Doc. no.:	J16/97-0026R1
	WG21/N1064R1
Date:	28 March 1997
Project:	Programming Language C++
Reply to:	Beman Dawes
	beman@esva.net

Libraries Issues List for CD2 – Version 4

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.

Open Issues

Library editorial issues

Issue: **CD2-editorial**
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 “equivalence of 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'.

Requester:
References:

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.

Proposed Resolution:

Change 17.3.2.2 [lib.using.linkage], paragraph 2 to read:

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

Or, replace the existing signatures for `atexit()`, `qsort()`, and `bsearch()`, `signal()` with:

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

extern "C" void qsort(void *base, size_t nmemb, size_t size,
      int (*compar)(const void *, const void *));
extern "C++" void qsort(void *base, size_t nmemb, size_t size,
      int (*compar)(const void *, const void *));
```

```
extern "C" void *bsearch(const void *key, const void *base,
      size_t nmemb, size_t size,
      int (*compar)(const void *, const void *));
extern "C++" void *bsearch(const void *key, const void *base,
      size_t nmemb, size_t size,
      int (*compar)(const void *, const void *));

extern "C" void (*signal(int sig, void (*func)(int)))(int);
extern "C++" void (*signal(int sig, void (*func)(int)))(int);
```

For the overloads of `atexit()`, calls to the registered functions should interleave as now, i.e. the implementation maintains one stack not two, and must remember the linkages of the registered functions if it matters.

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>
References:

Issue: CD2-18-003 Return value from type_info::name() implementation defined
Section: 18.5.1 Class type_info [lib.type.info]
Status: Closed
Description:

Page 18-15 Sec 18.5.1 Clause 7 defines `type_info::name()` as implementation-defined, so a conforming implementation could simply return a null string for all types, effectively making `name()`

unusable. Ideally, it should be defined to return the type name in some canonical form e.g. a fully-qualified elaborated type-id with no redundant spaces (although e.g. pointer non-type template parameters would require further specification). This would allow `name()` to be used to label types in a persistence library (e.g. a recent Microsoft Systems Journal described such a library). Failing this, I think that at least `name()` should be defined to return a unique string for each type to allow `type_info` to be used as a hook to further user-defined type information (as envisaged by Dr. Stroustrup in D&E.) In D&E page 318, it is suggested that `typeid(*p).name()` or `&typeid(*p)` could be used as an index into a map for this purpose, but currently neither expression is defined to be unique for different types.

Proposed Resolution:

Close, no change.

Requester: Brian Parker
References: Public comment 23/Parker

Issue: CD2-18-004 placement delete
Section: 18.4 Dynamic memory management [lib.support.dynamic], para
Status: Closed
Description:

Throughout this section, various placement delete functions are described as being called by a delete-expression. My understanding was that the placement delete functions were only called if an exception was thrown during a new expression. When are the "nothrow" placement delete functions called. Page 18-13 Sec 18.4.1.3 Clause 8 states that operator `delete(void* ptr, void*)` is the "default function called for a placement delete expression". What is a placement delete expression?

Proposed Resolution:

Core issue.

Requester: Brian Parker
References: Public comment 23/Parker

Issue: CD2-18-005 exceptions thrown from `unexpected()`
Section: 18.6.2.2 Type unexpected_handler [lib.unexpected.handler]
Status: Closed
Description:

Section 15.5.2 states that the `unexpected()` function can throw any exception and those not in the function's exception specification will be converted to`bad_exception` if that is in the exception specification. This section, however, states that a user supplied `unexpected_handler` must not throw exceptions not on the exception specification. What is the reason for this restriction?

Proposed Resolution:

Core issue.

Requester: Brian Parker
References: Public comment 23/Parker

Issue: CD2-18-006 unexpected
Section: 18.6.2.4 unexpected [lib.unexpected]
Status: Closed
Description:

Can this be declared static:

