects from:

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

to:

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

Make the same change to 27.5.2.3.2 [lib.streambuf.put.area](#) paragraph 4 pbump effects.

60. What is a formatted input function?

Section: 27.6.1.2.1 [lib.istream.formatted.reqmts](#) **Status:** DR **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_streambuf*);
```

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

Comments from Judy Ward: 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.

Additional comments from Dietmar Kühl: It appears to be somewhat nonsensical to consider the functions defined in 27.6.1.2.3 [lib.istream::extractors](#) paragraphs 1 to 5 to be "Formatted input function" but since these functions are defined in a section labeled "Formatted input functions" it is unclear to me whether these operators are considered formatted input functions which have to conform to the "common requirements" from 27.6.1.2.1 ([lib.istream.formatted.reqmts](#)): If this is the case, all manipulators, not just `ws`, would skip whitespace unless `noskipws` is set (... but setting `noskipws` using the manipulator syntax would also skip whitespace :-)

It is not clear which functions are to be considered unformatted input functions. As written, it seems that all functions in 27.6.1.3 ([lib.istream.unformatted](#)) are unformatted input functions. However, it does not really make much sense to construct a sentry object for `gcount()`, `sync()`, ... Also it is unclear what happens to the `gcount()` if eg. `gcount()`, `putback()`, `unget()`, or `sync()` is called: These functions don't extract characters, some of them even "unextract" a character. Should this still be reflected in `gcount()`? Of course, it could be read as if after a call to `gcount()` `gcount()` return 0 (the last unformatted input function, `gcount()`, didn't extract any character) and after a call to `putback()` `gcount()` returns -1 (the last unformatted input function `putback()` did "extract" back into the stream). Correspondingly for `unget()`. Is this what is intended? If so, this should be clarified. Otherwise, a corresponding clarification should be used.

Proposed Resolution:

Change the standard as specified in J16/99-0043==WG21/N1219, Proposed Resolution to Library Issue 60, section "VI Wording", by Judy Ward and Matt Austern.

62. `sync`'s return value

Section: 27.6.1.3 [lib.istream.unformatted](#) **Status:** [DR](#) **Submitter:** Matt Austern **Date:** 6 Aug 98

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

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

Proposed Resolution:

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

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

Section: 27.6.1.2.3 [lib.istream::extractors](#) **Status:** [DR](#) **Submitter:** Matt Austern **Date:** 11 Aug 98

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

Proposed Resolution:

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

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

66. `Strstreambuf::setbuf`

Section: D.7.1.3 [depr.strstreambuf.virtuals](#) **Status:** [DR](#) **Submitter:** Matt Austern **Date:** 18 Aug 98

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

Proposed Resolution:

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

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

68. Extractors for `char*` should store null at end

Section: 27.6.1.2.3 [lib.istream::extractors](#) **Status:** [DR](#) **Submitter:** Angelika Langer **Date:** 14 Jul 1998

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

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

Proposed Resolution:

27.6.1.2.3 [lib.istream::extractors](#), paragraph 7, change the last list item from:

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

to become a new paragraph which reads:

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

69. Must elements of a vector be contiguous?

Section: 23.2.4 [lib.vector](#) **Status:** [DR](#) **Submitter:** Andrew Koenig **Date:** 29 Jul 1998

The issue is this:

Must the elements of a vector be in contiguous memory?

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

Proposed Resolution:

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

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

Rationale:

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

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

70. `uncaught_exception()` missing `throw()` specification

Section: 18.6 [lib.support.exception](#), 18.6.4 [lib.uncaught](#) **Status:** [DR](#) **Submitter:** Steve Clamage **Date:**

In article 3E04@pratique.fr, *Valentin Bonnard* writes:

`uncaught_exception()` doesn't have a `throw` specification.

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

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

Proposed Resolution:

In 18.6 [lib.support.exception](#) and 18.6.4 [lib.uncaught](#) add `throw()` to `uncaught_exception()`.

71. `Do_get_monthname` synopsis missing argument

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

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

Proposed Resolution:

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

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

74. Garbled text for `codecvt::do_max_length`

Section: 22.2.1.5.2 [lib.locale.codecvt.virtuals](#) **Status:** [DR](#) **Submitter:** Matt Austern **Date:** 18 Sep 1998

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

Proposed Resolution:

Replace 22.2.1.5.2 [lib.locale.codecvt.virtuals](#) paragraph 11 with the following:

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

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

Section: 22.2.1.5 [lib.locale.codecvt](#) **Status:** [DR](#) **Submitter:** Matt Austern **Date:** 18 Sep 98

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

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

Proposed Resolution:

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

