Doc. no.:	J16/97-0036
	WG21/N1074
Date:	26 May 1997
Project:	Programming Language C++
Reply to:	Beman Dawes
	beman@esva.net

Libraries Issues List for CD2 – Version 6

History:
Version 0: Distributed in the pre-Nashua mailing.
Version 1: Distributed via the net before the Nashua meeting. Adds additional issues.
Version 2: Distributed at the start of the Nashua meeting. Adds additional issues.
Version 3: Distributed at the Nashua meeting. Includes library related ANSI public comments.
Version 4: Distributed in the post-Nashua mailing:
- Reflects changes to version 3 per the J16 meeting minutes.
- Issues with a status of Open (US) are the US National Body comments on CD-2.

Version 5: Distributed via the net in April, 1997.
- Issues close in Version 4 moved to Closed Issues section at end.
- New issues added.

Version 6: Distributed in the Pre-London mailing.
- Updated discussion and resolutions.
- New issues added.

Issues list purpose

The main purpose of the issues list is to serve as the Committee's memory.

The only issues the Committee is committed to handling are those reflecting ISO National Body comments. These issues are identified on the status line by the initials of National Bodies who have included the issue in their official comments.

At this point, putting an issue on the list is not a promise that it will be handled unless it is an NB comment. It does show that we know about the issue and perhaps decided to do nothing about it now.

Open Issues

Library editorial issues

Issue: **CD2-lib-edit-001 Nashua small and editorial issues**
Section:
Status: Open (US)
Description:

Proposed resolution:

Page 17-8 Sec 17.3.1.3 Clause 2: "implementation has has" to "implementation has"

18.4.1.1 [lib.new.delete.single] change throw(bad_alloc) to throw(std::bad_alloc).

18.4.1.2 [lib.new.delete.array] change throw(bad_alloc) to throw(std::bad_alloc).

Page 23-20 Top of page "nmespace" to "namespace"

23.2.5 [lib.vector.bool] delete "= allocator<bool>". See lib-5241 and lib-5242 for rationale.

Page 21-4 Sec 21.1.3 Clause 8 "derived classed" to "derived classes"

Page 23-6 Sec 23.1.2 Clause 4 "equal keys" to "equivalent keys"

Page 23-38 Sec 23.3.4 Clause 2 “the a_eu operations” to “the a_eq operations”

Page 24-20 Sec 24.5.1.1 Clause 3 “a copy of s” to “a copy of x”

24.4.1.1 [lib.reverse.iterator] change “Distance” to “difference_type” in 5 places.

24.4.1.1 [lib.reverse.iterator] change “Reference” to “reference” in 2 places, and “Pointer” to “pointer” in 1 place.

24.4.1.3.3 [lib.reverse.iter.op.star] change “Reference” to “reference” in 1 place.

24.4.1.3.4 [lib.reverse.iter.opref] change “Pointer” to “pointer” in 1 place.

In section 4.10 paragraph 1, the term “nul pointer constant” is used, but in section 18.1 paragraph 4, the term “nul-pointer constant” is used.

In 21.3.7.9 paragraph 1, start a new line before “After the last character (if any) is”.

21.3.1 basic_string constructors [lib.string.cons] For the constructor:

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

In Table 39, remove the line “get_allocator() str.get_allocator()”

This is a holdover from a previous version of this constructor which didn't take its own Allocation& argument but instead used str.get_allocator().

21.3.6.8 basic_string::compare [lib.string::compare]
The function “int compare(const basic_string<charT,traits,Allocator>& str)” should be const as declared in '21.3 Template class basic_string
[lib.basic.string]' at Paragraph 4.
The signature should be “int compare(const basic_string<charT,traits,Allocator>& str) const”

20.4.4.3 uninitialized_fill_n [lib.uninitialized.fill.n] template <class ForwardIterator, class Size, class T> void uninitialized_fill_n(ForwardIterator first, Size n, const T& x); the 'Effects:' section is simply incorrect; currently it is:

Effects:

```
    while (n--)
      new (static_cast<void*>(&*result++))
        typename
iterator_traits<ForwardIterator>::value_type(*first++);
```

This is erroneous. It must be:

Effects:

```
    while (n--)
      new (static_cast<void*>(&*first++))
         typename iterator_traits<ForwardIterator>::value_type(x);
```

5) 27.4.2.3 ios_base locale functions [lib.ios.base.locales] For the function 'locale imbue(const locale loc);' There are extraneous characters in the 'Returns:' section at the line “output operations.La Postcondition: loc == getloc().” Remove the extraneous 'La'.

Issue: **CD2-lib-edit-002 London small and editorial issues**
Section:
Status: Open
Description:

Proposed resolution:

17.3.3.1.2 [lib.global.names] (provided by Josee Lajoie) Change:

> --Each name that begins with an underscore is reserved to the implementation for use as a name with file scope or within the namespace std in the ordinary name space.

To:

--Each name that begins with an underscore is reserved to the implementation for use as a name in the global namespace or within the namespace std that is a member of the global namespace.

20.1.1 [lib.equalitycomparable]

"If a == b and b == a, then a == c" should be " If a == b and b == c, then a == c"

21.1.2 [lib.char.traits.require] The description of find() in table 37 has a typo: q is a pointer to a character, hence it should be `X::eq(*q,c)` instead of `X::eq(qc)`.

21.1.2 [lib.char.traits.require] The description of not_eof() in table 37 has a typo: e and X::eof() are of type int_type, hence eq_int_type() should be used for comparison instead of eq().

23.1 [lib.container.requirements] paragraph 5 begins: "In Table 66, X denotes…". Change to "In Table 66 and 67, X denotes…".

23.1.1 [lib.sequence.reqmts] paragraph 3, change "… `a` denotes value of X, …" to "… `a` denotes a value of X, …"

23.1.1 [lib.sequence.reqmts] The Optional Sequence Operations table specifies return type (in four places) as:

```
T&; const T& for constant a
```

This table should be changed to match other container requirements tables, which specify the return type as:

```
X::reference; X::const_reference for constant a
```

23.1.2 [lib.associative.reqmts] Associative container requirements table entry for `insert(i,j)` is unclear as to the effects of inserting keys already in the container. The wording for `insert(p,t)` can be used to correct the problem.

Change `insert(i,j)` wording from:

inserts the elements from the range [i, j) into the container.

To:

inserts each element from the range [i, j) into the container if and only if there is no element with key equivalent to the key of the element being inserted in containers with unique keys; always inserts the element being inserted in containers with equivalent keys .

25 [lib.algorithms] paragraph 9. The example uses an obsolete form of the distance function.
Change:

```
{ Distance n;
  distance(a, b, n);
  return n;
}
```

to:

```
{ return distance(a, b); }
```

Clause 17 – Library Introduction

Issue: **CD2-17-001 Illegal member functions default arguments**
Section: 17.3.4.4 para 4 [lib.member.functions]
Status: Open (US)
Description:

The standard library uses default arguments to member functions of template classes that are dependent on template arguments in a number of places, even though such defaults are not currently legal C++. This clash between the language and the library is currently resolved (17.3.4.4 paragraph 4) by specifying that default arguments are to be treated as equivalent overloadings with additional functions. This is called the "Plum Patch".

Defaults can be removed from the library by either eliminating the default case, or providing an additional overload.

There are at least 41 signatures affected. Because some of these include multiple defaults, there are a total of 54 cases.

An additional signature would have to be added to the library to deal with each of the cases, a total of 54 new signatures in all.

Of the 41 signatures affected, 29 are constructors. Because the classes involved already have several constructors, adding additional constructors is particularly confusing. The use of default arguments makes these classes easier to understand and easier to use.

Proposed Resolution:

Default arguments for member functions of template classes should either be added to the language or removed from the library. The "Plum Patch" is not intended to be a permanent solution to the problem. The core working group continues to investigate a language change while the library working group continues to investigate library changes.

Requester: Beman Dawes <beman@esva.net>
References: 97-0013/N1051, lib-5484

Issue: CD2-17-002 Pointers to Standard C library functions not portable
Section: 17.3.2.2 [lib.using.linkage]
Status: Open (US)
Description:

In Stockholm the language was changed to make the "language linkage" part of the type.

However, the library definition says (17.3.2.2 [lib.using.linkage], paragraph 2) that it is unspecified whether a name from the Standard C Library has C or C++ linkage.

The effect of this is that it is not possible to write a portable program that uses a pointer to a Standard C library function or that calls a Standard C library function that takes a function pointer as a parameter.

For example, the `atexit` function might be declared as either:

```
extern "C" int atexit(void (*f)(void));
```

or

```
extern "C++" int atexit(void (*f)(void));
```

How does one portably call `atexit` now?

```
void xxx(){}  // a C++ function
...
atexit(xxx);  // ill-formed if atexit has C linkage
```

I was not involved in the WG that introduced this language change, but it seems to me that if the language linkage is now part of the type then the language linkage of every library function must now be well defined.

Post-Nashua discussion:

The possible resolution based on overloading discussed in Nashua has been removed as unworkable. Several LWG members support "C++" linkage on the grounds that this is, after all, the C++ library.

Proposed Resolution:

Change 17.3.2.2 [lib.using.linkage], paragraph 2 to read (choose one):

1) Names from the Standard C library declared with external linkage have `extern "C"` linkage.

2) Names from the Standard C library declared with external linkage have `extern "C++"` linkage.

Requester: John H. Spicer <jhs@edg.com>
References: lib-5141

Issue: CD2-17-003 Remove C library names from namespace std
Section:
Status: Open (US)
Description:

We believe that C library names should be removed from namespace std. The draft currently states (Clause 17, Annex D) that the C++ Standard library will provide 18 ISO C library headers in a < cname> form which brings ISO C names into the namespace std and a <name.h> form which bring ISO C names into both the std and global namespace (excluding macros).

We believe that the implementation for this is highly error prone, leading to unmaintainable C headers and serious bugs. Some of our major concerns are:

- maintaining duplicate copies of the .h headers, one supplied by C and one by C++.
- adding complex macros to headers to avoid nested namespaces.
- ensuring that names are consistently available (or not) in namespace std regardless of the order of header file inclusion in a user program.
- coordinating an effort to modify, rewrite, reorganize C headers supplied by a C development environment which is outside of the scope of the C++ environment.

We believe that in practice the benefits of putting ISO C names into namespace std do not outweigh the increased complexity required for compliance. The burden of this support is not limited to C++ compiler/library vendors. It will impact any independent C++ library/tool vendor and operating system provider all of which will need to ensure that the correct C/C++ header interfaces are in place.

Proposed Resolution:

(The proposed resolution was supplied by the requesters. The LWG did not discuss this issue in Nashua, and the proposed resolution is not part of the US NB Comment.)

The resolution is to change the Working Paper as follows:

- 17.3.1.2 table 12, C++ Headers for C Library Facilities delete the leading "c" from header names and append ".h".
- Remove 17.3.1.2 paragraph 4, 7 and footnote 153. Add the ".h" headers place all their names into the global namespace.
- Delete from Annex D the [.depr.c.headers] section.
- Change references to std::ISO-C-name to ISO-C-name

Requester: Sandra Whitman, Judy Ward
References: Public comment 19/Whitman & Ward
Lib reflector messages: 4598-4611,4614-4615,4618-4626,4628,4630,4632-4636,4638-4641,4643,4645-4647,4650-4656,4662-4664,4666,4676,4689,4690

Clause 18 – Language Support

Issue: CD2-18-001 `Offset` macro needs additional restrictions
Section: 18.1 Types [lib.support.types]
Status: Open (US)
Description:

The offsetof macro (18.1) is restricted to work on POD-structs and POD-unions. So far so good. Two problems:

1. A POD-struct is allowed to have static data and non-virtual member functions. Surely they should be explicitly excluded from use with the `offsetof` macro.

Proposed Resolution:

Modify the 18.1 section referring to `offsetof` to say:

"The result of applying the `offsetof` macro to a field which is a static data member, a function member is undefined."

"Undefined" will allow existing implementations to continue to be valid. If we require error detection, compilers will have to jump through hoops to recognize the `offsetof` macro, since by the time the macro is expanded the fact that invalid code came from "`offsetof`" is lost.

Requester: Steve Clamage <clamage@taumet.Eng.Sun.COM>
References: lib-5249

Issue: CD2-18-002 Behavior of abort() with regard to atexit() functions unspecified
Section: 18.3 [lib.support.start.term]
Status: Open (US)
Description:

In 18.3 [lib.support.start.term], `atexit()` and `exit()` are described as having additional behavior compared to the Standard C library.

Shouldn't`abort()` also be included here? 3.6.3 [basic.start.term] explicitly states that calling `abort()` means that static destructors and atexit-registered functions do not get called. The C standard [ISO C 7.10.4.1, 7.10.4.2] does not explicitly say that atexit-registered functions are not called upon `abort()`, and indeed in some implementations they are.

If `abort()`'s additional behavior is included in 18.3, it should also be included in C.4.4 [diff.mods.to.behavior].

Proposed Resolution:

Add after 18.3 [lib.support.start.term] paragraph 2:

```
void abort(void)
```

The function `abort()` has additional behavior in this International Standard:

> The program is terminated without executing destructors for objects of automatic or static storage duration and without calling the functions passed to `atexit()` (3.6.3).

Add to list in C.4.4 [diff.mods.to.behavior], paragraph 1:

> -- abort

Requester: Jonathan Schilling <jls@sco.com>

Issue: CD2-18-007 Clause 18 member functions should return NTBS
Section: 18.4 [lib.support.dynamic], 18.5 [lib.support.rtti], 18.6 [lib.support.exception]
Status: Open
Description:

A number of member functions in Clause 18 return implementation-defined character strings. The current wording in the WP is inconsistent in describing these return values: for exception::what() the return value is "an implementation-defined NTBS", while for all other such functions, including bad_exception::what() which overrides the above, the return value is simply "implementation-defined". The latter description opens the possibility that the implementation returns zeroor perhaps even a bad pointer, which is not desirable.

Note that the proposed resolution does not require the implementation to return anything 'useful'; e.g. the implementation could return "" for everything. The goal is simply that usages such as

```
cout << typeid(*pd).name() << endl;
```

and

```
catch (std::bad_exception& be) {
```

```
cerr << be.what() << endl;
}
```

are guaranteed to execute with (implementation-) defined behavior.

[One additional note: in a Core-3 discussion, Bill Gibbons said that he thought the language should be "Returns: A pointer to an implementation-defined NTBS." However looking elsewhere in the WP, it seems that an NTBS is considered to be of type const char*; e.g. see footnote 146 in Clause 17, or the usage in 27.8.1.3.]

Proposed Resolution:

Change 18.4.2.1 /5 [lib.bad.alloc], 18.5.1 /7 [lib.type.info], 18.5.2 /5 [lib.bad.cast], 18.5.3 /5 [lib.bad.typeid], and 18.6.2.1 /5 [lib.bad.exception] to

Returns: An implementation-defined NTBS.

Issue: **CD2-18-008 Numeric_limits incorrectly excludes user specializations**
Section: 18.2.1 [lib.limits]
Status: Open
Description:

Evidently a series of well-intentioned editorial "clarifications" has clarified the purpose right out of numeric_limits<>, so that users appear not to be permitted to specialize it for their own numeric types. This is the opposite of the intention of the original proposal, though there have been no votes to change it.

Proposed Resolution:

In 18.2.1 [lib.limits], change paragraph 4 from

Non-fundamental types, such as complex<T> (26.2.2), shall not have specializations.

to

Non-fundamental standard types, such as complex<T> (26.2.2), shall not have specializations.

Requester: Nathan Myers <ncm@cantrip.org>

Clause 19 – Diagnostics

Clause 20 – Utilities

Issue: **CD2-20-001 Raw storage iterator implies reference to void**
Section: 20.4.2p1 [lib.storage.iterator]
Status: Open (US)
Description:

8.3.2 [dcl.ref] makes "reference to void" ill-formed. However, it seems to be implicitly required by the following:

20.4.2p1 [lib.storage.iterator] uses the template instance

```
iterator<output_iterator_tag,void,void>
```

24.2 [lib.iterator.synopsis] defines

```
template<class Category, class T, class Distance=ptrdiff_t,
     class Pointer=T*, class Reference=T&> struct iterator.
```

Unless there's a specialization of template struct iterator that we've missed, the definition in 20.4.2p1 uses `void&` as the default argument to the template.

Proposed Resolution:

Change 20.4.2 [lib.storage.iterator] to supply all 5 arguments to the iterator template, using void for the currently missing ones.

Requester: Steve Adamczyk <jsa@edg.com>

References:

Issue: **CD2-20-003 Function adaptors won't work with void return types**
Section: 20.3.8 Adaptors for pointers to members [lib.member.pointer.adaptors]
Status: Open (US)
Description:

The mem_fun adaptors specified in the December 1996 draft are not partially specialized for return type `void` (Section 20.3.8).

Since Bjarne's proposal "Relaxing the Rules for Void" was not accepted, don't we need to add some partial specializations (as described in Section 2.3 of X3J16/96-0030,WG21/N0848)?

Or is this an implementation issue and they do not need to be explicitly specified in the draft?

In addition, John Skaller pointed out in c++std-lib-5319 that the adaptors for pointers to functions in Section 20.3.7 also require partial specializations.

Proposed Resolution:

There are three possible resolutions:

1) Relax the language rules for returning void as described in X3J16/96-0031, WG21/N0849
2) Determine that this is an implementation issue which does not need to be explicitly specified in the draft.
3) Add partial specializations to Section 20.3.8 and Section 20.3.7 as follows:

Section 20.3.8:

```
template <class T>
class mem_fun_t<void,T> : public unary_function<T*,void>
{ public:
        explicit mem_fun_t(void (T::*p)());
        void operator()(T* p);
};

template <class T, class A>
class mem_fun1_t<void,T,A> : public binary_function<T*,A,void>
{ public:
        explicit mem_fun1_t(void (T::*p)(A));
        void operator()(T* p, A x);
};

template <class T> mem_fun_t<void,T>
    mem_fun(void (T::*f)());

template <class T, class A> mem_fun1_t<void,T,A>
    mem_fun1(void (T::*f)(A));

template <class T>
class mem_fun_ref_t<void,T>  : public unary_function<T,void>
{ public:
        explicit mem_fun_ref_t(void (T::*p)());
        void operator()(T* p);
};

template <class T, class A>
class mem_fun1_ref_t<void,
                     T,A> : public binary_function<T,A,void>
{ public:
        explicit mem_fun1_ref_t(void (T::*p)(A));
        void operator()(T* p, A x);
};

template <class T> mem_fun_ref_t<void,T>
    mem_fun_ref(void (T::*f)());

template <class T, class A> mem_fun1_ref_t<void,T,A>
```

```
        mem_fun1_ref(void (T::*f)(A));
```

Section 20.3.7

```
template <class Arg>
class pointer_to_unary_function<Arg,void> :
    public unary_function<Arg, void>
{ public:
        explicit pointer_to_unary_function (void (*f)( Arg));
        Result operator() (Arg x) const;
};

template <class Arg> pointer_to_unary_function<Arg, void>
    ptr_fun(void (*f)(Arg));

template <class Arg1, class Arg2>
class pointer_to_binary_function<Arg1,Arg2,void> :
    public binary_function<Arg1, Arg2, void>
{ public:
        explicit pointer_to_binary_function
             (void (*f)(Arg1, Arg2));
        void operator() (Arg1 x, Arg2 y) const;
};

template <class Arg1, class Arg2>
  pointer_to_binary_function<Arg1, Arg2, void>
    ptr_fun(void (*f)(Arg1, Arg2));
```

Requester: Judy Ward < j_ward@decc.enet.dec.com>
References: lib-5318, lib-5319, lib-5321, X3J16/96-0030,WG21/N0848,X3J16/96-0031,WG21/N0849

Issue: **CD2-20-004 Relax restrictions on allocator pointer**
Section: 20.1.5 Allocator requirements [lib.allocator.requirements]
Status: Open (US)
Description:

The restrictions on the allocator pointer type currently in the WP are too severe.

Proposed Resolution:

Relax the restrictions on allocator pointer type, as described in 97-0018/N1056

Requester:
References: 97-0018/N1056

Issue: **CD2-20-005 binder1st syntax error**
Section: bind1st (20.3.6.2)
Status: Open
Description:

The RETURNS clause for bind1st (20.3.6.2) reads
binder1st<Operation>(op, Operation::first_argument_type(x)).

There’s a similar problem for binder2nd, of course.

Sean Corfield in Message c++std-lib-5563: Maybe we need wording in Clause 17 that says:

"Wherever this International Standard specifies dependent-type-name::name and the context requires the keyword typename’, it is implemented ’as if’ the keyword were present in the specification." *grin*

Proposed Resolution:

Change the returns clause for bind1st (20.3.6.2) [lib.bind.1st] to:

binder1st<Operation>(op, typename Operation::first_argument_type(x))

Make a similar change to the returns clause for (20.3.6.4) [lib.bind.2nd].

Requester: Matt Austern

Issue: **CD2-20-006 mem_fun adaptor const problems**
Section: 20.3.8 [lib.member.pointer.adaptors]
Status: Open
Description:

Three problems, all of which are somewhat related.

(1) mem_fun_t's operator()() is not a const member function. This means that you can't pass amem_fun_t to any function that takes its argument by const reference. (Which many of the STL algorithms do.)

This is a fairly trivial problem, but, from experience, it's one that users will have a lot of trouble with. The sorts of error messages that you get when you pass a function object by const reference, and it doesn't have a const operator()(), are very hard to understand. People will spend hours looking at their code and trying to figure out what's wrong.

(2) You can't apply amem_fun_t<S, T> to an object of type const T*, but only to one of type T*. This is a serious limitation: it means that the mem_fun_t adaptor can't be used

(3) You can't create amem_fun_t where the member function pointer is initialized to a const member function. If, for example, you have

```
struct A {
  int x;
  int twice() const { return 2*x; }
};
```

then mem_fun(A::twice) doesn't work, because the argument is of type "int (A::*)() const" and mem_fun expects its argument to be of type "S (T::*)()".

Proposed Resolution:

Change 20.3.8 [lib.member.pointer.adaptors] to make operator() const in mem_fun_t, mem_fun1_t, mem_fun_ref_t, and mem_fun1_ref_t.

Requester: Fumiki Fukuda <ffukuda@ntes.nec.co.jp>, Matt Austern
References: lib-5716, 5776, 5777

Clause 21 – Strings

Issue: **CD2-21-001 basic_string element**
Section: 21.3.4 basic_string element access [lib.string.access]
Status: Open (US)
Description:

This clause says that the reference returned by the non-const version of `operator[]` is invalid after "any subsequent call to `c_str()`, `data()`, or any non-const member function for the object." This would seem to make expressions such as

```
foo(s[a], s[b])
```

invalid, where s is not const, as the second call to `operator[]` would invalidate the reference returned by the first call to `operator[]`. In general, it seems unreasonable that a call to `operator[]` would invalidate the reference returned by a previous call to `operator[]`.

Andrew Koenig questions in lib-5251 whether the following might be invalid:

```
s[i] = s[j];
```

Matt Austern discusses several possible resolutions in lib-5250.

Discussions in Nashua concluded that the only acceptable solution is to incorporate into the WP sufficient requirements on reference lifetimes that the above examples must work. These requirements should not disallow reference counted strings.

Post Nashua Discussion:

Invalidation of references, pointers, and iterators referring to the elements of a basic_string sequence can occur as a result of:

- The semantics of the operation itself. For example, clear().
- Memory reallocation, when size() would otherwise exceed capacity().
- In reference counted implementations, first modification or potential modification of elements. For example, non-const begin() after operator=().

The current CD fails to correctly specify when invalidation occurs:

- Invalidation due to operation semantics is not currently mentioned at all.
- Invalidation due to reallocation is described but there is no mention of when it may occur.
- Invalidation due to potential modification of elements is described for only one of the groups of members involved, and even that description fails to limit this case to first use.

Proposed Resolution:

Delete from 21.3.3 basic_string capacity [lib.string.capacity]:

> Reallocation invalidates all the references, pointers, and iterators referring to the elements of the sequence.

Delete from 21.3.4 basic_string elements access [lib.string.access]:

> Effects: The reference returned is invalid after any subsequent call to `c_str()`, `data()`, or any non-const member function for the object.

After paragraph 4 of 21.3 Template class `basic_string` [lib.basic.string] add:

> References, pointers, and iterators referring to the elements of a basic_string sequence may be invalidated by the following uses of that basic_string object:
>
> - As an argument to non-member functions `swap()` ([lib.string.special]), `operator>>()` ([lib.string.io]), and `getline()` ([lib.string.io]).
> - As an argument to `basic_string::swap()`.
> - Calling `data()` and `c_str()` member functions.
> - Calling non-const member functions, except `operator[]()`, `at()`, `begin()`, `rbegin()`, `end()`, and `rend()`.
> - Subsequent to any of the above uses except the forms of `insert()` and `erase()` which return iterators, the first call to non-const member functions `operator[]()`, `at()`, `begin()`, `rbegin()`, `end()`, or `rend()`.

Requester: Kevin S. Van Horn <kevin.s.vanhorn@iname.com>
References: lib-5248, lib-5250, lib-5251, lib-5252

Issue: CD2-21-002 `basic_string` member require non-existent `traits::eos()`
Section: 21.3.4 [lib.string.access], 21.3.6 [lib.string.ops] (2 places), 27.6.1.2.3 [lib.istream::extractors] (2 places), 27.6.2.7 [lib.ostream.manip]
Status: Open (US)
Description:

Several basic_string member functions are defined to require `traits::eos()`.

Unfortunately, character traits do not have an `eos()` member, either in the requirements table, or in the provided specializations.

Proposed Resolution:

Nathan Myers in lib-5247: "Yes, member `eos()` was removed; use `char_type()` as end-of-string where it is needed. We need to fix the Draft where it mentions `eos()`."

Put eos back in traits.

Requester: Hans-Juergen Boehm <boehm@mti.sgi.com>
References: lib-5245, lib-5247

Issue: CD2-21-003 "Maximum size" undefined
Section: 21.3.3 [lib.string.capacity] par 3
Status: Open (US)
Description:

What is "maximum size"? It can be deduced from reading about `resize()`, but I think it should be stated here.

Proposed Resolution:

Add a cross reference to table 66 “Container Requirements”.

Requester: Dag Brück <dag@dynasim.se>
References:

Issue: CD2-21-004 Capacity() description unclear
Section: 21.3.3 [lib.string.capacity] par 8 and 9
Status: Open (US)
Description:

Is `reserve()` guaranteed to accept any argument, even `size_type(-1)`? I think the description of `capacity()` is unclear, it doesn't stand for itself.

Proposed Resolution:

Maybe we should define it as:

> Returns: a value not less than the value of `res_arg` of the last call of `reserve()`, or an unspecified value if `reserve()` has not been called for this object. The returned value is not less than `size()`.

or something along those lines.

Requester: Dag Brück <dag@dynasim.se>
References:

Issue: CD2-21-005 insert() synopsis has default argument, definition doesn’t
Section: 21.3.5.4 basic_string::insert [lib.string::insert], paragraph 10
Status: Open (US)
Description:

The definition of `basic_string::insert()` does not include the default argument which is part of the synopsis in [lib.basic.string]. It probably should, as `insert()` of sequences have a default argument.

Proposed Resolution:

Change the appropriate definition of insert() in [lib.string.insert] to:

```
iterator insert(iterator p, charT c = charT());
```

Requester: Dag Brück <dag@dynasim.se>
References:

Issue: CD2-21-007 Character traits incorrect
Section: 21.1.2 Character traits requirements [lib.char.traits.require]
Status: Open (US)
Description:

In section 21.1.2 table 37, it seems that `not_eof()` should use `eq_int_type()` instead of `eq()`. What is the behavior for integer values passed to `not_eof()` for which `eq_int_type()` returns false, but `eq()` returns true. For example, consider the value 0x7FFF where `eof()` returns 0xFFFF and a char is 8 bits.

Proposed Resolution:

Change the table appropriately.

Requester: Arch Robison & David Nelson

References: Public comment 28/Robison/Nelson

Issue: CD2-21-009 String complexities are not specified

Section: 21.1.2 Character traits requirements [lib.char.traits.require]

Status: Open

Description:

If s1 and s2 are of type string, then users have no way of knowing whether operations like s1 = s2 and string(s1) are O(1) or O(N). This is a serious problem, since the type of code you write will look very different depending on whether you think that copy construction is slow or fast.

Proposed Resolution:

There are two options:

1. Specify that these operations are O(1). This effectively mandates reference counting.
2. Specify that they are O(N). This gives users fait warning that they shouldn't count on reference counting.

Requester: Matt Austern

References:

Issue: CD2-21-010 Mutable iterators into strings are expensive

Section: 21

Status: Open

Description:

In a reference counted implementation potentially mutative operations are much more expensive than constant ones. In particular, `begin()` and `end()` are overloaded on const. The non-const versions probably have O(N) complexity. Users must write something like:

```
((const string &)s).begin()
```

to avoid this behaviour.

Proposed Resolution:

Rename the non-const versions of `begin()` and `end()` to mutable_begin() and mutable_end(). Users who want mutable iterators must ask for them explicitly.

Requester: Matt Austern

References:

Issue: CD2-21-011 Getline should not skip whitespace

Section: 21.3.7.9 [lib.string.io]

Status: Open

Description:

The function getline for basic_string as described in the WP does skip whitespace. I think this is wrong.

```
template<class charT, class traits, class Allocator>
  basic_istream<charT,traits>&
   getline(basic_istream<charT,traits>& is,
        basic_string<charT,traits,Allocator>& str,
        charT delim);
```

Effects:

Begins by constructing a sentry object k as if by basic_istream<charT,traits>::sentry k(is). If bool(k) is true, it calls str.erase() and then extracts characters from is and appends them to str as if by calling str.append(1,c) until any of the following occurs:

Proposed Resolution:

Change

```
basic_istream<charT,traits>::sentry k(is)
```

to:

```
basic_istream<charT,traits>::sentry k(is,true)
```

Requester: Philippe Le Mouel

References: lib-5548

Issue: **CD2-21-012 Extraction of C strings and basic_strings inconsistent regarding null bytes**
Section: 21.3.7.9 [lib.string.io] and 27.6.1.2.3 [lib.istream::extractors]
Status: Open
Description:

The extractors for C strings, i.e. charT*, are slightly different from their counterparts for basic_string<charT>. The extraction of a C string [lib.istream::extractors] stops when a null byte is found in the next position. The same is not mentioned as a terminating condition for the basic_string extractor [lib.string.io].

An oversight or intention?

** Discussion: Jerry: I believe stopping on NULL was iostream classic behavior, and I don't see any reason to change it. Null's aren't allowed to appear in NTBS's.

Proposed Resolution:

Add to 21.3.7.9 [lib.string.io] extractor description:

> The extraction of a basic_string shall stop when a null byte is found in the next position.

Requester: Klaus Kreft & Angelika Langer

Issue: **CD2-21-013 Extraction of C strings and basic_strings inconsistent regarding width**
Section: 21.3.7.9 [lib.string.io] and 27.6.1.2.3 [lib.istream::extractors]
Status: Open
Description:

The extractors for C strings, i.e. charT*, are slightly different from their counterparts for basic_string<charT>. The number of characters extracted is different. For C strings it is width()-1; for basic_string it is width(). Is this to leave room for a terminating '\0' in a C string?

** Discussion: Jerry: Extractor's generally ignore width. But for charT I needed some way to pass the size of the array you're storing into, and yes you need the extra char to store a null. Since there is no such limitation on the string extractor I think I would have opted for ignoring width there too. If the argument for looking at width is to make it the same as the charT* extractor then you're right that this is an inconsistency.

Proposed Resolution:
Requester: Klaus Kreft & Angelika Langer

Issue: **CD2-21-014 Table 37: eq_int_type() insufficiently defined**
Section: 21.1.2 [lib.char.traits.require]
Status: Open
Description:

The description of eq_int_type is incomplete: A return value is defined for
1. values of char_type
2. values of int_type, if they are copies of eof()

In all other cases the return value is not even implementation-defined; it is simply undefined.

Proposed Resolution:
Add words to the description saying:

Returns false, if one value is a copy of eof() and the other is not.
The result of comparing two values of int_type, none of which is a copy of eof(), is implementation-defined.

Requester: Klaus Kreft & Angelika Langer

Issue: **CD2-21-015 Value of eof() implementation-defined?**

Section: 21.1.4 [lib.char.traits.specializations]
Status: Open
Description:

Why is char_traits<char>::eof() not required to be EOF, and char_traits<wchar_t>::eof() is allowed to be different from WEOF? I had expected that at least EOF would be required for compatibility with the classic IOStreams.

Proposed Resolution:
Requester: Klaus Kreft & Angelika Langer

Issue: CD2-21-016 find() takes an int instead of a size_t
Section: 21.1.4 [lib.char.traits.specializations]
Status: Open
Description:

Why does char_traits<charT>::find() take a int parameter whereas all other functions like move() or copy() take a size_t parameter?

Proposed Resolution:
Requester: Klaus Kreft & Angelika Langer

Issue: CD2-21-017 basic_string::getline() skipping whitespace
Section: 21.3.7.9[lib.string.io]
Status: Open
Description:

The function getline for basic_string as described in the WP skips whitespace. I think this is wrong.

> Effects: Begins by constructing a sentry object k as if by
> `basic_istream<charT,traits>::sentry k(is)`. ...

Proposed Resolution:

Change the above to:

> Effects: Begins by constructing a sentry object k as if by
> `basic_istream<charT,traits>::sentry k(is,true)`. ...

Requester: Philippe LeMouel, Nathan Myers
References: lib-5548, 5772

Issue: CD2-21-018 basic_string::insert() of a range does not specify effect of overlap
Section: 21.3.5.4 [lib.string::insert]
Status: Open
Description:

Issue 23-009 deals with the specification for the range flavor of insert() when the range is from the container itself.

Does the same issue apply to basic_string?

There are four basic_string signatures taking a range and modifying *this:

append
assign
replace
insert

The first three present no problem since they specify operation on the range in terms of construction. For example, append(InputIterator first, InputIterator last) is specified as:

> Returns: `append(basic_string<charT,traits,Allocator>( first,last))`.

In other words, range append() must work as if it constructs a basic_string from that range and then calls the flavor of append() which takes a basic_string argument.

Thus my reading is that append(), assign(), and replace() must work correctly even if the range is from *this.

The specification of insert() is:

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

Requires: p is a valid iterator on *this. [first, last) is a valid range.

Effects: inserts copies of the characters in the range [first, last) before the character referred to by p.

This leaves the question of overlap open. I suggest making insert() consistent with the others.

Proposed Resolution:

Replace the range insert effects clause (21.3.5.4 [lib.string::insert] paragraph 16) with:

Returns: `insert(p,basic_string<charT,traits,Allocator>(first,last))`.

Requester: Beman Dawes
References: lib-5828

Clause 22 – Localization

Issue: **CD2-22-001 Locale ctype<char> needs virtual do_widen & do_narrow**
Section: 22.2.1.3 [lib.facet.ctype.special], 22.2.1.3.4 [lib.facet.ctype.char.virtuals]
Status: Open (US)
Description:

During final proofreading of the Committee Draft in Kona, several members of the library group noticed that the locale facet ctype<char> specialization has members widen() and narrow() that do not delegate to virtual members do_widen() and do_narrow() like other members, resulting in Box 22 in the Draft.

These members must be virtual to support alternative encodings for type char, as the rest of the locale apparatus provides. While adding apparatus in support of flexibility for 8 bit characters might seem backward-looking, there is no reason to assume that "char" will always represent an 8 bit type. We have every reason to believe that some language implementations will use a larger type, and represent Unicode (along with other encodings) in a char. For this reason it is important that the char facilities remain flexible.

Proposed Resolution:

Add to the protected section of the specialization ctype<char> in 22.2.1.3 [lib.facet.ctype.special]:

```
virtual char do_widen(char c) const;
virtual const char* do_widen(const char* low,
                             const char* high,
                             char* to) const;
virtual char do_narrow(char c, char dfault) const;
virtual const char* do_narrow(const char* low,
                              const char* high,
                              char dfault, char* to) const;
```

In 22.2.1.3.2 [lib.facet.ctype.char.members], change the definitions of (non-virtual) members `ctype<char>::widen` and narrow to:

Returns: `do_widen(low, high, to)`

and

Returns: `do_narrow(low, high, to)`

respectively. Add the following to 22.2.1.3.4 [lib.facet.ctype.char.virtuals]:

```
char        do_widen(char) const;
const char* do_widen(char* low,
                     const char* high, char* to) const;
char        do_narrow(char) const;
const char* do_narrow(char* low, const char* high,
                      char dfault, char* to) const;
```

Requester: ncm@cantrip.org (Nathan Myers)
References:

Issue: CD2-22-002 Locale numeric parsing should use widen, not narrow
Section: 22.2.2.1.2 [lib.facet.num.get.virtuals]
Status: Open (US)
Description:

The description of the parsing process for the locale numeric facet num_get<> describes a process in which characters are read from an input sequence, converted to a corresponding char value, and then accumulated into a numeric value.

This process is necessarily slow, because it implies a virtual function call per character. The alternative is to widen the candidate digit values (once) to char_type, and then compare the input character values directly. This allows the time-consuming operations to be done once, on the first conversion, and then used on subsequent conversions. This does have different semantics: only the canonical digit or sign character in the codeset is recognised.

This proposal could have a substantial effect on the performance of numeric input parsing.

Proposed Resolution:
The LWG recommends taking no action on this issue pending discussion with relevant technical experts.

22.2.2.1.2 [lib.facet.num.get.virtuals]:

In paragraph 2, change the second list item to read:

-- Stage 2: Extract characters from `in` and determine a corresponding char value for the format expected by the conversion specification determined in stage 1.

Replace the second line of the code segment with:

```
char c = src[find(atoms, atoms + sizeof(src) - 1, ct) - atoms];
```

and add after the code example the following:

where the values `src` and `atoms` are defined as if by:

```
static const char src[] = "0123456789abcdefABCDEF+-";
char_type atoms[sizeof(src)];
use_facet<ctype<charT> >(loc).widen(src,
src + sizeof(src), atoms);
```

for this value of `loc`.

Requester: ncm@cantrip.org (Nathan Myers)
References:

Issue: CD2-22-003 num_put formatting error handling needs clarification
Section: 22.2.2.2.2 [lib.facet.num.put.virtuals]
Status: Open (US)
Description:

In 22.2.2.2.2 [lib.facet.num.put.virtuals], page 22-28, paragraph 21, the Draft reads:

If at any point `out.failed()` becomes true, then output is terminated.

However, the expression "out.failed()" is undefined, because the argument "out" has type OutputIterator, which is not required to have that member. The statement noted is not necessary, because if out is an instance of ostreambuf_iterator, then subsequent operations on it will have no effect, and the error will be detected by `operator<<` after `put()` returns.

Proposed Resolution:

In 22.2.2.2.2 [lib.facet.num.put.virtuals], page 22-28, paragraph 21, strike the sentence quoted.

Requester: ncm@cantrip.org (Nathan Myers)

References:

Issue: CD2-22-004 Interfacet dependency requirements inconsistent

Section: 22.1.1.1.1 [lib.locale.category]

Status: Open (US)

Description:

In 22.1.1.1.1 [lib.locale.category], page 22-6, paragraph 5 reads:

> For the facets num_get<> and num_put<> the implementation provided must depend only on the corresponding facets numpunct<> and ctype<>, instantiated on the same character type. Other facets are allowed to depend on any other facet that is part of a standard category.

This begs the question, if numpunct<> and ctype<> are allowed to depend on any other facet, doesn't this make num_get<> and num_put<> dependent on those facets as well?

The purpose for the restriction was to allow users to control numeric formatting and parsing easily by replacing facets numpunct<> and/or ctype<>. As it happens, none of the members functions of numpunct<> or ctype<> take any arguments from which a locale (and thus a facet) may be extracted, so they are perforce not dependent on any other facet.

Proposed Resolution:

Replace the above-quoted paragraph with:

> The provided implementation of members of facets `num_get<charT>` and `num_put<charT>` calls use_facet<*F*>(*l*) only for facet *F* of types numpunct<charT> and `ctype<charT>`, and for locale *l* the value obtained by calling member `getloc()` on the ios_base& argument to these functions.

Requester: Angelika Langer <langer@camelot.de>

References:

Issue: CD2-22-005 locale template constructor unusable

Section: 22.1.1 [lib.locale], 22.1.1.2 [lib.locale.cons]

Status: Open (US)

Description:

The language offers no syntax to allow calling of the locale constructor template

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

It seems unlikely that the necessary syntax will be added at this late date, despite its apparent utility. Without it, the member is uncallable.

A "factory" function cab be provided instead, with a similar interface, that can be called with legal syntax.

Proposed Resolution:

Eliminate from [lib.locale] and [lib.locale.cons] the locale constructor template

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

Replace it with a member function declared in the synopsis and among the locale member functions, as follows:

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

defined to yield the same locale value, where `*this` corresponds to argument `one` in the replaced constructor description.

Requester: Nathan Myers < ncm@cantrip.org>
References: lib-5519

Issue: CD2-22-006 locale codecvt_byname<> lacking member do_unshift()
Section: 22.2.1.6 [lib.locale.codecvt.byname]
Status: Open (US)
Description:

In Kona the locale facet codecvt<> got new members, `unshift()` and `do_unshift()`, to allow filebuf to cope with encodings that use a locking shift state. codecvt_byname<> should have all the same virtual members as codecvt<>, but this was missed in drafting. The oversight should be corrected.

Proposed Resolution:

Add to the class declaration for codecvt_byname<> in 22.2.1.6 [lib.locale.codecvt.byname] the member declaration:

```
  result do_unshift(stateT& state,
    externT* to, externT* to_limit, externT*& to_next) const;
```

Requester: Nathan Myers < ncm@cantrip.org>
References: lib-5520

Issue: CD2-22-007 global const functions in locale
Section: 22.1 Locales [lib.locales]
Status: Open (US)
Description:

The declarations of the non-member functions 'is*()' are declared to be 'const'. Although gcc extension allows this, I don't think that it is sanctioned by the remainder of the current CD.

Proposed Resolution:

Remove the final "const" from each of the declarations is*() in the synopsis and the description.

Requester: Dietmar Kuehl
References: Public comment 30/Kuehl

Issue: CD2-22-008 use_facet and has_facit incorrectly described as member functions
Section: 22.1.1 Class locale [lib.locale]
Status: Open (US)
Description:

It is stated that 'use_facet' and 'has_facet' are member functions. This does not match the later definition of those two functions as non-member function templates.

Proposed Resolution:

Remove the words that imply these are members.

Requester: Dietmar Kuehl
References: Public comment 30/Kuehl

Issue: CD2-22-009 cerberos object incorrectly constructed
Section: 22.1.1 Class locale [lib.locale]
Status: Open (US)
Description:

In the example, the object 'cerberos' of type basic_ostream<...>::sentry' is constructed with a default argument but there is no default constructor for this type. Instead, it has to be constructed like

```
typename basic_ostream<charT, traits>::sentry cerberos(s);
```

The same situation appears in other example, too.

Proposed Resolution:

In sample code in Clause 27 that constructs sentry objects, pass the appropriate stream to the constructor.

Requester: Dietmar Kuehl
References: Public comment 30/Kuehl

Issue: CD2-22-010
Section: 22.1.1.1.2 Class locale::facet [lib.locale.facet]
Status: Open (US)
Description:

It is missing in the definition of the static member ’id’ that this member has to be either publically accessible or at least accessible to the class ’locale’. As stated, it would be legal to make the member ’private’ which would not satisfy the intend (I think...).

Proposed Resolution:

Insert the word “public” in the description of the required member id.

Requester: Dietmar Kuehl
References: Public comment 30/Kuehl

Issue: CD2-22-011
Section: 22.1.1.1.2 Class locale::facet [lib.locale.facet]
Status: Open (US)
Description:

If 'refs == 0', does this imply that the 'locale' is supposed to delete the 'facet'? If this is the case, state that the 'facet' has to be a valid argument to 'delete' (or whatever) like it is done for the pointer managed by 'auto_ptr'.

If 'refs != 0', it is stated that the 'facet' is "deleted". This assumes that it is allocated by 'new' but I guess that the intent was to have the 'facet' be e.g. an object with static linkage: This would mean that "deleted" should be replaced by "destructed".

Proposed Resolution:

Replace the descriptions of the two cases as follows: for refs == 0, the implementation performs `delete static_cast<locale::facet*>(f)`, for f a pointer to the facet, when the last locale object containing it is destroyed; for refs == 1, the implementation never destroys the facet.

Requester: Dietmar Kuehl
References: Public comment 30/Kuehl

Issue: CD2-22-012
Section: 22.1.1.2 locale constructors and destructor [lib.locale.cons]
Status: Open (US)
Description:

It is stated at several points that the locale has a name if some conditions are given at construction time. However, it is not clear what this name should be. Is this intentional?

Proposed Resolution:

In the description of member locale::name(), add the statement: If *this has a name, then locale(name()) is equivalent to *this. Details of the contents of the resulting string are otherwise implementation-defined.

Requester: Dietmar Kuehl
References: Public comment 30/Kuehl

Issue: CD2-22-013
Section: 22.1.2 locale globals [lib.locale.global.templates]
Status: Open (US)
Description:

In the "Throws" section 'this' is mentioned. This is rather strange for a global function. It should probably be replaced by 'loc'.

Proposed Resolution:

Replace the text “*this” with “ loc”.
Requester: Dietmar Kuehl
References: Public comment 30/Kuehl

Issue: **CD2-22-014**
Section: 22.2.1.1 Template class ctype [lib.locale.ctype]
Status: Open (US)
Description:

For some of the functions arguments are not named. This is no problem most of the time, just inconsistent. However, for the description of '`toupper()`' I think it is an error. The [not named] argument is referenced in the description...

Proposed Resolution:

Add the referenced argument names to the prototype.

Requester: Dietmar Kuehl
References: Public comment 30/Kuehl

Issue: **CD2-22-015**
Section: 22.2.2.1.2 num_get virtual functions [lib.facet.num.get.virtuals]
Status: Open (US)
Description:

It is stated that the operation occurs in *two* stages. This statement is immediately followed by a description of *three* stages...

Proposed Resolution:

Change the remark to mention three stages.

Requester: Dietmar Kuehl
References: Public comment 30/Kuehl

Issue: **CD2-22-016**
Section: 22.2.2.1.2 num_get virtual functions [lib.facet.num.get.virtuals]
Status: Open (US)
Description:

The description of stage 2 ends with "If the character is *neither* discarded *nor* accumulated then `in` is advanced by `++in` and processing returns to the beginning of stage 2." I think this is exactly the negation of the intended wording, i.e. this should become: "If the character is *either* discarded *or* accumulated then `in` is advanced by `++in` and processing returns to the beginning of stage 2." I'm not 100% sure since I'm not a native English speaker...

Proposed Resolution:

Change the description as suggested.

Requester: Dietmar Kuehl
References: Public comment 30/Kuehl

Issue: **CD2-22-017**
Section: 22.2.3.1.2 numpunct virtual functions [lib.facet.numpunct.virtuals]
Status: Open (US)
Description:

In 'do_decimal_pointer()', 'do_thousends_sep()', 'do_truename()', and 'do_falsename()' objects of type 'char' are returned as 'char_type'. I think the objects returned have to be the results of 'widen()', e.g. using 'use_facet<ctype<char_type> >(locale::global())' or the same facet from a 'locale' passed as argument.

Discussion:

The only cases where the literals are used are in the instantiations for char and wchar_t. The encoding for the base class implementations is the native encoding, so no locale involvement is required.

Proposed Resolution:

In the descriptions, replace character literals of the form “`'.'`” with “`'.'` or `L'.'`”, and string literals of the form ‘`"true"`’ with ‘`"true"` or `L"true"`’.

Requester: Dietmar Kuehl

References: Public comment 30/Kuehl

Issue: CD2-22-020 facet "deleted" should be "destructed"
Section: 22.1.1.1.2 class locale::facet (lib.locale.facet) Section 2.
Status: Open (US)
Description:

If 'refs != 0', it is stated the the 'facet' is "deleted". This assumes that it is allocated by 'new' but I guess that the intent was to have the 'facet' be e.g. an object with static linkage: This would mean that "deleted" should be replaced by "destructed".

Requester: Dietmar Kuehl
References: Public comment 30/Kuehl

Issue: CD2-22-021 money_base::part not specified
Section: 22.2.6.3 [lib.locale.moneypunct]
Status: Open
Description:

```
> [lib.locale.moneypunct] defines:
>  class moneybase {
>    public:
>      enum part {none,space,symbol,sign,value};
>      struct pattern { char field[4] };
>  };
>
> The moneybase class isn't explained anywhere in the draft.
>
```

** Discussion:
The types pattern and part are used to explain moneypunct::do_pos_format() and moneypunct::do_neg_format(),money_put::do_put() and money_get::do_get(). However, the semantics of the enum values are mostly in the open, according to the draft.

The draft doesn't say that in the position specified for "symbol",money_put inserts the string returned by moneypunct<>::curr_symbol(). A malicious implementer could just use "DM" all the time and still conform. It also doesn't say that sign is taken for moneypunct<>::positive/negative_sign(). Is does not say what the difference between none and space is. (There does not seem to be any.) It also does not say what value is, e.g. how it is formatted. (Does it take information from the numeric facets for formatting and parsing of the value?)

There seems to be a lot of stuff implied though ...

Proposed Resolution:
Requester: Klaus Kreft & Angelika Langer

Issue: CD2-22-022 Relationship between moneypunct::pattern and moneypunct::part
Section: 22.2.6.3 [lib.locale.moneypunct]
Status: Open
Description:

Why is the type "pattern" in money_base an array of "char" instead an array of type "parts"? The intended use of pattern seemingly is to store parts.

** Discussion: It's an efficiency hack. As a struct containing an array of four chars, it can be passed in a register on many machines.

Proposed Resolution:

Specify the relationship between moneypunct::pattern and moneypunct::part, and state that there is an implicit requirement to moneypunct::part: it must fit into a char.

Requester: Klaus Kreft & Angelika Langer

Issue: CD2-22-023 Specification of do_unshift() incomplete
Section: 22.2.1.5.2 [lib.locale.codecvt.virtuals]

Status: Open

Description:

do_unshift does not specify what the value to_next will be on return. It also does not say that no more than (to_limit-to) characters are placed into the output.

Proposed Resolution:

Requester: Klaus Kreft & Angelika Langer

Issue: CD2-22-024 Confusion about return value of codecvt<>::do_always_noconv()

Section: 22.2.1.5 [lib.locale.codecvt]

Status: Open

Description:

codecvt<char,char,mbstate_t>::do_always_noconv() returns true in all cases.
All other instantiations of the codecvt template are said to return false. Is that true? It would mean that one cannot implement a non-converting wide character conversion facet of type codecvt <wchar_t, wchar_t, mbstate_t>, say for a Unicode file stream buffer that does not convert because the internal character encoding in iostreams already is Unicode.

I think the requirement of "others return false" is too restrictive and not necessary.

** Discussion:
This is a bit of confusion. At the time it was written there were three required base instantiations for that facet, and one returned true and the other two returned false. The requirement is not meant to imply anything about specializations other than those required in the table.

Proposed Resolution:

Requester: Klaus Kreft & Angelika Langer

Issue: CD2-22-025 Effects of ctype::do_is() difficult to understand

Section: 22.2.1.1.2 [lib.locale.ctype.virtuals]

Status: Open

Description:

Klaus and I came to an entirely different understanding of the functionality of ctype::do_is(), from just reading the draft.

The draft says (in the effects section): For each argument character, identifies a value M of type ctype_base::mask. The second form places M for all *p where (low<=p && p<high), into vec[p-low].

Okay, Klaus felt that is means: Places those classifications m from ctype_base::mask where all elements form the range [low,high) conform to m into vec.

I disagree and think it classifies each character in the array [low,high), i.e. for each character it determines a value of type ctype_base::mask, which can be a combination of bits, and places these mask values into an array called vec. Consequently, vec would have as many elements as the array [low, high) has, and there is one mask value in vec for each character in [low, high).

Whatever the intended meaning of the description of do_is() is, the description could be more understandable. :-)

** Discussion: Well, the vector form doesn’t take an ’m’ argument. I agree it should say "for each *p" rather than "for all *p".

Proposed Resolution:

Requester: Klaus Kreft & Angelika Langer

Issue: CD2-22-026 Derivation for moneypunct

Section: 22.2.6 [lib.category.monetary]

Status: Open

Description:

Why is moneypunct derived from moneybase, but money_put and money_get aren't?

They do not use the enums from moneybase in their interface, but they use the values internally, and one might wnat to extend the interfaces and use the values then.

** Discussion: You're right, but it's too late for that.

Proposed Resolution:
Requester: Klaus Kreft & Angelika Langer

Issue: CD2-22-027 Effect of negative values in do_frac_digits()
Section: 22.2.6.3.2[lib.moneypunct.virtuals]
Status: Open
Description:

Why is the return type of do_frac_digits() an int? What is the effect of negative values? Why isn't it unsigned?

** Discussion: It's anint in C. I don't know of any portable use for negative values.

Proposed Resolution:
Requester: Klaus Kreft & Angelika Langer

Issue: CD2-22-028 International currency symbol in moneypunct, mony_put, and money_get
Section: 22.2.6 [lib.category.monetary]
Status: Open
Description:

The moneypunct, mony_put, and money_get facets seem to be inconsistent in that some of them have template parameter and a static constant value for the international currency symbol, and some have not. money_put has no template parameter, money_get does not store the value. Is that intentional?

Why is it a template parameter at all? why don't the facets simply contain the international currency symbol and the ordinary one and let the using application decide which one to choose. This way a lot of stuff is duplicated that need to be duplicated. Or does the Intl template parameter have any impact on other parts of the money facets that I am not aware of?

** Discussion: I agree. I tried to change this bit awhile ago, but it was rejected. It would require some pretty big changes, big enough to delay the standard, maybe.

Proposed Resolution:
Requester: Klaus Kreft & Angelika Langer

Issue: CD2-22-029 Non-const ios_base& argument to the put() and get() functions of facets
Section: 22 [lib.localization]
Status: Open
Description:

Why is the ios_base& argument to the put() and get() functions of facets a non-const reference? Do these functions change the formatting information or the locale or anything in the stream base?

** Discussion: No good reason. They probably could be const, though member functions ofios_base are generally not const, and would be inaccessible. Originally, it was intended that errors would be reported by setting error state in the ios_base argument; so the real reason is "historical". It may be useful for user-derived facets to be able to call non-const members, though I don't know what for.

Proposed Resolution:
Requester: Klaus Kreft & Angelika Langer

Issue: CD2-22-030 Semantics of money_base::none and money_base::space
Section: 22.2.6.4 [lib.locale.moneypunct]
Status: Open
Description:

What is the difference between money_base::none and money_base::space? From the description in num_get and mnum_put the seem to be equivalent. If so, why are there two different values with the same semantics?

** Discussion: I agree that it doesn't say that "space" means put a space, and none" means put none, and it should. money_get treats both the same, because it will eat whitespace where either appears.

Proposed Resolution:
Requester: Klaus Kreft & Angelika Langer

Issue: CD2-22-031 Unspecified error reporting in money_get::do_get()
Section: 22.2.6.1.2 [lib.locale.money.get.virtuals]
Status: Open
Description:

The specification does not say anything about the error state. Say, it eats dozens of whitespaces and never finds a digit or whatever it is looking for. What's the return in that case?

** Discussion: You're right: it doesn't say anything about the error state, and it needs to say the same thing as the other facets. Of course if no value is stored it should be set to "fail".

Proposed Resolution:
Requester: Klaus Kreft & Angelika Langer

Issue: CD2-22-032 Confusing footnote in do_grouping()
Section: 22.2.3.1.2 [lib.facet.numpunct.virtuals]
Status: Open
Description:

The footnote 219 in [lib.facet.numpunct.virtuals] is confusing. Why does it talk of 51 character in a group? The string „\003" specifies a group size of 3 for all groups, and that's exactly what is described in the Returns section of do_grouping().

Proposed Resolution:

Strike the footnote!

Requester: Klaus Kreft & Angelika Langer

Issue: CD2-22-033 typos in money_put::do_put()
Section: 22.2.6.2.2 [lib.locale.money.put.virtuals]
Status: Open
Description:

There are occurrences of moneypunch (supposedly: moneypunct) and ios_type (supposedly: ios_base), both of which are not defined.

Proposed Resolution:
Requester: Klaus Kreft & Angelika Langer

Issue: CD2-22-034 time_put<>::put() incompletely specified
Section: 22.2.5.3.1 [lib.locale.time.put.members]
Status: Open
Description:

The definition of the time_put::put() function does not specify sufficiently how the interpretation of format specifiers is done, e.g. whether the original input string still exists after narrowing the characters in the format specification for comparison against '%'.

It also does not say whether or which characters are written out. A put() function not producing any output would conform to the standard the way it is specified at present.

Proposed Resolution:
Requester: Klaus Kreft & Angelika Langer

Clause 23 – Containers

Issue: **CD2-23-005 Library containers lack exception policy**
Section: 23 [lib.containers]
Status: Open (US)
Description:

As part of the interface to the standard container templates, users supply classes with member functions that are called during operation of the containers. These include particularly default and copy construction, assignment, and destruction. Iterator arguments provide arithmetic and dereference operations. Comparison arguments provide operator().

Any of these operations may, in general, result in an exception that propagates out of a container member. If, when this occurs, the container is left in an inconsistent state, it may be unable to satisfy its requirements on subsequent operations (including destruction), which will cause the program to display undefined behavior (crash, if you're lucky; give wrong answers, if not).

Requiring, for defined behavior, that these operations never throw exceptions is entirely impractical. At least, this would forbid placing most standard library objects in a container, because they are allowed to throw exceptions at unspecified times. More to the point, exceptions are a response to unexpected events, and to try to eliminate exceptions is to pretend to eliminate unexpected events. We have exceptions in the language because unexpected events cannot in general be eliminated.

Requiring that the standard containers satisfy their requirements regardless of exceptions appears to be equally impractical. It would impose prohibitive performance and storage overhead on the containers with only occasional benefit, when recovering from errors.

As it stands, the Standard Library containers are not compatible with exceptions: a program that uses both (intentionally or not) is undefined, and likely to crash. We should not deliver the Draft in this condition.

A resolution for this issue might identify

1. which container operations are subject to disruption, and what operations remain defined after such disruption;
2. when template argument operations can throw without causing undefined behavior;

This resolution might impose restrictions such as leaving the effect undefined if a container element destructor throws; and it might impose performance overheads such as requiring that a vector resize operation be implemented such that it (temporarily) requires storage for two copies of each element.

Proposed Resolution:

Prefer the direction of the solutions described in 97-0019/N1057.

Requester: Nathan Myers <ncm@cantrip.org>
References: 97-0019/N1057

Issue: **CD2-23-006 Impact of insert() and erase() on iterators not specified**
Section: 23.1.2 [lib.associative.reqmts]
Status: Open (US)
Description:

The original documentation for the HP STL ("The Standard Template Library" by Stepanov and Lee, technical report HPL-95-11(R.1)) contains the following sentence: "insert does not affect the validity of iterators and references to the container, and erase invalidates only the iterators and references to the erased elements."

Unfortunately, I can't find that sentence anywhere in CD2. It should go somewhere in 23.1.2.

As far as I can tell, this sentence was omitted accidentally; I don't think that anyone ever voted to remove it. (Also, so far as I know, all existing STL implementations do satisfy that guarantee.)

Proposed Resolution:

Add the following text to 23.1.2 after the Associative container requirements table:

The insert members shall not affect the validity of iterators and references to the container, and the erase members shall invalidate only iterators and references to the erased elements.

Yes

Requester: Matt Austern <austern@isolde.mti.sgi.com>
References: lib-5522, CD2-23-011
Public comment 32/Aldridge

Issue: **CD2-23-007 Can library containers be instantiated on incomplete types?**
Section: 23.1 Container requirements [lib.container.requirements]
Status: Open (US)
Description:

The issue is the interaction of template instantiation and partially defined classes. Consider the following example:

```
#include <list.h>

struct S
        {
        int a;
        list<S> b;
        };
```

Is this meant to be legal C++? The answer depends on whether the expansion of the list template tries to allocate a field of type S in the class list<S>. If so, it would violate paragraph 9.2.8 which states that non-static members of a class must be objects of previously defined classes. However, I couldn't find anything in the draft standard that states that list<S> may or may not expand into a class with a field of type S.

Please specify the behavior of definitions of all container templates (list, vector, etc.) in the standard library with respect to template parameters that are partially defined.

Proposed Resolution:

In 17.3.3.6 [lib.res.on.functions], add an item to the bullet list of cases causing undefined behavior:

-- if an incomplete type ([basic.types]) is used as a template argument when instantiating a template component.

Requester: Dr. Waldemar Horwat
References: Public comment #15/Horwat

Issue: **CD2-23-009 lifetime of iterators**
Section:
Status: Open (US)
Description:

The draft makes no statement about whether or not pointers/references remain valid DURING (not after) vector::insert. Since the value being inserted is a reference to const object, it is unclear whether or not you can insert an element of a vector into another location of that vector. For example (recalling from p. 23-5 Table 69 that push_back is defined in terms of insert):

```
vector<int> v(100);
v.push_back(v[0]);  // is this well defined?
```

The library implementations that I have seen do accommodate the code above because (when capacity must be increased) they fill-in the new memory region completely before destroying the objects in the original memory. I would suggest that the committee require that references into the vector remain valid during (but not after) the insertion. If such a restriction is not imposed, I would suggest that the standard explicitly say that code like the example above is undefined.

This issue applies to deque and string as well as vector. We believe the first two signatures of insert should be made to work as expected, and that the third (ie. range) can probably be made to work.

Post-Nashua Discussion:

The issue also applies to list, multimap, and multiset. It may apply to map and set in that such an operation is quite possibly the result of programmer misunderstanding or error.

Proposed Resolution:

In the 23.1.1 [lib.sequence.reqmts] Sequence requirements table entry for `a.insert(p,i,j)`, add a precondition:

pre: `[i,j)` are not iterators to `a`.

In the 23.1.3 [lib.associative.reqmts] Associative container requirements table entry for `a.insert(i,j)`, add a precondition:

pre: `[i,j)` are not iterators to `a`.

Requester: Bill Dimm
References: Public comment 21/Dimm
CD2-23-008

Issue: CD2-23-010 Is it possible to reclaim storage from a vector?
Section: 23.2.4 [lib.vector]
Status: Open (US)
Description:

I think that at least `clear()` should be defined to give the user some explicit control over memory leakage. Preferably, the function `compact()` would be added (or `v1.resize(v1.size())` defined to do the same) and the assignment behaviour specified as above. The container classes' definitions are careful to give time complexity guarantees without which they would not be usable in many situations. I think that some minimum guarantees on space complexity are also required

Proposed Resolution:

Discussion in Nashua concluded:

- don't change clear()
- don't change resize()
- don't change reserve()
- so we should add a new function

Requester: Brian Parker
References: Public Comment 25/Parker

Issue: CD2-23-011 Lifetime of iterators and references into containers
Section:
Status: Open (US)
Description:

The committee draft seems deficient in the statements it makes about the validity of iterators and references into STL containers.

In particular, I'd expected to find a statement such as:

> Unless otherwise stated (either explicitly or by defining a function in terms of the application of other functions), invoking a member function of a container or passing a container as argument to a container library function will not cause references or iterators to that container to become invalid.

Proposed Resolution:

Add text to 23.1 to this effect.

Requester: John Aldridge
References: Public comment 32/Aldridge
23-006

Issue: CD2-23-012 Container insert will not accept const iterator
Section: 23
Status: Open
Description:

Right now, the forms of container::insert() in the library that accept iterators specifying position will not accept const_iterators. That makes some very useful constructs impossible.

Proposed resolution:

All forms of insert that use iterators to specify position should be changed to accept const_iterators. e.g.

```
insert(const_iterator pos, const T&);
```

Requester:
References:

Issue: **CD2-23-013 Container requirements do not talk about allocator interface**
Section: 23
Status: Open
Description:

The interface for allocator<T> is not necessarily the same as that for allocator<S> when S != T. The CD does not specify which interfaces are allowed for the allocator template argument of containers. For example, is the following allowed:

```
List<T, allocator<double> > l;
```

Is it required to work?

Proposed resolution:
Requester: Jerry Schwarz
References:

Issue: **CD2-23-014 Conversion of container iterators to const_iterators**
Section: 23
Status: Open (US)
Description:

The WP currently does not guarantee that container iterators can be converted to const_iterators.

Proposed resolution:

Add to the assertion column of row 4 of the Container Requirements table, theX::iterator row:

Convertible to `X::const_iterator`

Requester:
References:

Issue: **CD2-23-015 Is X::reverse_iterator convertible to X::const_reverse_iterator?**
Section: 23
Status: Open
Description:

If X is a containers, then X::iterator is (or ought to be) convertible to X::const_iterator. X::reverse_iterator and X::const_reverse_iterator, however, are typedefs that involve the reverse_iterator<> template, so whether or not this guarantee exists for X::reverse_iterator and X::const_reverse_iterator depends on how that template is defined.

In fact, that conversion does not exist. reverse_iterator<X::iterator> and reverse_iterator<X::const_iterator> are unrelated types, and reverse_iterator<> has no member function that would provide a conversion between them.

Proposed resolution:

Add a member template "generalized copy constructor" to reverse_iterator<>, so that reverse_iterator<U> is convertible to reverse_iterator<V> if and only if U is convertible to V.

(Generalized copy constructors are used elsewhere in the standard library; see pair<>, for example.)

Issue: **CD2-23-016 Syntactically incorrect call to bitset::to_string**
Section: 23.3.5.3 bitset operators [lib.bitset.operators]
Status: Open (US)
Description:

On the last line of clause 23 [lib.containers] it says that operator<< applied to bitset<N>:

Returns:

```
os << x.to_string<charT,traits,allocator<charT> >()
```

But this is not valid C++ syntax. I believe this should be:

Returns:

```
os << x.template to_string<charT,traits,allocator<charT> >()
```

I think the fix is editorial, but I won’t argue with Andy.

Proposed Resolution:
Requester: Nathan Myers
References: lib-5305

Issue: **CD2-23-017 Illegal default template argument in vector<bool>**
Section: [lib.vector.bool] 23.2.5 Class vector<bool>
Status: Open
Description:

Template vector<bool, typename> is a partial specialization of vector<typename, typename>. Subsection [lib.vector.bool] says:

```
template <class Allocator = allocator<bool> >
class vector<bool, Allocator> {
```

However, default template arguments are not allowed on partial specializations.

Proposed Resolution:

Steve Rumsby in Message c++std-lib-5565:

True. Partial specialisations pick up the defaults of the primary template, so the fix here is to simply remove the default.

Requester: Daveed Vandervorde
References: lib-5564, 5565

Issue: **CD2-23-018 Container member typedefs unimplementable**
Section: [lib.deque] 23.2.1, [lib.list] 23.2.2, [lib.vector] 23.2.4, [lib.map] 23.3.1, [lib.multimap] 23.3.2, [lib.set] 23.3.3, [lib.multiset] 23.3.4
Status: Open
Description:

>> Jerry Schwarz (jerry@intrinsa.com) :
>> >I notice that the default allocator for map's is allocator<T> (where T is
>> >the value type) rather than
>> >
>> > allocator<pair<Key,T> >
>> >
>> >which is what I would have expected. Does this matter?
>>
>> I don't think it matters. vector<>, and perhaps deque<>, are
>> the only containers that have any use for an allocator<T>.
>> map<> can't actually use an allocator of pairs, either.

I was wrong. This does matter, but not for the Allocator type being useful as is; rather, the Container member typedefs, e.g.

```
typedef typename Allocator::reference reference;
```

are all wrong. We must change these to

```
typedef typename Allocator::rebind< U >::other::reference reference;
```

for appropriate type "U" in each case. Without this change the spec is inconsistent and unimplementable.

(Presumably the same applies to `const_reference`.)

Proposed Resolution:
Requester: Nathan Myers <ncm@cantrip.org>
References: lib-5508

Issue: **CD2-23-019 Unexpected constructor overload resolution**
Section:
Status: Open
Description:

I ran into a problem with vector constructors when using a compiler with member function template support. According to CD2 vector has the following constructors:

```
explicit vector(size_type n, const T& value = T(),
            const  Allocator& = Allocator());

template <class  InputIterator>
vector (InputIterator first, InputIterator last,
       const  Allocator& = Allocator());
```

When I create a vector<int> v2(1,0) the compiler is matching on the member function vector (InputIterator first, InputIterator last, ...). I had not expected this. In my environment size_t is an unsigned long so vector (InputIterator first, InputIterator last ...) does seem the better match.

I noticed a similar thing in basic_string but haven't checked any other containers yet.

P.J. Plauger comments: … [the above] are a few of many similar ambiguities, or near ambiguities, in the library. They aren't even the worst ones. I had to drop one member template from each of list, deque, and vector. It will stay dropped, too, until someone can tell me how to disambiguate:

```
template<class InputIterator>
       void assign(InputIterator first, InputIterator last;
template<class Size, class T>
       void assign(Size n, const T& t = T());
```

Beman Dawes comments: The problem arises in at least these cases:

basic_string<>:	constructor, append, assign, insert, replace.
deque<>:	constructor, assign (2 cases), insert.
list<>:	constructor, assign (2 cases), insert.
vector<>:	constructor, assign (2 cases), insert.
vector<bool>:	constructor, assign (2 cases), insert.

map<>, multimap<>, set<>, multiset<> have an insert range signature which should be kept consistent with other container's insert range signatures, even though there is less chance of ambiguity in the associative containers.

Proposed Resolution:

Requester: Sandra Whitman <whitman@zko.dec.com>
References: lib-5581, 5582, 5583, 5584, 5585, 5586

Issue: CD2-23-020 list<>::merge specification incomplete
Section: 23.2.2.4 [lib.list.ops]
Status: Open
Description:

The documentation in the standard reads as follows:

```
void   merge(list<T,Allocator>& x);
template <class Compare> void merge(list<T,Allocator>& x, Compare
comp);
```

Requires: comp defines a strict weak ordering (25.3), and the list and the argument list are both sorted according to this ordering.
Effects: Merges the argument list into the list.
Notes: Stable: for equivalent elements in the two lists, the elements from the list always precede the elements from the argument list. x is empty after the merge.
Complexity: At most size() + x.size() - 1 comparisons.

This seems somewhat inadequate. For example, does the first version of merge require that the less<T> is a strict weak ordering? Does either version of merge even make use of a comparison? What for? It seems to me that an implementation could splice the argument list onto the end of the list and still conform.

Proposed Resolution:

Add the follow sentence (taken from 25.3.4, suitably modified) to the above effects clause:

> The resulting list will be in non-decreasing order; that is, for every iterator i in [begin(), end()) other than first, the condition *i < *(i -1) or, respectively, comp(*i, *(i - 1)) will be false.

Requester: David Abrahams <abrahams@motu.com>
References: lib-5829, 5830, 5831, 5832, 5833, 5834

Issue: **CD2-23-021 Container assignment fails to specify copy of allocator**
Section: 23.1 [lib.container.requirements] paragraph 8
Status: Open
Description:

(23.1-8) reads as follows:

> Copy constructors for all container types defined in this clause copy the allocator argument from their respective first parameters. All other constructors for these container types take an Allocator& argument (20.1.5). A copy of this argument is used for any memory allocation performed, by these constructors and by all member functions, during the lifetime of each container object. ...

Shouldn't this also apply to assignment operators, in parallel to the way comparison objects work in (23.1.2-10) associative containers?

> When an associative container is constructed by passing a comparison object the container shall not store a pointer or reference to the passed object, even if that object is passed by reference. When an associative container is copied, either through a copy constructor or an assignment operator, the target container shall then use the comparison object from the container being copied, as if that comparison object had been passed to the target container in its constructor.

Proposed Resolution:
Requester: David Abrahams <abrahams@motu.com>
References: lib-5822

Clause 24 – Iterators

Issue: **CD2-24-003 Missing identity requirement for forward itertors**
Section: 24.1.3 Forward iterators [lib.forward.iterators]
Status: Open
Description:

The requirements for forward iterators is missing an object identity requirement such as:

> If a and b are dereferenceable, a == b if and only if &*a == &*b

Alex Stepanov: Two iterators are equal if and only if they point to the same object. (And it is part of C/C++ to view object identity as address identity.)

Nathan Myers: This is not a question of effects on the STL implementation, but on the values of pointers that have (temporarily) escaped from iterators into code outside the STL proper. Code that knows nothing about STL or iterators very properly relies on ordinary C++ semantics, including object identity, and if an STL component implicitly violates that, things can be expected not to work. How to express exactly the requirement on iterators, imposed by the core C++ object identity, is the topic under discussion.

Greg Colvin and Sean Corfield repeatedly expressed great concern that such a requirement would make writing iterators to disk based containers more difficult.

Beman Dawes: What worries me about not requiring &*x == &*y when x == y is silent failure of code like:

```
*x = *y;  // fails when operator= misses assignment to self
```

It seems to me iterators will get a bad name if code like the above fails silently. The problem is inside functions that take arguments by pointer or reference. Users of these functions will convert iterators to pointers by writing &*x or iterators to references by writing *x. Then the functions fail because they assume "sameness" is defined as pointer identity.

Current implementations (HP,etc.) rely on object identity, specifically operator&. Containers like vector<string> and list<vector<int> > fail without it.

Andy Koenig: Suppose we had a requirement that &*x == &*y when x == y.
Would that mean it would be impossible to have a forward iterator over a type with a private operator& member?

Dag Bruck: What operations in STL rely on object identity, specifically what operations in the current SGI implementation?

Matt Austern: Too many to count. Every algorithm that copies an iterator, assigns through the copy, and assumes that it has modified the object that the original iterator pointed to, relies on object identity. Since passing an argument by value uses copy semantics, every mutating algorithm that takes its iterator arguments by value (which is all of them) relies on this property. Unless you make that assumption, you have no reason to believe that reverse reverses, that sort sorts, and so on.

Proposed Resolution:

To the Forward iterator requirements table in 24.1.3 [lib.forward.iterators], change the portion of the entry for `*a` which currently reads:

`a == b` implies `*a == *b`.

to:

`a == b` implies `*a == *b` and `&*a == &*b`.

Requester: Beman Dawes <beman@esva.net>
References: lib-5620 and several dozen following that.

Clause 25 – Algorithms

Issue: **CD2-25-001 Rotate specified incorrectly?**
Section: [lib.alg.rotate] 25.2.10
Status: Open
Description:

Effects description uses the assignment:

```
*( first + i) = *( result + (i + ( middle - first)) % ( last - first))
```

Should that be changed to:

```
*(result + i) = *(first + (i + ( middle - first)) % ( last - first))
```

Proposed Resolution:
Requester: Editorial Box

Issue: **CD2-25-002 Random shuffle argument type conversion?**
Section: [lib.alg.random.shuffle] 25.2.11
Status: Open
Description:

Can it accept an argument that yields a result of a type that, although different from RandomAccessIterator::difference_type, can be converted to it?

Proposed Resolution:
Requester: Editorial Box

Issue: **CD2-25-003 lower_bound range of i?**
Section: [lib.lower.bound] 25.3.3.1
Status: Open
Description:

Should the range of i be changed from [first, i) to [first, last)?

Proposed Resolution:

Change the range of i from [first, i) to [first, last)?

Requester: Editorial Box

Issue: **CD2-25-004 upper_bound range of I?**
Section: [lib.upper.bound] 25.3.3.2
Status: Open
Description:

Should the range of i be changed from [first, i) to [first, last)?

Proposed Resolution:

Change the range of i from [first, i) to [first, last)?

Requester: Editorial Box

Clause 26 – Numerics

Issue: **CD2-26-002 Accumulate specification incorrect**
Section: 26.4.1 Accumulate [lib.accumulate]
Status: Open (US)
Description:

In 26.4.1 [lib.accumulate], the requirements for class T are not specified. The user is left wondering what properties class T has to have to work. For example, does class T have to allow assignment? Clearly there are (recursive) implementations of accumulate that would not require assignment. But are implementors required to handle the case where T does not allow assignment. The standard should specify exactly what properties class T has to have in order to work with the accumulate template.

There are other problems here also. There's no Returns: clause for accumulate. The type for the pseudo-variable "acc" is not specified.

Proposed Resolution:

In the Effects clause of 26.4.1, change:

> Initializes the accumulator acc…

to:

> Computes its result by initializing the accumulator acc…

Also, add the following to the Requires clause of 26.4.1 [lib.accumulate] and 26.4.2 [lib.inner.product]:

> T must meet the requirements of CopyConstructible (_lib.copyconstructible_)and Assignable (_lib.container.requirements_) types.

Requester: Arch Robison & David Nelson
References: Public comment 28/Robison/Nelson

Issue: **CD2-26-003 operator<< has unneeded ends**
Section: 26.2.6 complex non-member operations [lib.complex.ops]
Status: Open (US)
Description:

In 26.2.6 [lib.complex.ops], paragraph 15, `operator<<` inserts a NUL character when writing to an ostream stream. i.e., the "as if" code shown inserts an `ends`, which is retained by the result of `s.str()` used. Did you mean `s.c_str()` or should the `ends` not be appended? Surely the intent was not to insert NUL characters into output.

Proposed Resolution:

Remove the "`<< ends`".

Requester: Arch Robison & David Nelson
References: Public comment 28/Robison/Nelson

Clause 27 – Input/Output

Issue: **CD2-27-001 Incorrect post condition for ios_base::failure**
Section: 27.4.2.1.1 [lib.ios::failure]
Status: Open (US)
Description:

The problem that existed with the other exception classes still exists in ios_base::failure (Nov '96 WP [lib.ios::failure]):

```
explicit failure(const string& msg);
```

Effects: Constructs an object of class failure, initializing the base class with `exception(msg)`.

Postcondition: `what() == msg.str()`

Proposed Resolution:

Change the postcondition to:

Postcondition: `strcmp(what(), msg.c_str()) == 0`

Requester: Kevlin Henney <kevlin@two-sdg.demon.co.uk>
References:

Issue: **CD2-27-002 Assignment of stream undefined**
Section:
Status: Open (US)
Description:

The current WP does not specify if stream assignment or copy construction is valid, and if so what semantics it should have.

Jerry Schwarz comments "I'm pretty sure they (i.e. private copy constructor and assignment) were in basic_ios at some point. I don't remember an explicit decision to drop them. I know that there were always people who felt that was the wrong way to specify that copying wasn't allowed. It might have been done as an editorial change.

I'm pretty sure I never heard a deliberate decision allow copying."

Proposed Resolution:

Add to 27.4 (synopsis of basic_ios)

```
private:
    basic_ios(const basic_ios&) ; // exposition only
    basic_ios& operator=(const basic_ios&); // exposition only
```

And add to 27.4.5.1

> The copy constructor and assignment operator for basic_ios are declared but no specification is given here. Because they are declared private user programs cannot invoke them and classes derived from instances do not contain copy constructors or assignment.

Requester: Jerry Schwarz <jerry@intrinsa.com>
References: lib-5288 and responses

Issue: **CD2-27-003 Widening & narrowing of basic_ostream insertors & extractors undefined**
Section:
Status: Open (US)
Description:

What's the purpose of

```
template <class charT, class traits> basic_ostream<charT,traits>&
operator<<(basic_ostream<charT,traits>&out, const char* s);
```

I had expected that it would allow to insert a tiny character sequence, say a string literal, into a wide character output stream for instance. In other words, does the inserter widen the tiny characters? Is it an oversight that widening is not mentioned in [lib.ostream.inserters.character]?

How about the inserter for charT sequences?

```
template <class charT, class traits> basic_ostream<charT,traits>&
operator<<(basic_ostream<charT, traits>& out, const charT* s);
```

Does it widen the characters?

How about the inserter of a tiny character stream that inserts tiny character sequences?

```
template <class traits> basic_ostream<char, traits>&
operator<<(basic_ostream<char, traits>& out, const char* s);
```

Does it widen the characters, or not?

Do the corresponding extractors narrow the characters?

Proposed Resolution:

Widening and narrowing should not be done by operator<<, put, operator>> or get when inserting or extracting char's into or from streams whose character type is char.

Specifically this requires a change to 27.6.2.5.4 and possibly other locations.

Requester: Angelika Langer <langer@camelot.de>
References:

Issue: CD2-27-004 <iosfwd> missing some declarations
Section: 27.2 Forward declarations [lib.iostream.forward]
Status: Open
Description:

The default template arguments in <iosfwd> use some classes for which there are no forward declarations. For example, the basic_stringbuf declaration looks like:

```
template <class charT, class traits = char_traits<charT>,
             class Allocator = allocator<charT> >
class basic_stringbuf;
```

In order for this to be compiled, `char_traits` and `allocator` must be forward declared.

Proposed Resolution:

To the top of the list in Section 27.2 add:

```
template <class charT> class char_traits;
template <class T> class allocator;
```

Requester: Judy Ward <j_ward@decc.enet.dec.com>
References:

Issue: CD2-27-007 iso_base::failure::what missing throw()
Section: 27.4.2.1.1 Class ios_base::failure [lib.ios::failure]
Status: Open (US)
Description:

In 27.4.2.1.1 [lib.ios::failure], method `what()` has a more general exception specification than the method that it is overriding. 27.4.2.1.1 says that `std::ios_base::failure::what` can throw any exception. But 18.6.1 [lib.exception] says that `std::exception::what` cannot throw any exception. This clearly contradicts 15.4 [except.spec], paragraph 3, which requires that the overriding derived-class method throw only exceptions allowed by the base-class method.

Proposed Resolution:

Resolve the problem by adding a throw() to the declaration of ios_base::failure::what.

Requester: Arch Robison & David Nelson
References: Public comment 28/Robison/Nelson

Issue: CD2-27-008 Set eofbit on eof
Section: 27.6.1.3 Unformatted input functions [lib.istream.unformatted]
Status: Open (US)
Description:

In 27.6.1.3 paragraph 28, one would also expect eofbit to be set if end-of-file is encountered before n characters are stored.

Proposed Resolution:

Change the description to `set(failbit|eofbit)`, as requested.

Requester: Arch Robison & David Nelson
References: Public comment 28/Robison/Nelson

Issue: CD2-27-009 Incorrect return type from showmanyc()
Section: 27.8.1.4 Overridden virtual functions [lib.filebuf.virtuals]
27.5.2.4.3 Get area [lib.streambuf.virt.get]
Status: Open (US)
Description:

Considering section 27.8.1.4 paragraphs 1,2 and section 27.5.2.4.3 paragraphs 1-3. Should the return type of showmanyc be streamsize instead of int. Consider the case when streamsize is a 32 bit long int and int is 16 bits.

Proposed Resolution:

Change the return type of `showmanyc` as requested.

Requester: Arch Robison & David Nelson
References: Public comment 28/Robison/Nelson

Issue: CD2-27-010 ~basic_ostringstrema() not specified
Section: 27.7.3.1 basic_ostringstream constructors [lib.ostringstream.cons]
Status: Open (US)
Description:

~basic_ostringstream is not specified

Proposed Resolution:

Delete the declaration of `~basic_ostringstream`.

Requester: Angelika Langer

Issue: CD2-27-011 Inconsistent treatment of typedefs.
Section: 27
Status: Open (US)
Description:

Specifically. Some classes derived from basic_streambuf basic_stringbuf) inherit typedefs for char_type, int_type, pos_type, off_type and traits type. And others redeclare them. This should be done consistently

Proposed Resolution:

Make the changes described above.

Requester: Angelica Langer

Issue: CD2-27-012 Interconvertibility of streamoff with integral types
Section: 27.4.4 fpos requirements [lib.fpos.operations]
Status: Open (US)
Description:

Other places in the WP suggest that streamoff can be converted from and to integral types, and interconvertibility of streamoff and streampos is required here, but there is no mention of interconvertibility of streamoff and any integral type.

Proposed Resolution:

Add lines to table in 27.4.4 requiring streamsize(O) and O(streamsize) be allowed.

Requester: Angelica Langer

Issue: CD2-27-013 location of fpos is not stated.
Section: 27.4 Iostreams base classes [lib.iostream.base]

Status: Open (US)
Description:
27.4.3 contains a definition of template fpos, but no synopsis requires the definition of fpos to be included
Proposed Resolution:
Add fpos to the synopsis of <ios> in 27.4
Requester: Angelika Langer

Issue: CD2-27-014 Incorrect specification of get
Section: 27.6.1.3 Unformatted input functions [lib.istream.unformatted]
Status: Open (US)
Description:
Specifications of

```
basic_istream<charT,traits>& get(char_type* s, streamsize n)
```

and

```
basic_istream<charT,traits>&
get(basic_streambuf<char_type,traits>&)
```

refer to getline (variants with a delim parameter) rather than get.
Proposed resolution:
Change reference from getline to get.
Requester: Jerry Schwarz

Issue: CD2-27-015 Manipulators are described as only applicable to ostreams
Section: 27.6.3 Standard manipulators [lib.std.manip]
Status: Open (US)
Description:
Manipulators that modify ios_base should be usable with either input or output streams.
Proposed Resolution:
Add to specifications of restiosflags, setiosflags, setbase, setprecision and setw the pharse “and if `in` is a basic_istream then the expression in>>s ...” Also add that the result of the expression is the original stream.
Requester: Jerry Schwarz

Issue: CD2-27-017 Return clause of iword refers to int&
Section: 27.4.2.5 ios_base storage functions [lib.ios.base.storage]
Status: Open (US)
Description:
Proposed Resolution:
Replace with long&.
Requester: Tom Plum

Issue: CD2-27-020 Template arguments undefined
Section: 27.6.1.1 [lib.istream]
Status: Open (US)
Description:
charT is not defined in the following:

```
template<class traits>
   basic_istream<char,traits>&
operator>>(basic_istream<charT,traits>&,unsigned char*);
template<class traits>
   basic_istream<char,traits>&
operator>>(basic_istream<charT,traits>&,signed char*);
```

Proposed Resolution:
`charT` should be replaced by `char`.
Requester: Klaus Kreft & Angelika Langer <langer@camelot.de>

Issue: CD2-27-021 Constness of tellg() and operator bool() in istream
Section: 27.6.1.1 [lib.istream] and 27.6.1.1.2 [lib.istream::sentry]
Status: Open (US) [Incorrectly marked "active" instead of "open" in prior version of issues list]
Description:
Some functions, judged from their semantics, seem to be constant functions. Why are they defined as non-constant functions? The functions in question are:

```
pos_type tellg();                        in [lib.istream]
operator bool() { return _ok; }          in [lib.istream::sentry]
```

Proposed Resolution:

Make istream::sentry::bool and ostream::sentry::bool const functions. Do not make istream::tellg const.

** Reported by: Klaus Kreft & Angelika Langer <langer@camelot.de>

** Owner:

Issue: **CD2-27-023 Undefined e in description of seekoff**

Sections: 27.8.1.4 [lib.filebuf.virtuals]

Status: Open (US)

Description:

The description of seekoff() mentions an expression off*e, but e is not defined. Probably it is the constant returned by a_codecvt.encoding().

Proposed Resolution:

Replace “e” by “a_codecvt.encoding()”.

Requester: Klaus Kreft & Angelika Langer <langer@camelot.de>

Issue: **CD2-27-024 Open mode in seekoff()**

Sections: 27.8.1.4 [lib.filebuf.virtuals]

Status: Open (US)

Description:

The standard does not say anything about the purpose and effect of the open mode in filebuf's seekoff() function [lib.filebuf.virtuals]. I guess that something along the line of what is described for seekpos() is meant.

Proposed Resolution:

Change the description of seekpos to ignore the value of its openmode argument.

Requester: Klaus Kreft & Angelika Langer <langer@camelot.de>

Issue: **CD2-27-025 Return of rdstate()**

Sections: 27.4.5.3 [lib.iostate.flags]

Status: Open (US)

Description:

[lib.iostate.flags] says that rdstate() returns the control state of the stream buffer. I had expected it would return the stream state.

Proposed Resolution:

Change [lib.iostate.flags] to say that rdstate() returns the error state.

Requester: Klaus Kreft & Angelika Langer <langer@camelot.de>

Issue: **CD2-27-028 Return type & value of insertion or extraction involving a manipulator**

Sections: 27.6.3 [lib.std.manip]

Status: Open (US)

Description:

The text doesn't say what the type or value of an insertion or extraction involving a manipulator is. For out<<resetios(...) it should say that the expression has type ostream& and value out. Similarly in>>resetios(...) has type istream& and value in.

Proposed Resolution:

Make the changes as above:

Requester: Jerry Schwarz <jerry@intrinsa.com>

Issue: **CD2-27-031 Which character inserters for character arrays apply widening?**

Sections: 27.6.2.5.4 [lib.ostream.inserters.character]

Status: Open (US)

Description:

What's the purpose of :

```
template <class charT,class traits>
basic_ostream<charT,traits>&
operator<<(basic_ostream<charT,traits>& out, const char*
s);
```

I had expected that it would allow to insert a tiny character sequence, say a string literal, into a wide character output stream for instance. In other words, does the inserter widen the tiny characters? Is it an oversight that widening is not mentioned in [lib.ostream.inserters.character], right?

Proposed Resolution:

Specify that the inserter for arrays of char (to non-char stream) widen the chars.

Requester: Angelika Langer <langer@camelot.de>

Issue: **CD2-27-032 Which character inserters for single characters apply widening?**

Sections: 27.6.2.5.4 [lib.ostream.inserters.character], 27.6.1.2.3 [lib.istream::extractors]

Status: Open (US)

Description:

The description of the effects of the character inserters for single characters in [lib.ostream.inserters.character] says:

> In case c's type is char the character to be inserted is widen(c); otherwise the character is c.

A footnote adds:

> In case the insertion is into a char stream, widen(c) will usually be c.