```
static void unexpected();
```

If not, is it an error to declare such a function? The implication of the manual appears to be that this cannot be static, but it is possible that this is wishful thinking on my part as an implementor.
BTW 18.6.2 is empty. Is this intentional?

Proposed Resolution:

Core issue.

Requester: David L Moore
References: Public comment 24/Moore

Clause 19 – Diagnostics

Issue: CD2-19-001 Should exception hierarchy constructors be explicit?
Section: 19.1 [lib.std.exceptions]
Status: Closed

Description:
Should the constructors in the exception hierarchy be explicit to prevent undesirable conversions from string?
Proposed Resolution:
Close, no action.
Requester:
References:

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: **CD2-20-002 CopyConstructible const equivalence question**
Section: 20.1.3 [lib.copyconstructible]
Status: Closed
Description:

The second line of the CopyConstructible requirements table specifies that for the expression `T(u)` where `u` is a value of type `const T` that "`u is equivalent to T(u)`".

Should this read "`u is equivalent to` **`const`** `T(u)`"?
Proposed Resolution:
Close, no action.
Requester: John Benito <jb@peren.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

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];
```

Proposed Resolution:

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.

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: CD2-21-006 Add basic_string::push_back()
Section: 21.3 Template class basic_string [lib.basic.string]
Status: Closed
Description:

The basic_string class should have a `push_back()` member function (as in Table 69, p. 23-5) to make it more compatible with the other container classes.

Proposed Resolution:

Close, no action.

Requester: Bill Dimm

References: Public comment 21/Dimm

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: CD2-21-008 basic_string constructor specification

Section: 21.3.1 basic_string constructors [lib.string.cons]

Status: Closed

Description:

For the constructor:

```
basic_string(const charT* s, size_type n,
        const Allocator& a = Allocator());
```

Paragraphs 6 & 7 are:

Requires:

`s` shall not be a null pointer and `n < npos`.

Throws:

`out_of_range` if `n == npos`.

Should be:

Requires:

`s` shall not be a null pointer and `n <= max_size()`.

Throws:

`out_of_range` if `n > max_size()`.

If you look in '21.3.3basic_string capacity [lib.string.capacity]' at the function

```
void resize(size_type n, charT c)
```

you'll see that this same thing done correctly (once) in the current draft. For this resize function the implementation is described in part as:

Requires:

`n <= max_size()`

Throws:

`length_error` if `n > max_size()`.

Moreover, there doesn't seem to be any particular reason to prohibit `n == npos`' specifically other than the fact that it won't succeed. The real limit on allocation is `max_size()`. One supposes that this is six of one and half a dozen of another, that is, if the condition for throwing an `out_of_range` exception involves npos as is currently stated in the draft then the user will inevitably get a `bad_alloc` exception for all `n > max_size()` if not an `out_of_range` exception. I believe that the expression should involve `max_size()` as shown above to be an effective error indication.

The same reasoning applies to the following instances of conditions on a throw statement, in addition to the instance cited above.

Proposed Resolution:

Reject.

Requester: John Mulhern

References: Public comment 31/Mulhern

Issue: **CD2-21-009 String complexities are not specified**
Section: 21.1.2 Character traits requirements [lib.char.traits.require]
Status: Pending
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: Pending
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:

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
Section: 22.1 Locales [lib.locales]
Status: Open (US)
Description:

The declarations of the non-member functions 'is*()' are declared to be 'const'. Although a 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
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**
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: **CD2-22-018 LC_CATEGORY in the C library**
Section: Annex C.4 [diff.library]
Status: Closed
Description:

LC_CATEGORY values are used in several interfaces of class locale. According to the C++ standard they are allowed to be combined by logical && and ||. Consequently the same must be required of the LC_CATEGORY values in the C library, as they are supposed to have the same semantics in C and C++.

We suggest to require a modification to the C library: LC_CATEGORY shall conform to the C++ specifications of LC_CATEGORY, i.e. it shall allow logical && and ||.

Proposed Resolution:

Take no action.

Proposed response to requester:

We do not want to require any changes to the C library. The values of LC_CATEGORY that can be or'ed are not valid values for the C library.

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

Issue: **CD2-22-019 Requirements to a facet type not clear**
Section: 22
Status: Closed
Description:

Is a facet class allowed to be pure virtual?