```
const stateT&
```

to

```
stateT&
```

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

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

78. Typo: `event_call_back`

Section: 27.4.2 [lib.ios.base](#) **Status:** [DR](#) **Submitter:** Nico Josuttis **Date:** 29 Sep 98

typo: `event_call_back` should be `event_callback`

Proposed Resolution:

In the 27.4.2 [lib.ios.base](#) synopsis change "`event_call_back`" to "`event_callback`".

79. Inconsistent declaration of `polar()`

Section: 26.2.1 [lib.complex.synopsis](#), 26.2.7 [lib.complex.value.ops](#) **Status:** [DR](#) **Submitter:** Nico Josuttis **Date:** 29 Sep 1998

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 1998

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

Proposed Resolution:

Reduce redundancy according to the general style of the standard.

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

Section: 21.3 [lib.basic.string](#) **Status:** [DR](#) **Submitter:** Nico Josuttis **Date:** 29 Sep 1998

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:

After 21.3 [[lib.basic.string](#)] paragraph 4 ("The functions described in this clause...") add a new paragraph:

For any string operation, if as a result of the operation, `size()` would exceed `max_size()` then the operation throws `length_error`.

Rationale:

The LWG believes `length_error` is the correct exception to throw.

90. Incorrect description of operator `>>` for strings

Section: 21.3.7.9 [lib.string.io](#) **Status:** [DR](#) **Submitter:** Nico Josuttis **Date:** 29 Sep 1998

The effect of operator `>>` for strings contains the following item:

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

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

Proposed resolution:

In 21.3.7.9 [lib.string.io](#) paragraph 1 Effects clause replace:

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

with:

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

106. Numeric library private members are implementation defined

Section: 26.3.5 [lib.template.slice.array](#), etc. **Status:** [DR](#) **Submitter:** AFNOR **Date:** 7 Oct 98

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

Proposed Resolution:

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

- 26.3.5 [lib.template.slice.array](#)
- 26.3.7 [lib.template.gslice.array](#)
- 26.3.8 [lib.template.mask.array](#)
- 26.3.9 [lib.template.indirect.array](#)

110. istreambuf_iterator::equal not const

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

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

Proposed Resolution:

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

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

with:

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

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

Section: 17.4.4.8 [lib.res.on.exception.handling](#) **Status:** [DR](#) **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-specification for a function"

to:

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

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

Section: 22.2.1.2 [lib.locale.ctype.byame](#) **Status:** [DR](#) **Submitter:** Judy Ward **Date:** 15 Dec 1998

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

Proposed Resolution:

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

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

Section: 26.3.2 [lib.template.valarray](#) **Status:** [DR](#) **Submitter:** Judy Ward **Date:** 15 Dec 1998

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

Proposed Resolution:

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

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

Section: 22.2.1.1.2 [lib.locale.ctype.virtuals](#) **Status:** [DR](#) **Submitter:** Judy Ward **Date:** 15 Dec 1998

Proposed Resolution:

In Section 22.2.1.1.2 [lib.locale.ctype.virtuals](#) change:

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

to:

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

and change:

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

to:

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

132. list::resize description uses random access iterators

Section: 23.2.2.2 [lib.list.capacity](#) **Status:** [DR](#) **Submitter:** Howard Hinnant **Date:** 6 Mar 99

The description reads:

-1- Effects:

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

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

Proposed Resolution:

Change 23.2.2.2 paragraph 1 to:

Effects:

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

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

133. map missing get_allocator()

Section: 23.3.1 [lib.map](#) **Status:** [DR](#) **Submitter:** Howard Hinnant **Date:** 6 Mar 99

The title says it all.

Proposed Resolution:

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

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

139. Optional sequence operation table description unclear

Section: 23.1.1 [lib.sequence.reqmts](#) **Status:** [DR](#) **Submitter:** Andrew Koenig **Date:** 30 Mar 1999

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

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

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

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

Proposed Resolution:

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

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

141. `basic_string::find_last_of`, `find_last_not_of` say pos instead of xpos

Section: 21.3.6.4 [lib.string::find.last.of](#), 21.3.6.6 [lib.string::find.last.not.of](#) **Status:** [DR](#) **Submitter:** Arch Robison **Date:** 28 Apr 99

Sections 21.3.6.4 paragraph 1 and 21.3.6.6 paragraph 1 surely have misprints where they say:

```
— xpos <= pos and pos < size();
```

Surely the document meant to say ``xpos < size()" in both places.

[Judy Ward also sent in this issue for 21.3.6.4 with the same proposed resolution.]

Proposed Resolution:

Change Sections 21.3.6.4 paragraph 1 and 21.3.6.6 paragraph 1, the line which says:

```
— xpos <= pos and pos < size();
```

to:

```
— xpos <= pos and xpos < size();
```

148. Functions in the example facet `BoolNames` should be const

Section: 22.2.8 [lib.facets.examples](#) **Status:** [DR](#) **Submitter:** Jeremy Siek **Date:** 3 Jun 99

In 22.2.8 [[lib.facets.examples](#)] paragraph 13, the `do_truename()` and `do_falsename()` functions in the example facet `BoolNames` should be const. The functions they are overriding in `numpunct_byname<char>` are const.

Proposed Resolution:

In 22.2.8 [[lib.facets.examples](#)] paragraph 13, insert "const" in two places:

```
string do_truename() const { return "Oui Oui!"; }
string do_falsename() const { return "Mais Non!"; }
```

150. Find_first_of says integer instead of iterator

Section: 25.1.4 [lib.alg.find.first.of](#) **Status:** [DR](#) **Submitter:** Matt McClure **Date:** 30 Jun 99

Proposed Resolution:

Change 25.1.4 [lib.alg.find.first.of](#) paragraph 2 from:

Returns: The first iterator *i* in the range [first1, last1) such that for some integer *j* in the range [first2, last2) ...

to:

Returns: The first iterator *i* in the range [first1, last1) such that for some iterator *j* in the range [first2, last2) ...

151. Can't currently clear() empty container

Section: 23.1.1 [lib.sequence.reqmts](#) **Status:** [DR](#) **Submitter:** Ed Brey **Date:** 21 Jun 99

For both sequences and associative containers, `a.clear()` has the semantics of `erase(a.begin(),a.end())`, which is undefined for an empty container since `erase(q1,q2)` requires that `q1` be dereferenceable (23.1.1,3 and 23.1.2,7). When the container is empty, `a.begin()` is not dereferenceable.

The requirement that `q1` be unconditionally dereferenceable causes many operations to be intuitively undefined, of which clearing an empty container is probably the most dire.

Since `q1` and `q2` are only referenced in the range `[q1, q2)`, and `[q1, q2)` is required to be a valid range, stating that `q1` and `q2` must be iterators or certain kinds of iterators is unnecessary.

Proposed Resolution:

In 23.1.1, paragraph 3, change:

`p` and `q2` denote valid iterators to `a`, `q` and `q1` denote valid dereferenceable iterators to `a`, `[q1, q2)` denotes a valid range

to:

`p` denotes a valid iterator to `a`, `q` denotes a valid dereferenceable iterator to `a`, `[q1, q2)` denotes a valid range in
a

In 23.1.2, paragraph 7, change:

`p` and `q2` are valid iterators to `a`, `q` and `q1` are valid dereferenceable iterators to `a`, `[q1, q2)` is a valid range

to

`p` is a valid iterator to `a`, `q` is a valid dereferenceable iterator to `a`, `[q1, q2)` is a valid range into `a`

152. Typo in `scan_is()` semantics

Section: 22.2.1.1.2 [lib.locale ctype.virtuals](#) **Status:** DR **Submitter:** Dietmar Kühl **Date:** 20 Jul 99

The semantics of `scan_is()` (paragraphs 4 and 6) is not exactly described because there is no function `is()` which only takes a character as argument. Also, in the effects clause (paragraph 3), the semantic is also kept vague.

Proposed resolution:

In 22.2.1.1.2 [lib.locale ctype.virtuals](#) paragraphs 4 and 6, change the returns clause from:

"... such that `is(*p)` would..."

to: "... such that `is(m, *p)` would...."

154. Missing `double` specifier for `do_get()`