I conclude that widen is always applied once a char is inserted into any kind of output stream (be it a stream for char, wchar_t, or a generic charT). On the other hand widen is never applied if a wchar_t or a generic charT is inserted. Why is this asymmetry? What is the purpose of widening a char even if is inserted into a char stream?

Discussion:

The purpose of widening a char even if is inserted into a char stream could be to allow for character encodings inside a stream that are different from the compiler's native encoding. If this were the intent then all inserters and extractors and all the unformatted operations would have to apply narrowing and widening. The draft does not require this, only in the case of the char inserter for char output streams.

I think a clarification is needed that says whether the encoding used inside the stream always has to be compiler's native encoding or not. If encodings different from the native encoding are allowed then it needs to be specified which operations apply widen/narrow. This would be a major change, and I doubt that there is a need for non-native character encodings inside a stream.

Proposed Resolution:

In [lib.ostream.inserters.character], change:

> In case c's type is (signed, unsigned or plain) char the character to be inserted is widen(c);

to:

> In case c has type `char` and the character type of the stream is not `char`, then the character to be inserted is widen(c);

Requester: Angelika Langer <langer@camelot.de>

Issue: **CD2-27-033 Read not testing stream state**

Section: 27.6.1.3 Unformatted input functions [lib.istream.unformatted]

Status: Open (US)

Description:

The specification of read does not test for the state of the stream.

Proposed resolution:

Add at the beginning of the description:

> If `!good()`, calls `setstate(failbit)`, which may throw an exception, and return.

Issue: **CD2-27-034 Should istream::sentry and ostream::sentry be copyable?**

Section:

Status: Open (US)

Description:

Should istream::sentry and ostream::sentry be copyable?

Proposed resolution:

No they shouldn't.

Requester: Jerry Schwarz

Issue: **CD2-27-035 Box 31**

Section: 27.3.1, 27.3.2
Status: Open (US)
Description: Box 31
Proposed solution:

Add to the beginning of 27.3.1 and 27.3.2.

"Except as noted below the state of the objects is unchanged from their initial state."

Issue: **CD2-27-36 Box 33**
Section: 27.4.2.7
Status: Open (US)
Description:

The fpos constructor should be explicit.

Proposed resolution:

Accept the proposed suggestion.

Issue: **CD2-27-037 Box 34**
Section: 27.6.2.5.3
Status: Open (US)
Description:

Action of insertor does not describe what happens on failure:

Proposed Resolution:

Add that:

"If failed is true call setstate(badbit), which may throw and exception and return."

Issue: **CD2-27-038 Box 37**
Section: 27.8.1.4
Status: Open (US)
Description:

The imbue function has no effects clause.

Proposed Resolution:

Add the following as the missing Effects clause:

Effects: Causes characters inserted or extracted after this call to be converted according to loc until another call of imbue.

Notes: This requires reconversion of previously converted characters. This, in turn, requires the implementation to be able to reconstruct the original contents of the file.

Issue: **CD2-27-039 Request to access the underlying file from a filebuf**
Section:
Status: Open (US)
Description:

The standard fstream classes are missing a key feature that most existing fstream classes have, namely the ability for users to access the association between an streambuf and the underlying C file descriptor/pointer.

For example, most iostream classes have these member functions to create or attach a C file to a C++ fstream:

```
filebuf::filebuf(int file_descriptor,...)
filebuf* filebuf::attach(int file_descriptor, ...);
```

Most existing iostream classes have a way to access the underlying C file descriptor or pointer given the fstream, i.e:

```
int filebuf::fd() const;
```

We think this functionality is essential for C++ users who need to work with other C library features, i.e. sockets, extensions to stdio for special file types, etc.

We understand that file descriptors are not in the C standard, but C FILE* pointers are included. So changing the above functions to accept or return C FILE pointers would be fine.

We also understand that this would require implementors to use the underlying C input/output libraries to implement iostreams. We don't know of any vendor who does not plan to do that anyway.

Alternatively, one could consider writing a stdiobuf class to encapsulate these conversion functions (to connect a streambuf to a FILE *), but it's probably too late for that.

Proposed Resolution:

Requester: Judy Ward <j_ward@decc.enet.dec.com>
References:

Issue: **CD2-27-040 impact of sync() on the get area**
Section: 27.5.2.4.2 [lib.streambuf.virt.buffer]
Status: Open
Description:

I don't understand the description of sync() in [lib.streambuf.virt.buffer].

Does sync() have an impact on the get area? From the description is sounds as though only the put area would be affected.

What is the purpose of sync()? My understanding was, although I'm not so sure anymore, that sync() synchronizes the buffer areas with the external device.

Discussion:
Jerry Schwarz <jerry@intrinsa.com>: That's the intention. But exactly what it means depends a lot on what the "external device" is, and since user's can define their own streambuf's it gets pretty vague.

My usual assumption is that since after synchronization all state should be in the external device, both the get and put areas should be empty after a sync. But this is not required. For a file this means that if the get area is non-empty before the sync the implementation needs to do a seek to move the file to the correct position. I'm not sure if anyone does that. Of course if the file isn't seekable you can't do anything. However you can still throw away any waiting characters, and it's common practice (especially on Microsoft OS's) to use sync to do so. Of course what people really want to do when they do this is throw away not just any characters in the get area, but also any that the OS might have accumulated. But this is certainly not required.

So the bottom line, is that beyond flushing output there really isn't much guaranteed for sync on a filebuf.

It might have been better if we had distinguished somehow between seekable and non-seekable files, but we didn't and I fear it's too late now

Proposed Resolution:
Add a description for seekable files and state that nothing can be said for non-seekable files.

My main concern was that I didn't want to invalidate existing practice (either in iostream classic or recent implementations of std::filebuf). I fear that if we make changes in the direction you're suggesting we would do that. Also I'm not completely sure what it is you want. So specific words for the definition of filebuf::sync would be appreciated.

Requester: Angelika Langer <langer@camelot.de>

Issue: **CD2-27-041 Relationship between narrow and wide stream objects is not specified**
Section: 27.3 [lib.iostream.objects]
Status: Open
Description:

Somebody recently asked me whether output to cout and wcout is supposed to produce any reasonable, readable result. I didn't have an answer. The draft says wcout is connected to stdout, but nothing about its relationship to cout. Are narrow stream objects and their wide stream counterparts required to be synchronized? The standard does not say anything about it. Is it implementation defined whether the output is garbled or not?

Proposed Resolution:
Requester: Klaus Kreft & Angelika Langer

Issue: CD2-27-042 Inserter for void * should have const argument
Section: [lib.ostream] 27.6.2.1 Template class basic_ostream
Status: Open
Description:

In [lib.ostream] we find:

```
basic_ostream<charT,traits>& operator<<(void* p);
```

I think this ought to be

```
basic_ostream<charT,traits>& operator<<(const void* p);
```

(or we can't print the values of const pointers).

Jerry Schwarz: I don't remember seeing this issue lately. I think the first time it came to my attention was around 1990. I thought it had been fixed a long time ago.

Steve Clamage: The lack of const on the void* inserter has been brought up time and again for years. I have never understood why it was never changed to const void*. I'd love to see it changed, or for someone to explain why it shouldn't be changed.

Proposed Resolution:
Change the signature for operator<< on void * in 27.6.2.1 and 27.6.2.5 as indicated above.

Requester: Bjarne Stroustrup
References: lib-5588, 5589, 5590

Issue: CD2-27-043 failure specification uses non-existent exception constructor.
Section: 27.4.2.1.1
Status: Open
Description:

One of our customers spotted a typo in Section 27.4.2.1.1:

The class failure defines the base class for the types of all objects thrown as exceptions, by functions in the iostreams library, to report errors detected during stream buffer operations.

```
explicit failure(const string& msg);
```

Effects: Constructs an object of class failure, initializing the base class with exception(msg).

Std::exception doesn't have a constructor which takes a conststring& argument.

Proposed Resolution:
Change the effects to:

Effects: Constructs an object of class failure.

Requester: Judy Ward <j_ward@zko.dec.com>

Issue: CD2-27-044 Missing fstream open modes
Section:
Status: Open
Description:

Document X3J16/96-0126 (WG21/N0944) proposed definitions for fstream open modes, and I believe the proposal was accepted as written. In particular, it specifies the following flag combinations for read/write access to fstreams:

stream flags	stdio equivalent
------------	----------------
in|out	"r+" -- file must already exist
in|out|trunc	"w+" -- file truncated to zero length
in|out|app	"a+" -- all writes forced to eof
in|out|binary	"r+b"
in|out|trunc|binary	"w+b"
in|out|app|binary	"a+b"

The two combinations

in|out|app	"a+"
in|out|app|binary	"a+b"

did not make it into the draft -- accidentally dropped, I assume. (I supplied the text to Andy, so it is probably my fault.) There is presently no way to get the stdio "a+" functionality with fstreams.

Can we add an issue to put back the missing forms? Or does someone recall that these forms were eliminated on purpose?

Proposed Resolution:

To Table 93 "File open modes" in section 27.8.1.3 "Member functions" add two rows as follows:

After the "w+" row add the row

0 + + 0 + "a+"

and after the "w+b" row add the row

\+ + + 0 + "a+b"

where in all cases "0" means a blank column.

Requester: Steve Clamage <Stephen.Clamage@Eng.Sun.COM>

Issue: CD2-27-045 basic_istream::getline() specification error

Section: 27.6.1.3 Unformatted input functions [lib.istream.unformatted]

Status: Open

Description:

The two argument get()

basic_istream<charT,traits>& get(char_type* s, streamsize n)

is define to call getline:

getline(s,n,widen('\n'));

This implies that get(s,n,'\n') differs from get(s,n) in that the latter removes '\n' from the input stream.

(1) Am I right?
(2) Was this intentional?

I believe this wasn't always the case and that get(s,n) used to mean get(s,n,widen('\n')). My conjecture is that someone did a cut & paste error when changing a single

basic_istream<charT,traits>& get(char_type* s, streamsize n, charT = traits::newline())

into the current

basic_istream<charT,traits>& get(char_type* s, streamsize n)
basic_istream<charT,traits>& get(char_type* s, streamsize n, charT c)

If The answer to (1) is yes, and the answer to 2 is no, we have a problem:

(3) Do we change it back so that get(s,n) means get(s,n,widen('\n'))?
(4) Is this on any NB list?

- Bjarne

PS Note that ignore() still relies on a default argument the way get() used to do.I consider that a good thing, but find the unevenness of style worrying and error-prone.

PJ Plauger comments: That's how it looks to me. I believe the definition should be get(s, n, widen('\n')).

Philippe Le Mouel comments: Yes, this is the way it was intended when we added the basic_ios member functions widen and narrow.

Jerry Schwarz comments: The history of this is, as usual somewhat convoluted. The problem is that defaults which used to be just '\n' (or the like) became widen('\n'). This can't be a default. (widen is a member function of the class)

Beman Dawes comments: There seems to be a similar cut-and-paste problem with `basic_istream<charT,traits>& get(basic_streambuf<char_type,traits>& sb);` The effects are currently specified as: Calls `getline(s,n,widen('\n'))`

Proposed Resolution:

Change paragraph 10 of 27.6.1.3 Unformatted input functions [lib.istream.unformatted] from:

Effects: `Calls getline(s,n,widen('\n'))`

to:

Effects: `Calls get(s,n,widen('\n'))`

Change paragraph 15 of 27.6.1.3 Unformatted input functions [lib.istream.unformatted] from:

Effects: `Calls getline(s,n,widen('\n'))`

to:

Effects: `Calls get(sb,widen('\n'))`

Requester: Bjarne Stroustrup <bs@day.research.att.com>
References: lib-5786, 5791, 5794, 5795

Closed Issues

Issue	Action	Doc. #
18-003 Return value from type_info::name() implementation defined	None	N1064R1
18-004 placement delete	Core	N1064R1
18-005 exceptions thrown from `unexpected()`	Core	N1064R1
18-006 unexpected	Core	N1064R1
19-001 Should exception hierarchy constructors be explicit?	None	N1064R1
20-002 CopyConstructible const equivalence question	None	N1064R1
21-006 Add basic_string::push_back()	None	N1064R1
21-008 basic_string constructor specification	None	N1064R1
22-018 LC_CATEGORY in the C library	None	N1064R1
22-019 Requirements to a facet type not clear	None	N1064R1
23-001 Priority_queue<> missing typedef for compare_type	None	N1064R1
23-002 Gratuitous pointer and const_pointer typedefs	None	N1064R1
23-003 Member not required to be template function	None	N1064R1

23-004 Is function assign() required?	None	N1064R1
23-008 vector::resize() takes second argument by value	None	N1064R1
24-001 Undefined lifetime of references from iterators.	None	N1064R1
24-002 `istreambuf_iterator::proxy`	None	N1064R1
26-001 Example text should be normative	None	N1064R1
27-005 mistake in streampos declarations in <iosfwd>	None	N1064R1
27-006 no simple way to extract all remaining data from a stringstream	None	N1064R1
27-016 Status of copy operations (constructor and assignment) is unclear. (See 27-002)	None	N1064R1
27-018 Suppressing iostreams exceptions	None	N1064R1