Project: ISO C++
Date: 1998-8-13
Doc. No.: J16/98-0017 = WG21/N1160
Author: Nathan Myers
<ncm@nospam.cantrip.org

Nathan's Issues for ISO 14882 Library Clauses   DRAFT 98-08-13
by Nathan C. Myers ncm@nospam.cantrip.org
URL: http://www.cantrip.org/draft-bugs.txt

---

1.   22.1.1.3 [lib.locale.members] locale::combine should be const

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;
```

---

2. 22.1.1.3 [lib.locale.members] locale::name requirement inconsistent

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())".

---

3. 22.2.1.4, [lib.locale.ctype.byname.special] bad ctype_byname<char> decl

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].

---

4. 22.2.2.1.2 [lib.facet.num.get.virtuals] bad bool parsing

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. Finally, it doesn't define the value "loc".

In other words, the description is full of errors, and if the obvious errors are corrected, the result is unusable.

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();
const string_type f = np.falsename();
bool tm = true; bool 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) { err |= str.failbit; return in; }
}
val = tm;
return in;
```

Notice this works when the candidate strings are both empty, and 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], 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< numpunct<charT> >(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. If a target sequence is uniquely matched, _val_ is set to the corresponding value; or if the targets are identical and matched, _val_ is set to _true_.

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 _val==true_ and _err==str.goodbit_. --end example]

Also: in the first line of paragraph 14, change "&&" to "&".

----
5. 22.2.2.1.1 [lib.facet.num.get.members] get(...bool&) omitted

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].

----
6. 22.2.1.5.2 [lib.locale.codecvt.virtuals] "noconv" definition too vague.

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.

----
7. 22.2.3.1.2 [lib.facet.numpunct.virtuals] thousands_sep returns wrong type

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".

----
8. 22.1.1.1.1 [lib.locale.category] codecvt_byname<> instantiations

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>

----
9. 27.8.1.7 [lib.ifstream.members] member open vs. flags

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], _and_ in 27.8.1.10 [lib.ofstream.members], under open(), one of

A. no change
B. Prepend to Effects: "If fail(), returns. Otherwise"...
C. Prepend to Effects: "Call clear(); then," ...

---

10. 27.8.1.10 [lib.ofstream.members] member open vs. flags

(same as issue 9, respective.)

---

11. 22.2.2.1.2 [lib.facet.num.get.virtuals] num_get overflow result

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_.

---

12. 22.2.1.5.2 [lib.locale.codecvt.virtuals] "do_convert" doesn't exist

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".

---

13. 21.3.7.9 [lib.string.io] string op>> uses width() value wrong.

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:
In 21.3.7.9 [lib.string.io], paragraph 3, change the word "smaller"

to "larger".

```
-----------------------------------------------------------------------------
```

14. 27.6.1.1.2 [lib.istream::sentry] Bad sentry example

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 = char_traits<charT> >
  basic_istream<charT,traits>::sentry(
           basic_istream<charT,traits>& is, bool noskipws = false) {
      ...
      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(ctype.space,c);
           c = is.rdbuf()->snextc())
        {}
      ...
   }
```

```
-----------------------------------------------------------------------------
```

15. 21.3.5.5 [lib.string::erase] string::erase(range) yields wrong iterator

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.

```
-----------------------------------------------------------------------------
```

16. 22.2.1.3.2 [lib.facet.ctype.char.members] ctype<char>is ambiguous

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].

---

17. 27.3.1 [lib.narrow.stream.objects] ios_base::init doesn't exist

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

---

18. 22.1.1.1.1 [lib.locale.category] wrong header for LC_*

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>".

---

19. 22.1.1 [lib.locale] immutable locale values

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] paragraph 6, replace the text

"An instance of _locale_ is *immutable*;"

with

"A _locale_ value is *immutable*;"

---

20. 27.5.2.4.4 [lib.streambuf.virt.pback] pbackfail description inconsistent

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 "*gptr()" to read instead "gptr()[-1]".

---

21. 22.2.1.5.2 [lib.locale.codecvt.virtuals] codecvt<> mentions from_type

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

  enountered a character in [from,from_end) that it could not convert.

---

22. 22.2.2.2.2 [lib.facet.num.get.virtuals] true/falsename() not in ctype<>.

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.put.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();
```

---

23. 27.4 [lib.iostreams.base] No manipulator unitbuf in synopsis

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);
```

---

24. 27.4.2.5 [lib.ios.base.storage] iword & pword storage lifetime omitted

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.

25. 22.1.1 [lib.locale] leftover "global" reference

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)

----

26. 22.1.2 [lib.locale.global.templates] Facet definition incomplete.

Esa Pulkkinen has 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]).

----

27. 24.5.3.4 [lib.istreambuf.iterator::op++] sbufiter ++ definition garbled

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.

---

28. 22.2.8 [lib.facets.examples] meaningless normative paragraph in examples

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, or (at the editor's option) replace it with a place-holder to keep the paragraph numbering the same.

---

29. 27.4.2 [lib.ios.base] ios_base needs clear(), exceptions()

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:
One of:
A. Move the definitions of basic_ios<> members clear, setstate, good, eof, fail, bad, and exceptions from basic_ios<> to ios_base. In particular, move them from the basic_ios<> synopsis 27.4.4 [lib.ios] and the definitions 27.4.4.3 [lib.iostate.flags] to the ios_base synopsis 27.4.2 [lib.ios.base] and definitions 27.4.2.1.2 [lib.ios::fmtflags] sections, respectively.

B. Change the description in 27.4.2.4 [lib.ios.members.static] 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 subobject of a basic_ios<> object or subobject, the failure may be detected by

---

30. 21.3 [lib.basic.string] string ctors specify wrong default allocator

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:
One of:
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);
```

---

31. 23.2.2 [lib.list] list operations should not invalidate list iterators

A resolution was passed to add a statement that list iterators are not invalidated by various list<> operations which do not affect the specific nodes referred to. That statement failed to be edited into the final draft. The correct semantics of list<> depend on such a statement; we should restore it.

Proposed Resolution:
(none yet)

---

32. 27 [lib.input.output] iostreams use operator== on int_type values

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:
(not ready yet)

---

33. 27.7 [lib.string.streams] stringstream in/out pointer positions

There have been reports about inconsistencies in the description of stringstream "file positions".

---

34. 22.2.2.1.2 [lib.facet.num.get.virtuals] op<< exit conditions inconsistent

The condition of the iterator (file position) and the states of the failbit and eofbit flags for the various input parsing functions in facets num_get, time_get, money_get, and istream operators << and member get have been noted to be inconsistent.

---

35. 21.1.1 [lib.char.traits.require] char_traits<>::lt and eq vs compare

I have a note that suggests the char_traits<> members lt and eq are inconsistent with the definition of member compare.

Proposed Resolution:
(none)

----
36. 22.2.5.1 [lib.locale.time.get] do_get_monthname synopsis missing argument

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.

This could reasonably be considered a purely-editorial inconsistency.

Proposed Resolution:
In 22.2.5.1 [lib.locale.time.get], replace 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;
```