Section: 22.2.2.1.2 [lib.facet.num.get.virtuals](#) **Status:** DR **Submitter:** Dietmar Kühl **Date:** 20 Jul 99

The table in paragraph 7 for the length modifier does not list the length modifier `l` to be applied if the type is `double`. Thus, the standard asks the implementation to do undefined things when using `scanf()` (the missing length modifier for `scanf()` when scanning doubles is actually a problem I found quite often in production code, too).

Proposed resolution:

In 22.2.2.1.2 [lib.facet.num.get.virtuals](#), paragraph 7, add a row in the Length Modifier table to say that for `double` a length modifier `l` is to be used.

Rationale:

The standard makes an embarrassing beginner's mistake.

155. Typo in naming the class defining the class `Init`

Section: 27.3 [lib.iostream.objects](#) **Status:** DR **Submitter:** Dietmar Kühl **Date:** 20 Jul 99

There are conflicting statements about where the class `Init` is defined. According to 27.3 ([lib.iostream.objects](#)) paragraph 2 it is defined as `basic_ios::Init`, according to 27.4.2 ([lib.ios.base](#)) it is defined as `ios_base::Init`.

Proposed resolution:

Change 27.3 ([lib.iostream.objects](#)) paragraph 2 from "`basic_ios::Init`" to "`ios_base::Init`".

Rationale:

Although not strictly wrong, the standard was misleading enough to warrant the change.

156. Typo in `imbue()` description

Section: 27.4.2.3 [lib.ios.base.locales](#) **Status:** [DR](#) **Submitter:** Dietmar Kühl **Date:** 20 Jul 99

There is a small discrepancy between the declarations of `imbue()`: in 27.4.2 ([lib.ios.base](#)) the argument is passed as `locale const&` (correct), in 27.4.2.3 ([lib.ios.base.locales](#)) it is passed as `locale const` (wrong).

Proposed resolution:

In 27.4.2.3 ([lib.ios.base.locales](#)) change the `imbue` argument from "locale const" to "locale const&".

158. Underspecified semantics for `setbuf()`

Section: 27.5.2.4.2 [lib.streambuf.virt.buffer](#) **Status:** [DR](#) **Submitter:** Dietmar Kühl **Date:** 20 Jul 99

The default behavior of `setbuf()` is described only for the situation that `gptr() != 0 && gptr() != egptr()`: namely to do nothing. What has to be done in other situations is not described although there is actually only one reasonable approach, namely to do nothing, too.

Since changing the buffer would almost certainly mess up most buffer management of derived classes unless these classes do it themselves, the default behavior of `setbuf()` should always be to do nothing.

Proposed resolution:

Change 27.5.2.4.2 [lib.streambuf.virt.buffer](#), paragraph 3, Default behavior, to: "Default behavior: Does nothing. Returns this."

160. Typo: Use of non-existing function `exception()`

Section: 27.6.1.1 [lib.istream](#) **Status:** [DR](#) **Submitter:** Dietmar Kühl **Date:** 20 Jul 99

The paragraph 4 refers to the function `exception()` which is not defined. Probably, the referred function is `basic_ios::exceptions()`.

Martin suggests that this issue be reopened. Further discussion:

A committee member [*Martin Sebor, lib-7672, 30 Jun 2000*] comments:

This is an amendment to issue 160 -- Typo: Use of non-existing function `exception()`.

27.6.1.2.1, p1 says: "... If `(exception()&badbit)!= 0` then the exception is rethrown. In any case, the formatted input function destroys the sentry object. If no exception has been thrown, it returns `*this`."

The same typo (`exceptions()` spelled `exception()`) appears in 27.6.1.3, p1, 27.6.2.1, p3, 27.6.2.5.1, p1, and the proposed resolution to [issue 63](#).