More generally: The standard does not specify what is required of a facet class, besides the fact that is has to be derived from class locale::facet and must have a static id member. It is not clear whether a facet class needs to have a public copy constructor, assignment, default constructor, destructor, etc.
For instance it does make sense to call use_facet for a facet type that is pure virtual, because use_facet returns a reference to a facet. One might wish to access the concrete facet object via this reference. However, some implementations do allow this, and the draft does not specify whether such an implementation would conform to the standard or not.

Proposed Resolution:

Close, no action.

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

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

Clause 23 – Containers

Issue: **CD2-23-001 Priority_queue<> missing typedef for compare_type**
Section: 23.2.3.2 [lib.priority.queue]
Status: Closed
Description:

std::priority_queue<> takes a template parameter "Compare", a function object, and defines a protected member with it, but there is no typedef for that parameter.

Proposed Resolution:

Add to the public interface of priority_queue<> in 23.2.3.2 [lib.priority.queue] the following definition:

```
typedef Compare compare_type;
```

Close, no action.

Requester: Nathan Myers <ncm@cantrip.org>
References: lib-5246

Issue: **CD2-23-002 Gratuitous pointer and const_pointer typedefs**
Section: 23, 21
Status: Closed
Description:

The standard containers provide `pointer` and `const_pointer` typedefs, but these do not appear in any requirement or function signature for any container, including basic_string.

Proposed Resolution:

Remove these typedefs.
Close, no action.

Requester: Greg Colvin <Greg@imr.imrgold.com>
References:

Issue: **CD2-23-003 Member not required to be template function**
Section: 23.2.1 [lib.deque], 23.2.2 [lib.list], 23.2.4 [lib.vector]
Status: Closed
Description:

The standard library sequences have a member declared

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

It is defined as

```
erase(begin(), end());
insert(begin(), n, t);
```

Maybe there is something profound I have completely missed, but I don't see why it is a member template function. Looking at the definition of insert(), it follows from Table 77 that

n	must be a value of size_type
t	must be a value of value_type (i.e., T)

Proposed Resolution:

Change the version of `assign()` taking `Size, const T&` arguments in 23.2.1, 23.2.1.1, 23.2.2, 23.3.3.1, 23.2.4. and 23.2.4.1 to be declared as:

```
void assign(size_type n, const T& t = T());
```

Close, no action.

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

Issue: **CD2-23-004 Is function assign() required?**
Section: 23.2.1 [lib.deque], 23.2.2 [lib.list], 23.2.4 [lib.vector]
Status: Closed
Description:

Function `assign()` is part of the standard sequences `deque`, `list` and `vector`, but not required from sequences in general (not listed in Table 77). Is it required or not?

Proposed Resolution:

Close, no action.

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

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:

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:

None yet.

Requester: Dr. Waldemar Horwat

References: Public comment #15/Horwat

Issue: **CD2-23-008 vector::resize() takes second argument by value**

Section: 23.2.4.2 vector capacity [lib.vector.capacity]

Status: Closed

Description:

The second argument for `vector::resize()` is passed by value (instead of reference to const object). In the absence of a compelling reason for pass by value, this should be changed to use a reference to const (for greater efficiency and more uniformity in the library). Page 23-25 defines `vector::resize()` in terms of `vector::insert()` (which uses a reference), so it is surprising to see that the two functions treat their arguments differently.

Proposed Resolution:

Close, no action.

Requester: Bill Dimm

References: Public comment 21/Dimm
CD2-23-009

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.

Proposed Resolution:

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 can probably be made to work.