The proposed resolution is to correct all these typos by adding the missing `s` as well as the `ios_base::` qualification (as suggested by the note in issue 160).

Note that in the last case ([issue 63](#)), the change may require that the issue be reopened (it's in a Ready state).

Proposed resolution:

In 27.6.1.1 [lib.istream](#) change "exception()" to "basic_ios::exceptions()".

[Note to Editor: in addition to adding "basic_ios::", "exceptions" with an "s" is the correct spelling.]

161. Typo: `istream_iterator` vs. `istreambuf_iterator`

Section:: 27.6.1.2.2 [lib.istream.formatted.arithmetic](#) **Status:** [DR](#) **Submitter:** Dietmar Kühl **Date:** 20 Jul 99

The note in the second paragraph pretends that the first argument is an object of type `istream_iterator`. This is wrong: It is an object of type `istreambuf_iterator`.

Proposed resolution:

Change 27.6.1.2.2 [lib.istream.formatted.arithmetic](#) from:

The first argument provides an object of the `istream_iterator` class...

to

The first argument provides an object of the `istreambuf_iterator` class...

168. Typo: `formatted` vs. `unformatted`

Section:: 27.6.2.6 [lib ostream.unformatted](#) **Status:** [DR](#) **Submitter:** Dietmar Kühl **Date:** 20 Jul 99

The first paragraph begins with a descriptions what has to be done in `*formatted*` output functions. Probably this is a typo and the paragraph really want to describe `unformatted` output functions...

Proposed resolution:

In 27.6.2.6 [lib ostream.unformatted](#) paragraph 1, the first and last sentences, change the word "formatted" to "unfomatted":

"Each **unformatted** output function begins ..."
"... value specified for the **unformatted** output function."

169. Bad efficiency of `overflow()` mandated

Section:: 27.7.1.3 [lib.stringbuf.virtuals](#) **Status:** [DR](#) **Submitter:** Dietmar Kühl **Date:** 20 Jul 99

Paragraph 8, Notes, of this section seems to mandate an extremely inefficient way of buffer handling for `basic_stringbuf`,

especially in view of the restriction that `basic_ostream` member functions are not allowed to use `xspn()` (see 27.6.2.1 [lib.ostream](#)): For each character to be inserted, a new buffer is to be created.

Of course, the resolution below requires some handling of simultaneous input and output since it is no longer possible to update `egptr()` whenever `eptr()` is changed. A possible solution is to handle this in `underflow()`.

Proposed resolution:

In 27.7.1.3 [lib.stringbuf.virtuals](#) paragraph 8, Notes, insert the words "at least" as in the following:

To make a write position available, the function reallocates (or initially allocates) an array object with a sufficient number of elements to hold the current array object (if any), plus **at least** one additional write position.

172. Inconsistent types for `basic_istream::ignore()`

Section:: 27.6.1.3 [lib.istream.unformatted](#) **Status:** [DR](#) **Submitter:** Greg Comeau, Dietmar Kühl **Date:** 23 Jul 99

In 27.6.1.1 ([lib.istream](#)) the function `ignore()` gets an object of type `streamsize` as first argument. However, in 27.6.1.3 ([lib.istream.unformatted](#)) paragraph 23 the first argument is of type `int`.

As far as I can see this is not really a contradiction because everything is consistent if `streamsize` is typedef to be `int`. However, this is almost certainly not what was intended. The same thing happened to [basic_filebuf::setbuf\(\)](#).

Darin Adler also submitted this issue, commenting: Either 27.6.1.1 should be modified to show a first parameter of type `int`, or 27.6.1.3 should be modified to show a first parameter of type `streamsize` and use `numeric_limits<streamsize>::max`.

Proposed resolution:

In 27.6.1.3 ([lib.istream.unformatted](#)) paragraph 23 and 24, change both uses of `int` in the description of `ignore()` to `streamsize`.

173. Inconsistent types for `basic_filebuf::setbuf()`

Section:: 27.8.1.4 [lib.filebuf.virtuals](#) **Status:** [DR](#) **Submitter:** Greg Comeau, Dietmar Kühl **Date:** 23 Jul 99

In 27.8.1.1 ([lib.istream](#)) the function `setbuf()` gets an object of type `streamsize` as second argument. However, in 27.8.1.4 ([lib.istream.unformatted](#)) paragraph 9 the second argument is of type `int`

As far as I can see this is not really a contradiction because everything is consistent if `streamsize` is typedef to be `int`. However, this is almost certainly not what was intended. The same thing happened to [basic_istream::ignore\(\)](#).

Proposed resolution:

In 27.8.1.4 ([lib.istream.unformatted](#)) paragraph 9, change all uses of `int` in the description of `setbuf()` to `streamsize`.

174. Typo: `OFF_T` vs. `POS_T`

Section: D.6 [depr.ios.members](#) Status: [DR](#) Submitter: Dietmar Kühl Date: 23 Jul 99

According to paragraph 1 of this section, `streampos` is the type `OFF_T`, the same type as `streamoff`. However, in paragraph 6 the `streampos` gets the type `POS_T`

Proposed resolution:

Change D.6 [depr.ios.members](#) paragraph 1 from `"typedef OFF_T streampos;"` to `"typedef POS_T streampos;"`

175. Ambiguity for `basic_streambuf::pubseekpos()` and a few other functions.

Section: D.6 [depr.ios.members](#) Status: [DR](#) Submitter: Dietmar Kühl Date: 23 Jul 99

According to paragraph 8 of this section, the methods `basic_streambuf::pubseekpos()`, `basic_ifstream::open()`, and `basic_ofstream::open` "may" be overloaded by a version of this function taking the type `ios_base::open_mode` as last argument argument instead of `ios_base::openmode` (`ios_base::open_mode` is defined in this section to be an alias for one of the integral types). The clause specifies, that the last argument has a default argument in three cases. However, this generates an ambiguity with the overloaded version because now the arguments are absolutely identical if the last argument is not specified.

Proposed resolution:

In D.6 [depr.ios.members](#) paragraph 8, remove the default arguments for `basic_streambuf::pubseekpos()`, `basic_ifstream::open()`, and `basic_ofstream::open()`.

176. `exceptions()` in `ios_base`...?

Section: D.6 [depr.ios.members](#) Status: [DR](#) Submitter: Dietmar Kühl Date: 23 Jul 99

The "overload" for the function `exceptions()` in paragraph 8 gives the impression that there is another function of this function defined in class `ios_base`. However, this is not the case. Thus, it is hard to tell how the semantics (paragraph 9) can be implemented: "Call the corresponding member function specified in clause [lib.input.output](#)."

Proposed resolution:

In D.6 [depr.ios.members](#) paragraph 8, move the declaration of the function `exceptions()` into class `basic_ios`.

187. `iter_swap` underspecified

Section: 25.2.2 [lib.alg.swap](#) Status: [DR](#) Submitter: Andrew Koenig Date: 14 Aug 99

The description of `iter_swap` in 25.2.2 paragraph 7, says that it "exchanges the values" of the objects to which two iterators refer.

What it doesn't say is whether it does so using `swap` or using the assignment operator and copy constructor.

This question is an important one to answer, because `swap` is specialized to work efficiently for standard containers. For example:

```
vector<int> v1, v2;
iter_swap(&v1, &v2);
```

Is this call to `iter_swap` equivalent to calling `swap(v1, v2)`? Or is it equivalent to

```
{
vector<int> temp = v1;
v1 = v2;
v2 = temp;
}
```

The first alternative is $O(1)$; the second is $O(n)$.

A LWG member, Dave Abrahams, comments:

Not an objection necessarily, but I want to point out the cost of that requirement:

```
iter_swap(list<T>::iterator, list<T>::iterator)
```

can currently be specialized to be more efficient than `iter_swap(T*, T*)` for many `T` (by using splicing). Your proposal would make that optimization illegal.

[Kona: The LWG notes the original need for `iter_swap` was proxy iterators which are no longer permitted.]

Proposed resolution:

Change the effect clause of `iter_swap` in 25.2.2 paragraph 7 from:

Exchanges the values pointed to by the two iterators `a` and `b`.

to

```
swap(*a, *b).
```

189. `setprecision()` not specified correctly

Section: 27.4.2.2 [lib.fmtflags.state](#) **Status:** [DR](#) **Submitter:** Andrew Koenig **Date:** 25 Aug 99

27.4.2.2 paragraph 9 claims that `setprecision()` sets the precision, and includes a parenthetical note saying that it is the number of digits after the decimal point.

This claim is not strictly correct. For example, in the default floating-point output format, `setprecision` sets the number of significant digits printed, not the number of digits after the decimal point.

I would like the committee to look at the definition carefully and correct the statement in 27.4.2.2

Proposed resolution:

Remove from 27.4.2.2 [lib.fmtflags.state](#), paragraph 9, the text "(number of digits after the decimal point)".

193. Heap operations description incorrect

Section: 25.3.6 [lib.alg.heap.operations](#) **Status:** [DR](#) **Submitter:** Markus Mauhart **Date:** 24 Sep 99

25.3.6 [lib.alg.heap.operations] states two key properties of a heap [a,b), the first of them is

“(1) *a is the largest element”

I think this is incorrect and should be changed to the wording in the proposed resolution.

Actually there are two independent changes:

A-"part of largest equivalence class" instead of "largest", cause 25.3 [lib.alg.sorting] asserts "strict weak ordering" for all its sub clauses.

B-Take 'an oldest' from that equivalence class, otherwise the heap functions could not be used for a priority queue as explained in 23.2.3.2.2 [lib.priqueue.members] (where I assume that a "priority queue" respects priority AND time).

Proposed Resolution:

Change 25.3.6 [[lib.alg.heap.operations](#)] property (1) from:

(1) *a is the largest element

to:

(1) There is no element greater than *a

195. Should `basic_istream::sentry`'s constructor ever set eofbit?

Section: 27.6.1.1.2 [lib.istream::sentry](#) **Status:** [DR](#) **Submitter:** Matt Austern **Date:** 13 Oct 99

Suppose that `is.flags() & ios_base::skipws` is nonzero. What should `basic_istream<>::sentry`'s constructor do if it reaches eof while skipping whitespace? 27.6.1.1.2/5 suggests it should set failbit. Should it set eofbit as well? The standard doesn't seem to answer that question.

On the one hand, nothing in 27.6.1.1.2 [lib.istream::sentry] says that `basic_istream<>::sentry` should ever set eofbit. On the other hand, 27.6.1.1/4 [lib.istream] says that if extraction from a `streambuf` "returns `traits::eof()`", then the input function, except as explicitly noted otherwise, completes its actions and does `setstate(eofbit)`". So the question comes down to whether `basic_istream<>::sentry`'s constructor is an input function.

Comments from Jerry Schwarz:

It was always my intention that eofbit should be set any time that a virtual returned something to indicate eof, no matter what reason istream code had for calling the virtual.

The motivation for this is that I did not want to require streambufs to behave consistently if their virtuals are called after they have signalled eof.

The classic case is a streambuf reading from a UNIX file. EOF isn't really a state for UNIX file descriptors.

The convention is that a read on UNIX returns 0 bytes to indicate "EOF", but the file descriptor isn't shut down in any way and future reads do not necessarily also return 0 bytes. In particular, you can read from tty's on UNIX even after they have signalled "EOF". (It isn't always understood that a ^D on UNIX is not an EOF indicator, but an EOL indicator. By typing a "line" consisting solely of ^D you cause a read to return 0 bytes, and by convention this is interpreted as end of file.)

Proposed Resolution:

Add a sentence to the end of 27.6.1.1.2 paragraph 2:

If `is.rdbuf()->sbumpc()` or `is.rdbuf()->sgetc()` returns `traits::eof()`, the function calls `setstate(failbit | eofbit)` (which may throw `ios_base::failure`).

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