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: Pending
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: Pending

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: Pending

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

Clause 24 – Iterators

Issue: **CD2-24-001 Undefined lifetime of references from iterators.**
Section: 24
Status: Closed
Description:

Chapter 24 places no requirements on the lifetime of the reference returned by *iterator. For example, given a dereferenceable input iterator p on type int, must the following assertion be true?

```
const int& r = *p;
int i = r;
p++;
assert(i == r);
```

Proposed Resolution:
The assertion should not be required to be true. The *iterator operation might return a temporary.
Close, no action.
Requester: Greg Colvin <Greg@imr.imrgold.com>
References:

Issue: **CD2-24-002 `istreambuf_iterator::proxy`**
Section: 24.5.3.2 istreambuf_iterator constructors [lib.istreambuf.iterator.cons]
Status: Closed
Description:

Earlier it is stated that class proxy was for exposition only and need not be supplied, but here a constructor taking it is required.
Proposed Resolution:
Close, no action.
Requester: Brian Parker
References: Public comment 25/Parker

Clause 25 – Algorithms
Clause 26 – Numerics

Issue: **CD2-26-001 Example text should be normative**
Section: 26.3.2.7 valarray member functions [lib.valarray.members] par 6
Status: Closed
Description:

The example says that new elements are created using the default constructor. I think this is normative text that should be moved outside of the example.
Close, no action. See para 5.
Proposed Resolution:
Requester: Dag Brück <dag@dynasim.se>
References:

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: Pending
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: CD2-27-005 mistake in streampos declarations in <iosfwd>
Section: 27.2 Forward declarations [lib.iostream.forward]
Status: Closed
Description:

The last two lines of <iosfwd> contain the declarations:

```
typedef fpos<char_traits<char>::state_type> streampos;
typedef fpos<char_traits<wchar_t>::state_type> streampos;
```

When I try to compile this, my compiler complains:

cxx: Error: ../cms_headers/iosfwd, line 75: incomplete type is not allowed
typedef fpos<char_traits<char>::state_type> streampos;

Proposed Resolution:

Since I assume we don't want the char_traits class definitions in <iosfwd>, the only solution the I can see is hard code the value of the typedef (mbstate_t). This would require a forward declaration of mbstate_t. Add:

```
class mbstate_t;
```

And replace the declarations of streampos and wstreampos with:

```
typedef fpos<mbstate_t> streampos;
typedef fpos<mbstate_t> wstreampos;
```

Close, no action.

Recommended response to proposer:

Although the code in the synopsis requires prior knowledge of what the types are, it does not require that the declaration of them be available in the form present in the synopsis. This is explained in the Note following the synopsis.

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

Issue: CD2-27-006 no simple way to extract all remaining data from a stringstream
Section:
Status: Closed
Description:

Presently there seems to be no obvious and simple way to extract all remaining data from a stringstream into a string. If true, this is a very serious oversight and one I hope will be addressed. The solution I suggest is to allow getline to accept an "int" for the terminating character (presently it requires a "char"). Then one could use getline (..., traits::eof()) to extract the data.

Proposed Resolution:

Close, no action.

Response to Requester:

There are many alternatives including doing the operation yourself. Another alternative is to use the string constructor that accepts an iterator and use an ostreambuf_iterator.

Requester: Russell E. Owen

References: Public comment #09/Owen

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: CD2-27-016 Status of copy operations (constructor and assignment) is unclear.
Section: 27.6.3 Standard manipulators [lib.std.manip]
Status: Closed
Description:
No mention is made of these operations. By the ordinary rules of interpretation of the library this means they would be allowed, but their definition is unclear. Historically they were not allowed by iostream classic.
Proposed Resolution:
Duplicate of CD2-27-002
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: CD2-27-018 Suppressing iostreams exceptions
Section: Annex D [depr]
Status: Closed
Description:

The stream classes for type char shall not throw any exceptions if the exception mask is set to zero. We think that users will expect that clearing the exception mask will suppress all exceptions from iostreams. In other words, there is a demand for a true compatibility mode to existing implementations of the old tiny character iostreams.

It is our understanding that in general iostreams is not required to catch all conceivable exceptions, not even if the exception mask is set to zero. For instance, exceptions that might be thrown by an operation of the character or the traits type need not be suppress. All that is said is that exceptions raised by the stream buffer will be caught by the stream classes and will result in badbit set, which then might or might not raise an exception, depending on the mask.

Proposed Resolution:

For the sake of compatibility we suggest to require the tiny character streams to exhibit truly compatible behavior and suppress ALL exceptions in case the exception mask is cleared.

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

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: active
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::boo and ostream::sentry::boo 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: