By: Philippe Le Mouël
Rogue Wave Software Inc.
philippe@roguewave.com

Doc. No.: X3J16/96-0009
WG21/N0827
Date: January 30 1996

IOStreams Issues List
Library Clause 27

Revision History

Pre-Santa Cruz	X3J16/96-0009 WG21/N0827
Post-Tokyo	X3J16/95-0221 WG21/N0821
Pre-Tokyo	X3J16/95-0194 WG21/N0794
Pre-Monterey	X3J16/95-0089 WG21/N0689
Pre-Austin	X3J16/95-0034 WG21/N0634

Summary of Issues

27.4.4 basic_ios — P. 18

Active 27-203 operator bool( ) needs to be fixed

Closed 27-201 remove throw specifications for clear and setstate (Tokyo)
Closed 27-202 tie not required to be associated with an input sequence (Tokyo)
Closed 27-204 replace int_type by char_type in int_type fill( ) and int_type fill(int_type) (Tokyo)

27.5.2 basic_streambuf — P. 19

Active 27-301 imbuing on streambufs. When, how often, etc...
Active 27-303 not_eof needs to be used where appropriate
Active 27-304 uflow needs editing
Active 27-305 basic_streambuf::showmanyc Incorrect return clause
Active 27-306 basic_streambuf::uflow has incorrect default behavior
Active 27-307 basic_streambuf::uflow has nonsense returns clause
Active 27-308 streambuf inlines
Active 27-309 (27-350 in Post Tokyo) two return clauses for streambuf::underflow
Active 27-310 (27-351 in Post Tokyo) streambuf::pbackfail has incorrect **Notes:** clause
Active 27-311 (27-352 in Post Tokyo) caching results of calls to locale functions

Closed 27-302 sungetc has an incorrect return type (Tokyo)

27.6.1 basic_istream — P. 24

Active 27-401 isfx what does it do?
Active 27-402 ipfx example is incorrect
Active 47-403 Clarification of exceptions thrown
Active 27-404 istream functions need to check for NULL streambuf
Active 27-405 (27-450 in Post Tokyo) confusing English in formatted requirements
Active 27-406 (27-451 in Post Tokyo) operator>>(char_type *) failure
Active 27-407 (27-452 in Post Tokyo) operator>>(char_type) failure
Active 27-408 (27-453 in Post Tokyo) ws manipulator
Active 27-409 unsigned short extractors cannot use unsigned long get function

27.6.2 basic_ostream — P. 29

Active 27-501 op<<(char) needs to be consistant with the other formatted inserters
Active 27-502 op<<(void *) should it be const volatile void *
Active 27-503 ostream functions need to check for NULL streambuf
Active 27-504 (27-550 in Post Tokyo) exceptions in ostream
Active 27-505 (27-551 in Post Tokyo) incorrect conversion specifier for operator<<(unsigned long)
Active 27-506 wrong default behavior for padding

27.6.1-27.6.2 basic_istream, basic_ostream — P. 35

Active 27-601 op[<<|>>](ios_base&) needed for manipulators
Active 27-602 positional typedefs in istream/ostream derived classes are not needed
Active 27-603 read/write should take a void * instead of a char_type *
Active 27-604 Should we require ios::in to be set for istream's and ios::out to be set for ostream's?

Active	27-918	(27-953 in Post Tokyo) Validity of OFF_T to POS_T conversion
Active	27-919	(27-954 in Post Tokyo) Question on Table 2 assertions
Active	27-920	(27-955 in Post Tokyo) destination of clog and wclog
Active	27-921	default locale argument to constructor

Annex D P. 66

Active	27-1001	description of function setbuf is not sufficient
Active	27-1002	strstreambuf Editorial issues (typo's)
Active	27-1003	istrstream Editorial issues (typo's)
Active	27-1004	ostrstream Editorial issues (typo's)

ios_traits issues

Issue Number: 27-001
Title: changing traits::newline to be locale aware
Section: 27.4.2.1 **ios_traits value functions [lib.ios.traits.values]**
Status: **active**
Description:

The problem with traits::newline is that it does not know about the currently imbued locale.

This proposal addresses the need for a locale-aware newline.

Possible Resolution:

Change traits::newline by adding a parameter for locale information:

```
static char_type newline(const ctype<char_type>& ct);
```

The default definition is as if it returns:

```
ct.widen('\n');
```

Some functions in basic_istream have a default parameter that is: traits::newline() (getline, get). These defaults will have to be changed to use the currently imbued locale. Changing the default value to: traits::newline(getloc()) won't work because getloc() is not static. Therefore the functions that have newline() as a default value must be split into two functions; one function that has three parameters, and one function that has two parameters and calls the three parameter function with a "default" value. For example:

```
istream_type& getline(char_type *, streamsize, char_type delim);

istream_type& getline(char_type *s, streamsize n)
    {
   return getline(s, n, newline(
     use_facet<ctype<char_type> >( getloc() )));
  }
```

The functions that need to change are:

```
istream_type& get(char_type *, streamsize, char_type);
istream_type& get(streambuf_type&, char_type);
istream_type& getline(char_type *, streamsize, char_type);
```

Requestor: Nathan Myers (ncm@cantrip.org),
John Hinke (hinke@roguewave.com)

Issue Number: 27-002
Title: traits::is_whitespace() is inconsistent
Section: 27.4.2.2 **ios_traits test functions [lib.ios.traits.tests]**
Status: **active**
Description:

This function is inconsistent throughout the document. For example:

27.4.2 Template struct ios_traits [lib.ios.traits]
static bool is_whitespace(int_type, const ctype<char_type>&);

27.4.2.2 ios_traits test functions [lib.ios.traits.tests]
bool is_whitespace(int_type c, const ctype<char_type>& ct);

Returns: true if *c* represents a white space character. The default definition is as if it returns *ct.*isspace(*c*).

The returns paragraph calls a method of ctype that does not exist.
It should say:

Returns: true if *c* represents a white space character. The default definition is as if it returns *ct.*is(*ct*.space, *c*).

27.6.1.1.2 basic_istream::ipfx [lib.istream.prefix]
Notes: ...uses the function
bool traits::is_whitespace(charT, const ctype<charT>&)

The same paragraph goes on to use ctype<...> in the example.

27.6.1.1.2 Paragraph 4: [lib.istream.prefix]
static bool is_whitespace(char, const ctype<charT>&)

Possible Resolution:

The problem is determining which signature is correct.

As pointed out in Box 6 (27.4.2 Template struct ios_traits [lib.ios.traits]) the locale functions that actually test for whitespace work on char_type values.

I propose the following change:

static bool is_whitespace (char_type *c*, const ctype<char_type>& *ct*);

Returns: true if *c* represents a white space character. The default definition is as if it returns *ct.*is(*ct*.space, *c*).

Requestor: John Hinke (hinke@roguewave.com)
Philippe Le Mouël (philippe@roguewave.com)

Issue Number: 27-004
Title: example of changing the behavior of is_whitespace is incorrect.
Section: 27.6.1.1.2 Paragraph 4 **basic_istream prefix and suffix [lib.istream.prefix]**
Status: **active**
Description:

The example of changing behavior of is_whitespace is incorrect. It should read:

```
struct my_char_traits : public ios_traits<char> {
    static bool is_whitespace(char c, const ctype<char>& ct)
        { ...my own implementation... }
};
```

Possible Resolution:

Change from:

```
struct my_char_traits : public ios_traits<char> {
    static bool is_whitespace(char c, const ctype<charT>& ct)
        { ...my own implementation... }
};
```

To:

```
struct my_char_traits : public ios_traits<char> {
    static bool is_whitespace(char c, const ctype<char>& ct)
        { ...my own implementation... }
};
```

Requestor: John Hinke (hinke@roguewave.com)

Issue Number: 27-005
Title: not_eof specification
Section: 27.4.2.1 ios_traits value functions **[lib.ios.traits.values]**
Status: **active**
Description:

int_type not_eof(int_type c);

Editorial: "**Notes:**" should also mention it is used for sbumpc and sgetc.

Per Bothner writes:
"The **Returns:** is incompatible with the traditional masking function for zapeof. This is because int_type(-2) == -2 while zapeof(-2) == ((-2) & 0xFF). And nowhere else does it say anything that would allow the traditional implementation."

"I don't understand the presentation style well enough to suggest the proper fix. But somewhere it should say or imply that when charT is specialized with char, then not_eof(c) is int_type((unsigned char)(c))."

Possible Resolution:

Requestor: Per Bothner (bothner@cygnus.com)

Issue Number: 27-007
Title: ios_traits typedefs are ‘char’ oriented.
Section: 27
Status: **active**
Description:

We cannot specify int_type, off_type, pos_type, and state_type corresponding to some specialized charT type.

For example, if in order to think about ‘char’ specialization, we might define

```
template <class charT> struct ios_traits {
    ....
    typedef charT char_type;
    typedef int int_type;
    ....
};
```

we would have to accept it as constant definition in all of the specialized traits, not only ios_traits<char>, but ios_traits<wchar_t>, ios_traits<ultrachar>. This would lead to the restriction upon implementations that all of the charT must be converted in 'int' range. The restriction is too heavy for future wide character types and user-defined character types.

Possible Resolution:

Adopt the following definition:

```
namespace std {
    template <class charT> struct ios_traits {};

    struct ios_traits<char> {
            typedef char            char_type;
            typedef int             int_type;
            typedef streampos       pos_type;
            typedef streamoff       off_type;
            typedef mbstate_t       state_type;

        // 27.4.2.2 values:
            static char_type        eos();
            static int_type         eof();
            static int_type         not_eof(int_type c);
            static char_type        newline();
            static size_t           length(const char_type* s);

        // 27.4.2.3 tests:
            static bool             eq_char_type(char_type, char_type);
            static bool             eq_int_type(int_type, int_type);
            static bool             is_eof(int_type);
            static bool             is_whitespace(int_type, const ctype<char_type>
                                            ctype&);

        // 27.4.2.4 conversions:
            static char_type        to_char_type(int_type);
```

```
        static int_type           to_int_type(char_type);
        static char_type*         copy(char_type* dst, const char_type* src, size_t n);
        static state_type         get_state(pos_type);
        static pos_type           get_pos(streampos fpos, state_type state);
    };

    struct ios_traits<wchar_t> {
        typedef wchar_t           char_type;
        typedef wint_t            int_type;
        typedef wstreampos        pos_type;
        typedef wstreamoff        off_type;
        typedef mbstate_t         state_type;

      // 27.4.2.2 values:
        static char_type          eos();
        static int_type           eof();
        static int_type           not_eof(int_type c);
        static char_type          newline();
        static size_t             length(const char_type* s);

      // 27.4.2.3 tests:
        static bool               eq_char_type(char_type, char_type);
        static bool               eq_int_type(int_type, int_type);
        static bool               is_eof(int_type);
        static bool               is_whitespace(int_type, const ctype<char_type>
                                          ctype&);

      // 27.4.2.4 conversions:
        static char_type          to_char_type(int_type);
        static int_type           to_int_type(char_type);
        static char_type*         copy(char_type* dst, const char_type* src, size_t n);
        static state_type         get_state(pos_type);
        static pos_type           get_pos(streampos fpos, state_type state);
    };
}
```

According to the separation of the two specializations, we have to change the descriptions in **[lib.streams.types]**, as follows;

27.4.1 Types

```
typedef OFF_T streamoff;
```

The type streamoff is an implementation-defined type that satisfies the requirements of type OFF_T.

```
typedef WOFF_T wstreamoff;
```

The type wstreamoff is an implementation-defined type that satisfies the requirements of type WOFF_T.

```
typedef POS_T streampos;
```

The type streampos is an implementation-defined type that satisfies the requirements of type POS_T.

typedef WPOS_T wstreampos;

The type wstreampos is an implementation-defined type that satisfies the requirements of type WPOS_T.

typedef SIZE_T streamsize;

The type streamsize is a synonym for one of the signed basic integral types. It is used to represent the number of characters transferred in an I/O operations, or the size of I/O buffers.

Comments:

We can find the above approach, "defining nothing in the template version of traits and defining everything in each specializations", in my original proposal (X3J16/94-0083). I am afraid (and sorry) that one of the mistakes made in my document for Austin (X1J16/95-0064) introduced such an inappropriate definitions to the current WP.

I feel this change request is in a kind of 'editorial' class.

We should not put any definitions (static member functions or typedefs) related to int_type, off_type, pos_type and/or state_type in the template definition of the traits. The reason is that in fact, these three types depend on the template parameter class 'charT' for variety of environments (ASCII, stateless encodings for double byte characters, UniCode). For example,

charT	char	wchar_t
int_type	int	wint_t
off_type	streamoff	wstreamoff
pos_type	streampos	wstreampos
state_type	mbstate_t	mbstate_t

Note that two of the above types, 'wint_t', 'mbstate_t' are defined in C Amendment 1 (or MSE).

We cannot assume that two implementation-defined types, streampos and wstreampos have the same definitions because under some shift encodings, wstreampos have to keep an additional information, the shift state, as well as the file position. We should represent them with two different symbols, POS_T and WPOS_T so as to give a chance to provide separate definitions in these two specializations.

For pos_type in both specialized traits, the type 'mbstate_t' is introduced from C Amendment 1(or former MSE) and is an implementation-defined type that can represent any of shift states in file encodings.

The type, INT_T is not suitable for the definition of streamsize because INT_T represents another character type, whose meaning is different to those of streampos. So a new symbol 'SIZE_T' will need to specify the definitions of streampos.

Possible Resolution:

Issue 27-006 closed in Tokyo solves the streamsize problem by defining it as:

typedef SZ_T streamsize;

The WP also specifies (27.4.2 Template struct ios_traits [lib.ios.traits] paragraph 2) that an implementation shall provide the following two instantiations of ios_traits:

```
struct ios_traits<char>;
struct ios_traits<wchar_t>;
```

Like Norihiro Kumagai, I feel that the types int_type, pos_type, off_type and state_type have to be specified in each specialization. But to me, the WP is correct when it says (27.4.2 Template struct ios_traits [lib.ios.traits]):

```
namespace std {
  template <class charT> struct ios_traits<charT> {

      typedef  charT        char_type;
      typedef   INT_T       int_type;
      typedef   POS_T       pos_type;
      typedef   OFF_T       off_type;
      typedef  STATE_T      state_type;
                    .
                    .
```

I understand by the above that a specialization has to provide the types int_type, pos_type, off_type and state_type and that these types have to obey the description of type INT_T for int_type, the description of POS_T for pos_type, the description of OFF_T for off_type, and the description of STATE_T for state_type.
Therefore you can have the following:

```
struct ios_traits<char> {
        typedef char              char_type;
        typedef int               int_type;
        typedef streampos         pos_type;
        typedef streamoff         off_type;
        typedef mbstate_t         state_type;
                     .
                     .
```

Which means to me:

int is following the description of INT_T (27.1.2.2 Type INT_T [lib.iostream.int.t])
streampos is following the description of POS_T (27.1.2.4 Type POS_T [lib.iostream.pos.t])
streamoff is following the description of OFF_T (27.1.2.3 Type OFF_T [lib.iostream.off.t])
mbstate_t is following the description of STATE_T (27.1.2.6 Type STATE_T)

May be we should make clarifications in the WP about this fact and also add that an implementation is required to specialized ios_traits on whatever charT type it is using.

Requestor: Norihiro Kumagai (kuma@slab.tnr.sharp.co.jp)

Issue Number: 27-008
Title: `ios_traits::length` is missing **Returns:** clause
Section: 27.4.2.1 **ios_traits value functions [lib.ios.traits.values]**
Status: **active**
Description:

`ios_traits::length` has an **Effects:** clause but no **Returns:** clause. The **Effects:** clause should be reworded as a **Returns:** clause.

Possible Resolution:

Remove the **Effects** clause and add:

Returns: The length of a null terminated character string pointed to by s.

Requestor: Public Comment

Issue Number: 27-009
Title: definition for `get_state`
Section: 27.4.2.3 **ios_traits conversion functions [lib.ios.traits.convert]**
Status: **active**
Description:

The definition of `ios_traits::get_state` is incomplete. Here is the complete description:

`state_type get_state(pos_type pos);`

Returns: A `state_type` value which represents the conversion state in the object *`pos`*.

Possible Resolution:

In section 27.1.2.4 **Type POS_T** paragraph 2 of the WP it is specified:

“ The type POS_T describes an object that can store all the information necessary to restore an arbitrary sequence to a previous stream position and conversion state. “

So I think we can safely change the return clause, as proposed by Norihiro Kumagai, to:

Returns: A `state_type` value which represents the conversion state in the object *`pos`*.

Requestor: Norihiro Kumagai (kuma @ slab.tnr.sharp.co.jp)

Issue Number: 27-010
Title: definition for `get_pos`
Section: 27.4.2.3 **ios_traits conversion functions [lib.ios.traits.convert]**
Status: **active**
Description:

The definition of `ios_traits::get_pos` is incomplete. Here is the complete description:

`pos_type get_pos(streampos pos, state_type s);`

Effects: Constructs a `pos_type` value which represents the stream position corresponding to the pair of *pos* and *s*.

Returns: A `pos_type` value which consists of the values of *pos* and *s*.

Possible Resolution:

Requestor: Norihiro Kumagai (kuma @ slab.tnr.sharp.co.jp)

Title: Return type for `ios_traits::copy` is incorrect
Section: 27.4.2.3 **`ios_traits` conversion functions [lib.ios.traits.convert]**
Status: **active**
Description:

The return type for `ios_traits::copy` says to return `dst`. It should return `dest`.

Possible Resolution:

Change the returns clause to: **Returns:** dest

Requestor: John Hinke (hinke@roguewave.com)

`ios_base` issues

Issue Number: 27-101
Title: `ios_base` manipulators
Section: 27.4.5 **`ios_base` manipulators [lib.std.ios.manip]**
Status: **active**
Description:

There is only one ios_base manipulator that says, “Does not affect any extractors.” (showbase)

This implies that the rest of the manipulators affect extractors. If the manipulators only affect insertors (ignoring skipws), then perhaps they should be ostream manipulators instead of ios_base manipulators. If they are left as ios_base manipulators, then they should affect extractors as well as insertors.

The locale num_get facet says, “Reads characters from *in*, interpreting them according to *str*.flags()...” This implies that the manipulators affect the extraction of values from a stream.

A couple of cases:

```
unsigned int    ui;
int             i;

cout << -10;
cin >> ui;                          // What should this read in?
cout << showpos << 10;              // +10
cin >> ui;                          // What about this?

cout << showbase << hex << 10;      // 0xa
cin >> i;                           // Should this be valid?
cout << showbase << hex << 10;      // 0xa
cin >> showbase >> hex >> i;        // What about this?
```

Possible Resolution:

John wrote the following possible resolution:

“Keep all manipulators as they are but say something to the effect that the manipulators affect both insertors and extractors. Remove the Notes on showbase. This is different behavior than the original AT&T implementation.

Editorial Issue: These manipulators should be moved to the ios_base clause.”

In section 27.4.3.1.2 **type ios_base::fmtflags**, Table 3 specifies how input and output are affected by the different fmtflags fields (therefore by the fmtflags manipulators).
So if you do:

```
cin  >> i;              //  enter 0xa
cout  << dec << i;      // print 10
cin >> hex >> i;        // enter 10
cout << dec << i;       // print 16
```

We should remove the note on showbase or add a note to all the other manipulators specifying their behavior on insertors and extractors according to Table 3.

Requestor: John Hinke (hinke@roguewave.com)

Issue Number: 27-102
Title: `ios_base::width` semantics are incorrect
Section: 27.4.3.2 **`ios_base fmtflags` state functions [lib.fmtflags.state]**
Status: **active**
Description:

The current description for `ios_base::width()` is:
Returns: The field width (number of characters) to generate on certain output conversions."

It should read "**Returns:** The minimum field width ...."

Possible Resolution:

Replace the returns clause by:

Returns: The minimum field width (number of characters) to generate on certain output conversions.

Requestor: Public Comment

Issue Number: 27-103
Title: proposal for adding `ios_base::maxwidth`
Section: 27
Status: **active**
Description:

This is probably too late to make it into the standard (unless the process rolls into further extensive revisions and balloting anyway, which -- judging from the state of the Input/Output library section -- seems likely :->}), but I'll point it out it all the same. If we really want programs to use the iostreams package instead of the FILE I/O calls, the iostreams package should provide as a minimum the same facilities as the older library. Specifically, the standard C I/O package provides a convenient method for controlling the maximum number of characters to write in formatted I/O, e.g.:

```
fprintf(fp, "FONT NAME: %.16s\n", font_desc.font_name);
```

This handles the case of a structure which has enough space for a string which will not necessarily be NUL-terminated if the maximum number of characters are stored for the string (a common enough situation when one is manipulating data structures written by someone else's software).

What are the reasons for leaving this out of the iostreams package? Also (while on the topic of rounding out iostreams to match what the competition can do), how difficult would it be to provide the ability to control the (minimum) number of digits in the exponent for a formatted floating point number written using scientific notation (as, for example, one can do in Ada)?

Possible Resolution:

The previous example “fprintf(fp, "FONT NAME: %.16s\n", font_desc.font_name);” can easily be achieved by iostream, with unformatted output.

Here is an example of achieving the same behavior:

```
ofstream fp("filename");
fp.write ( font_desc.font_name , 16 );
```

According to the example above, I think we should close the issue concerning the maxwidth field, I do not see the advantage of providing the same functionality for the formatted output.

Concerning the second remark “how difficult would it be to provide the ability to control the (minimum) number of digits in the exponent for a formatted floating point number written using scientific notation”, I propose the following:

Add to **27.4.3 Class ios_base [lib.ios.base]:**

```
streamsize expwidth( ) const;
streamsize expwidth( streamsize wide );
```

Add to **27.4.3.2 ios_base fmtflags state functions [lib.fmtflags.state]**

streamsize expwidth( ) const;

Returns: The minimum exponent width to generate when outputting floating point number in scientific notation.

Streamsize expwidth(streamsize wide);

Postcondition: wide == expwidth( ).
Returns: The previous value of expwidth( ).

Add to **27.4.4.1 basic_ios constructors [lib.basic.ios.cons] Table 8-ios_base( ) effects:**

expwidth( )   zero

Add to **27.6 Formating and manipulators [lib.iostream.format] Header <iomanip> synopsis**

T7 setexpw(int n);

Add to **27.6.3 Standard manipulators [lib.std.manip]:**

smanip setexpw(int n);

Returns: smanip(f, n), where f can be defined as:

```
ios_base& f( ios_base& str, int n )
 { // set minimum exponent width
   str.expwidth( n );
   return str;
 }
```

Requestor: Public Comment

Issue Number: 27-104
Title: `ios_base` unitbuf and nounitbuf manipulators
Section: 27.4.5.1 **`fmtflags` manipulators [lib.fmtflags.manip]**
Status: **active**
Description:

In section 27.4.3.1.2 (Type ios_base::fmtflags) Table 3 describes all the different fmtflags, and section 27.4.5.1 (`fmtflags` manipulators) describes all the fmtflags manipulators. The remark is that all the fmtflags are associated with a manipulator or two (ex: showpos and noshowpos manipulators) except unitbuf fmtflags. I think we should provide manipulators for the unitbuf fmtflags, otherwise users will have to be familiar with both way of setting the fmtflags.

Possible Resolution:

Add the two following manipulators:

ios_base& unitbuf (ios_base& str);

Effects: Calls str.setf (ios_base::unitbuf).
Returns: str.

ios_base& nounitbuf (ios_base& str);

Effects: Calls str.unsetf (ios_base::unitbuf).
Returns: str.

Requestor: Philippe Le Mouël (philippe@roguewave.com)

Issue Number: 27-105
Title: ios_base storage functions are not exception safe
Section: 27.4.3.4 **ios_base storage functions [lib.ios.base.storage]**
Status: **active**
Description:

This issue is just a reference to Nathan's proposal, which is in a separate document.
The document title is "Exception Safety for Iostreams" and its number is "X3J16/96-0024, WG21/N0842"..

Possible Resolution:

Requestor: Nathan Myers (ncm@cantrip.org)

`basic_ios` issues

Issue Number: 27-203
Title: `operator bool()` needs to be fixed
Section: 27.4.4.3 **basic_ios iostate flags functions [lib.iostate.flags]**
Status: **active**
Description:

Defining ios_base (or, as it appears in my copy of the WP, basic_ios) with a member operator bool() seemed like a good idea at the time, but perhaps the change should be withdrawn.

The reason is: while a conversion to void* is mostly harmless because few functions accept a void* argument, and void* doesn't silently convert to anything else, with an operator bool, the following absurdities are well-defined:

```
1 + cin
sin(cin)
vector<int> v(cin);
```

and (worse) ambiguities like

```
void f(istreambuf_iterator<char>);
void f(double);

f(cin);  // ambiguous
```

have been introduced. In other words, this change broke reasonable code. The problem is that bool is an arithmetic type, and is ill-behaved.

Possible Resolution:

Replace the member `ios_base::operator bool()` with member `ios_base::operator const void*()`, specified to return 0 if fail( ) is true, and non 0 if it is false.

This restores the code we broke, and also prevents frustrating ambiguities in new code.

[ED Note: This is assuming that these functions will be moved to `ios_base` *as suggested in one of the editorial boxes]*

The Tokyo meeting add editorial box 25.

Requestor: Nathan Myers (ncm@cantrip.org)

`basic_streambuf` issues

Issue Number: 27-301
Title: imbuing on streambufs: when, how often, etc...
Section: 27.5.2.2.1 **Locales [lib.streambuf.locales]**
Status: **active**
Description:

There needs to be something said as to when a new locale can be imbued into a streambuf or stream. Which operations are considered "atomic" in regards to locale changes.

Possible Resolution:

The effect of calling `imbue` during activation of any member of a class derived from `basic_ios<>`, or of any operator << or >> in which the class is the left argument, is unspecified. In particular (e.g.) any `codeset` conversion occurring in the streambuf may become incompatible with the formats specified by the old locale and still used.

The effect of calling `streambuf::imbue` or `pub_imbue` during activation of any streambuf virtual member is also undefined.

Requestor: Nathan Myers (ncm@cantrip.org)

Issue Number: 27-303
Title: not_eof needs to be used where appropriate
Section: 27.5.2.2.3 **Get area [lib.streambuf.pub.get]**
Status: **active**
Description:

27.5.2.2.3 Get area **[lib.streambuf.pub.get]**

int_type sbumpc();
Returns: "...returns char_type(*gptr())..."

This should be changed to say, "...returns not_eof(*gptr())..."

int_type sgetc();
Returns: "...returns char_type(*gptr())."

This should be changed to say, "...returns not_eof(*gptr())..."

See also box 29, 30, 31.

P. J. Plauger wrote:

"traits::not_eof is used in two places, both in overrides to virtual members of basic_streambuf. The first is in overflow, which should begin with code like:

```
virtual int_type overflow ( int_type ch = traits::eof( ) )
        { if  ( traits::is_eof ( ch ) )
```

```
        return (traits::not_eof( ch ) ); }
```

The second is in pbackfail, which should begin with code like:

```
virtual int_type pbackfail( int_type ch = traits::eof( ) )
     {  if ( gptr( ) !=0 && eback( ) < gptr( ) && traits::is_eof( ch ) )
        { <decrement next pointer for input buffer>;
          return ( traits::not_eof( ch ) ); }
```

These are the two places in basic_streambuf where an eof argument can lead to a successful (non-EOF) return".

Possible Resolution:

I agree with P. J. Plauger and I do not think we should add not_eof in the return statement of functions sbumpc, sgetc, uflow and sputc. I propose we remove boxes 29, 30 and 31 and change " … returns the value of traits::not_eof(*gptr( )) …" in box 34 to " … returns the value of *gptr( ) …".

Requestor: Per Bothner (bothner@cygnus.com)

Issue Number: 27-304
Title: uflow needs editing
Section: 27.5.2.4.3 **Get area [lib.streambuf.virt.get]**
Status: **active**
Description:

27.5.2.4.3 Get area **[lib.streambuf.virt.get]**

int_type uflow();

Default behavior: "...returns *gptr()."

This should be changed to, "...returns not_eof(*gptr())."

Returns: traits::not_eof(c)

This should be changed to, "traits::not_eof(*gptr())"

See also box 34.

Possible Resolution:

Box 34 describes the correct behavior of the uflow function except for the return value which should be *gptr( ) rather than traits::not_eof(*gptr ( )) (see issue 27-303). I propose to change Box 34 to reflect this fact and close the issue.

Requestor: Per Bothner (bothner@cygnus.com)

Issue Number: 27-305
Title: basic_streambuf::showmanyc Incorrect return clause
Section: 27.5.2.4.3 **Get area [lib.streambuf.virt.get]**
Status: **active**
Description:

basic_streambuf::showmanyc Returns has been corrupted. The function should return the number of characters that can be read with no fear of an indefinite wait while underflow obtains more characters from the input sequence. traits::eof() is only part of the story. Needs to be restored to the approved intent. (See footnote 218.)

Possible Resolution:

Footnote number 12 says " … The intention is not only that the calls will not return eof( ) but that they will return immediately.". I think the footnote clarifies the above remark and, therefore, the issue can be closed.

Requestor: Public Comment

Issue Number: 27-306
Title: basic_streambuf::uflow has incorrect default behavior
Section: 27.5.2.4.3 **Get area [lib.streambuf.virt.get]**
Status: **active**
Description:

basic_streambuf::uflow default behavior "does" gbump(1), not gbump(-1). It also returns the value of *gptr() *before* "doing" gbump.

Possible Resolution:

The description of uflow says:
"The constraints are the same as for underflow( ), except that the result character is transferred from the pending sequence to the backup sequence"

The description of underflow says:
"Returns: the first character of the pending sequence, if possible, without moving the input sequence position past it …"

Therefore uflow must:

+ Call underflow(traits::eof( )), which will return the first character of the pending sequence (*gptr( )) without moving it or traits::eof ( ).

+ If underflow does not return traits::eof( ) uflow has to transfer the result character (coming from underflow, which is *gptr( )) from the pending sequence to the backup sequence. This is done by doing gbump(1), and is supposed to return the same character as underflow, which is *gptr( ) before doing gbump(1).

The Tokyo meeting added Box 34, which fixes the problem except for the return clause, which should be *gptr( ) and not "traits::not_eof(*gptr( ))" (see issue 27-304). I propose that we change the return clause in Box 34 as described previously and close the issue.

Requestor: Public Comment

Issue Number: 27-307
Title: basic_streambuf::uflow has nonsense returns clause
Section: 27.5.2.4.3 **Get area [lib.streambuf.virt.get]**

Status: **active**
Description:

basic_streambuf::uflow has a nonsense **Returns:** clause. Should be struck.

Possible Resolution:

Change the **Returns:** clause to: “traits::eof() to indicate failure.”
The Tokyo meeting added Box 34 which fixes the problem. I propose we close the issue.

Requestor: Public Comment

Issue Number: 27-308
Title: streambuf inlines
Section: 27.5.2
Status: **active**
Description:

Nathan Myers (ncm@cantrip.org) writes:
I have begun looking more closely into the description of streambuf semantics, particularly the inlines like sgetc() and sbumpc().

These functions are typically called in inner loops of I/O code, so their performance critically affects I/O bandwidth. Any unnecessary elaboration costs everyone.

I notice that these functions are specified in terms of pointers that are (e.g.) "NULL or >= egptr()". This means that the inline functions must check the buffer pointers for both a NULL value *and* for end-of-buffer. Traditional implementations only check for end-of-buffer, resulting in smaller/faster code.

Does anyone remember when the possibility of these pointers being set to NULL was added, and why?

Per Bothner (bothner@cygnus.com) writes:
Traditional implementations allow *all* of the get pointers to be NULL, which is the initial state before buffers have been allocated. This case would be subsumed by (say) "gptr() < egptr()" on normal machines. But the standard perhaps does not require that "NULL < NULL" be well-defined (think weird segmented architectures), so NULL may need to be mentioned especially.

Jerry Schwarz (jss@declarative.com) writes:
(a) It has always been possible for them to be NULL. However when they are NULL they must all be NULL so you don't need a special check. This is the traditional interface.

(b) These are private pointers. The only way to set them or get them is through member functions. What those member functions do with NULL values is up to them.

Possible Resolution:

The Tokyo meeting adds Box 26 in clause 27.5.1 **Stream buffer requirements [lib.streambuf.reqts]**, which fixes the issue. I propose to close the issue.

Requestor: Nathan Myers (ncm@cantrip.org)

Issue Number: 27-309
Title: two return clauses for streambuf::underflow
Section: 27.5.2.4.3 **Get area [lib.streambuf.virt.get]**
Status: **active**
Description:

basic_streambuf::underflow has two **Returns:** clauses. Should combine them to be comprehensive.

Possible Resolution:

Remove the last return clause "Returns: traits::eof to indicate failure" and correct the typo in the first return clause "… If the pending sequence is null then the function returns traits::eof() to indicate failure" and not "… to indicate faulure".

Requestor: Public Comment

Issue Number: 27-310
Title: streambuf::pbackfail has incorrect **Notes:** clause
Section: 27.5.2.4.4 **Putback [lib.streambuf.virt.pback]**
Status: **active**
Description:

basic_streambuf::pbackfail Note begins a sentence with "Other calls shall." Can't apply "shall" to user program behavior, by the accepted conformance model.

Possible Resolution:

A user program will not directly call this function, since it belongs to the protected interface of the class basic_streambuf. Therefore I think it is reasonable to use the verb shall to point out that any derived class from basic_streambuf has to meet the listed requirements before calling pbackfail. I propose to close the issue.

Requestor: Public Comment

Issue Number: 27-311
Title: caching results of calls to locale functions
Section: 27.5.2.4.1 **Locales [lib.streambuf.virt.locales]**
Status: **active**
Description:

"Between invocations of this function a class derived from streambuf can safely cache results of calls to locale functions and to members of facets so obtained." Does this mean that changes in locale can be effectively ignored by the streambuf?

Possible Resolution:

This issue should be resolved with issue 27-301.

Requestor: Public Comment

---

basic_istream issues

Issue Number: 27-401
Title: istream::isfx
Section: 27.6.1.1.2 **basic_istream prefix and suffix [lib.istream.prefix]**
Status: **active**
Description:

What is the purpose of this function? The WP says, "**Effects:** None." Should it do something more? Or is it implementation defined!

Possible Resolution:

The Tokyo meeting deprecated ipfx and isfx in favor of the member type sentry (see Box 38). Therefore I propose to close the issue.

Requestor: John Hinke (hinke@roguewave.com)

Issue Number: 27-402
Title: examples for ipfx
Section: 27.6.1.1.2 **basic_istream prefix and suffix [lib.istream.prefix]**
Status: **active**
Description:

The example for a "typical" implementation of ipfx() has an incorrect function declaration. It should read:

template<class charT, class traits>
bool basic_istream<charT, traits>::ipfx(bool *noskipws*)

Possible Resolution:

The Tokyo meeting deprecated ipfx and isfx in favor of the member type sentry (see Box 38). Therefore I propose to close the issue.

Requestor: John Hinke (hinke@roguewave.com)

Issue Number: 27-403
Title: Clarification of exceptions thrown
Section: 27.6.1.2.2 **basic_istream::operator>> [lib.istream::extractors]**
Status: **active**
Description:

27.6.1.2.2 paragraph 4 says
"If one of these called functions throws an exception, then unless noted otherwise the input function calls setstate(badbit) and if badbit is on in exception() (sic) rethrows the exception without completing its actions."

Problem: If badbit is on in exceptions() then ios_base::clear, which is called by setstate(badbit), will throw an object of ios_base::failure and the original exception will NEVER be rethrown, i.e., it will be lost.

Discussion:

Jerry Schwarz,

"This has been discussed a lot. My preference has always been that if any of the virtuals throws an exception then

a) set badbit in error state
b) check badbit in exception state
    b1) if its on then rethrow the original exception
    b2) do not throw anything, treat as an error.

"Other implementors have complained that this was hard to do, and have preferred to just let the exception be passed through without being caught at all.

"Other people think that all iostream operations should only through ios_base::failure."

Possible Resolution:

See issue 27-504.

Requestor: Modena Software (modena@netcom.com)

Issue Number: 27-404
Title: istream functions need to check for NULL streambuf
Section: 27.6.1.1 **Template class basic_istream [lib.istream]**
Status: **active**
Description:

Functions in basic_istream that call members of rdbuf() need to check for a NULL streambuf before calling the function. There are some functions that make sure rdbuf() is not a NULL pointer before calling any functions on the buffer, but some functions don't check for the NULL pointer. This needs to be consistent.

Discussion:

P.J. Plauger wrote: "Any attempt to store a null stream buffer pointer causes badbit to be set in the stored status. Hence, no input or output is ever attempted, using such a pointer, by formatted functions."

Possible Resolution:

As pointed out by P.J. Plauger, we should add a footnote to explain why there is no need to check for a NULL streambuf.

We should also add, in section 27.4.4.2 **Member functions [lib.basic.ios.members],** the following to the description of basic_streambuf<charT,traits>* rdbuf(basic_streambuf<charT,traits>* sb); :

Postcondition: sb == rdbuf() and if sb is a NULL pointer rdstate() == badbit.

Note: This issue has to be discussed with issue 27-503.

Requestor: John Hinke (hinke@roguewave.com)

Issue Number: 27-405
Title: confusing English in formatted requirements
Section: 27.6.1.2.1 **Common requirements [lib.istream.formatted.reqmts]**
Status: **active**
Description:

27.6.1.2.1 [lib.istream.formatted.reqmts]: Paragraph 5: "In case the converting result is a value of either an integral type ... or a float type ... performing to parse and convert the result depend on the imbued locale object." This is really French converted to English by translation software, right? :->}

Possible Resolution:

The imbued locale object is responsible for parsing and converting the result when extracting an integral type (short, unsigned short, int, unsigned int, long, unsigned long) or a float type (float, double, long double). So the behavior of the above type extractors are locale-dependent. The imbued locale object uses an istreambuf_iterator to access the input character sequence.

Requestor: Public Comment

Issue Number: 27-406
Title: operator>>(char_type *) failure
Section: 27.6.1.2.2 **basic_istream::operator>> [lib.istream::extractors]**
Status: **active**
Description:

27.6.1.2.2 [lib.istream::extractors]: Paragraph 2: "If the function stores no characters, it calls setstate(failbit), which may throw ios_base::failure (27.4.4.3). In any case, it then stores a null character ...." How can it store anything if an exception is thrown? C++ does not use the resumption model for exception handling. Different language than "In any case" is needed here.

Possible Resolution:

Change paragraph 2 to:

“If the function stores no characters, it calls setstate(failbit), which may throw ios_base::failure (27.4.4.3).”

Add paragraph 3:

“Before returning or throwing an exception the function stores a null character into the next successive location of the array and calls width(0).”

Requestor: Public Comment

Issue Number: 27-407
Title: operator>>(char_type) failure
Section: 27.6.1.2.2 **basic_istream::operator>> [lib.istream::extractors]**
Status: **active**
Description:

27.6.1.2.2 [lib.istream::extractors]: Paragraph 2:

basic_istream<charT,traits>& operator>>(char_type& c);

Effects: Extracts a character, if one is available, and stores it in c. Otherwise, the function calls setstate(failbit).

Not eofbit?

Possible Resolution:

In 27.6.1.2.1 **Common requirements [lib.istream.formatted.reqmts]** paragraph 8 says:

"If the scan fails for any reason, the formatted input function calls setstate(failbit), which may throw ios_base::failure (27.4.4.3)."

This is one of the requirements for all the formatted input functions. Because of this the user can call the ios_base member function fail( ) or the operator bool ( ) to check if the extraction failed. The user can therefore write code like this:

```
if ( in >> s )
        { perform some action }
```

Requestor: Public Comment

Issue Number: 27-408
Title: ws manipulator
Section: 27.6.1.4 **Standard basic_istream manipulators [lib.istream.manip]**
Status: **active**
Description:

27.6.1.4 [lib.istream.manip]: "... saves a copy of is.fmtflags ...."
Should this not read "... saves a copy of is.flags ...."?

Possible Resolution:

The **Effects:** clause should be changed to:

"**Effects:** Skips any white space in the input sequence. Saves a copy of the fmtflags by storing the result of the call to is.flags( ), calls is.setf(ios_base::skipws), then constructs a sentry object and restores the fmtflags to their saved values."

Requestor: Public Comment

Issue Number: 27-409
Title: unsigned short extractors cannot use unsigned long get function
Section: 27.6.1.2.2 basic_istream ::operator>> **[lib.istream::extractors]**
Status: **active**
Description:

Unsigned short (and unsigned int) extractors cannot use unsigned long get function in num_get. It cannot distinguish certain valid inputs from errors.

Possible Resolution:

P.J. Plauger wrote: "num_get should add a get function (and underlying do get) for unsigned short and unsigned int extractions. Otherwise, input values in the range -1 through -USHRT_MAX (or -UINT_MAX) look erroneous, and cannot be distinguished from truly erroneous values."

Requestor: P.J. Plauger (plauger!pjp@uunet.uu.net)

---

basic_ostream issues

Issue Number: 27-501
Title: ostream<<(char) : formatting, padding, width
Section: 27.6.2.4.2 **basic_ostream::operator<< [lib.ostream.inserters]**
Status: active
Description:

For historical reasons, this function has usually ignored padding and formatting. In the WP, it does not mention anything about ignoring padding or formatting. This needs to be clarified.

Reasons for ignoring padding on op<<(char):

1. Historical reasons/compatibility

Reasons for full formatting on op<<(char):

1. put(char) currently does no formatting. But there is no way to insert a char with formatting.
2. Some implementations do formatting.

Since put can insert a character without formatting, there needs to be a way to insert a character with formatting. Currently this does not exist. It would be nice not to introduce an inconsistency with the other formatted inserters, but it would also be nice to provide compatibility. I think that consistency would be much better in this case than compatibility.

Possible Resolution:

At the Tokyo meeting the straw vote gave the following result:

5 for past practice (no padding), 1 for consistency.

We should organize another straw vote at the next meeting and if the result matches the one above, close the issue.

Requestor: John Hinke (hinke@roguewave.com),
Bernd Eggink (admin@rrz.uni-hamburg.de)

Issue Number: 27-502
Title: ostream::operator<<(void *)
Section: 27.6.2.4.2 **basic_ostream::operator<< [lib.ostream.inserters]**
Status: **Active**
Description:

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

should take 'const volatile void *' rather than void *.

Resolution:

The function now takes a const void *.

Reopened:

Does anyone know why the resolution was for it to take a const void * rather than a const volatile void *?

I can't think of any good reason why we should make the code:

```
#include <iostream>
volatile int x;
int main() {
    cout << & x;
    return 0;
}
```

ill-formed.

Possible Resolution:

We need to change basic_ostream<charT,traits>& operator<<(void *) to basic_ostream<charT,traits>& operator<<(const volatile void *) to avoid breaking the code above, but also because of issue 27-203. If we adopt issue 27-203 and we do not make the change described above we will end up with the following:

```
volatile int x;
cout << &x;
```

This will call operator const void*( ) which will return !fail( ) and then cout the result.

Requestor:	Fergus Henderson (fjh@munta.cs.mu.oz.au) Philippe Le Mouël (philippe@roguewave.com)

Issue Number:	27-503
Title:	ostream functions need to check for NULL streambuf
Section:	27.6.2.1 **Template class basic_ostream [lib.ostream]**
Status:	**active**
Description:	

Functions in basic_ostream that call members of rdbuf() need to check for a NULL streambuf before calling the function. There are some functions that make sure rdbuf() is not a NULL pointer before calling any functions on the buffer, but some functions don't check for the NULL pointer. This needs to be consistent.

Discussion:

P.J. Plauger wrote: "Any attempt to store a null stream buffer pointer causes badbit to be set in the stored status. Hence, no input or output is ever attempted, using such a pointer, by formatted functions."

Possible Resolution:

As pointed out by P.J. Plauger we should add a footnote to explain why there is no need to check for a NULL streambuf.

We should also add in section 27.4.4.2 **Member functions [lib.basic.ios.members]** the following to the description of basic_streambuf<charT,traits>* rdbuf(basic_streambuf<charT,traits>* sb); :

Postcondition: sb == rdbuf() and if sb is a NULL pointer rdstate() == badbit.

Note: This issue has to be discussed with issue 27-404.

Requestor: John Hinke (hinke@roguewave.com)

Issue Number: 27-504
Title: exceptions in ostream
Section: 27.6.2.4.2 **basic_ostream::operator<< [lib.ostream.inserters]**
Status: **active**
Description:

In basic_ostream::operator<<(basic_streambuf<charT,traits>* sb), the last line of Effects paragraph 3 can't happen. Previous sentence says that if ``an exception was thrown while extracting a character, it calls setstate(failbit) (which may throw ios_base::failure)." Then the last sentence says, ``If an exception was thrown while extracting a character and failbit is on in exceptions() the caught exception is rethrown." But in this case, setstate has already thrown ios_base::failure. Besides, I can find no committee resolution that calls for exceptions() to be queried in this event. And an earlier sentence says unconditionally that the exception is rethrown. Last sentence should be struck.

Discussion:

This issue and issue 27-403 are both related to the exception-handling mechanism in iostream. The problem is that the WP is not clear about which policy we are supposed to implement. Here is an example where the user is deriving his own stream buffer and istream object:

```
template <class charT, class traits>
class mon_buffer : public basic_streambuf<charT, traits>
{
    .
    .
    .

protected:

    virtual int_type overflow( int_type c = traits::eof() )
       { implementation }
     virtual int_type underflow( )
      {
        .
        .
        if ( something goes wrong )  throw mon_exception;   // mon_exception is a user's class
        .
        .
      }

    .
```

```
    .
    .
};

template <class charT, class traits>
class mon_istream : public basic_istream<charT,traits>
{
    .
    .
    .

  public:

    mon_buffer<charT,traits> *rdbuf( ) const
    { implementation}
    .
    .
    .
}
```

Here is the user main:

```
void main( )
 {
   try {
    .
    .
    mon_istream in( parameters );
    cout << in.rdbuf();
    .
    .
      }

  catch ( mon_exception op )
   {
     // do something about it
   }
 }
```

The line cout << in.rdbuf(); callls the function basic_ostream::operator << (basic_streambuf<charT,traits> * sb), which outputs the content of mon_buffer to stdout.
A problem arise if, when reading characters from mon_buffer, one of the underflow calls results in throwing mon_exception. In this case there are several possibilities for the function basic_ostream::operator << (basic_streambuf<charT,traits> * sb) to handle the problem:

1) The function does nothing and the exception is caught by the user. The problem with this approach is that the cout object never get its failbit set.

2) The function catches the exception, calls setstate(failbit) and rethrows the exception. The problem here is that if failbit is on in exceptions( ), the call to setstate(failbit) will result in throwing ios_base::failure and not rethrowing the previous exception.

3) The function catches the exception, calls setstate(failbit), catches ios_base::failure if it is thrown by the previous call to setstate(failbit), and then rethrows the exception.
4) The function catches the exception, calls setstate(failbit), catches ios_base::failure if it is thrown by the previous call to setstate(failbit), then if ios_base::failure was thrown, rethrows the previous exception, otherwise treats as an error.

Possible Resolution:

As first pointed out by Jerry Schwarz, in issue 27-403, there are different ways of implementing the exception mechanism in iostream. My own preference is the fourth possibility I described above. If we chose this scheme to handle exceptions in iostream, functions like basic_ostream::operator << (basic_streambuf<charT,traits> * sb) will look like this:

```
template <class charT, traits>
basic_ostream<charT, traits>&
basic_ostream<charT, traits>::operator<<(basic_streambuf<charT,traits> *sb)
{
    try {
            //function implementation
        }

   catch ( … )
    {
       bool flag = FALSE;
       try {
             setstate(failbit);
           }
       catch ( ios_base::failure ) { flag = TRUE; }
       if ( flag ) throw;
    }
}
```

Requestor: Public Comment

Issue Number: 27-505
Title: incorrect conversion specifier for operator<<(unsigned long)
Section: 27.6.2.4.2 **basic_ostream::operator<< [lib.ostream.inserters]**
Status: **active**
Description:

basic_ostream<charT,traits>& operator<<(unsigned long n);

Effects: Converts the unsigned long integer n with the integral conversion specified preceded by l.

Should this be "... preceded by ul."?

Possible Resolution:

The **Effects:** clause says:

“**Effects:** Converts the unsigned long integer n with the integral conversion specifier preceded by l.”

To me this is correct, but it may be not precise enough. The integral conversion specifier can be “d” for a signed integral type and “u” for a unsigned integral type. If we decide to be precise about this fact in the Effects clause, we will have to do the same for all the other unsigned inserters.

Requestor: Public Comment

Issue Number: 27-506
Title: wrong default behavior for padding
Section: 27.6.2.4.1 Common requirements Table 13 Fill padding **[lib.ostream.formatted.reqmts]**
Status: **active**
Description:

27.6.2.4.1 Table13 Fill padding changes the long-standing default behavior for padding output field. It has always been true that setting none of left, right, and internal called for left padding (pad after text). Now it calls for right padding (pad before text). Since this is the initial state of all ios objects, many simple C++ programs will change behavior.

Possible Resolution:

P.J. Plauger wrote: “Table 13 should describe the effect of right/internal/otherwise, as it has long been, rather than left/internal/otherwise. Change was originally unauthorized, then endorsed (I hope by accident) at the July ’95 meeting.”

I tested the default padding by compiling the following code:

“cout << setw(10) << setfill(‘@’) << “test” << endl; ”

With the following old iostream library:

- AT&T Release 3.0
- Borland C/C++ Run Time Library - Version 6.5

The result was right padding (pad before text) for all of them.

Therefore I think the current behavior is correct.

Requestor: P.J. Plauger (plauger!pjp@uunet.uu.net)

---

basic_istream/basic_ostream issues

Issue Number: 27-601
Title: istream::operator>>(ios_base&), ostream::operator<<(ios_base&)
Section: 27.6.1.2.2 **basic_istream::operator>> [lib.istream::extractors],** 27.6.2.4.2 **basic_ostream::operator<< [lib.ostream.inserters]**
Status: active
Description:

The ios_base manipulators 27.4.5.1**[lib.std.ios.manip]** will not work as written. They won't work because there is no conversion from ios_base to basic_ios.

They are currently declared as:
ios_base& boolalpha(ios_base&);

I propose adding a new insertor/extractor for istream and ostream that does insertion/extraction for ios_base.

Possible Resolution:

John wrote:

"Add to basic_istream:

basic_istream<charT, traits>& operator>>(ios_base& (*pf)(ios_base&));

Effects: Calls (*pf)(*this)
Returns: *this.

Add to basic_ostream:

basic_ostream<charT, traits>& operator<<(ios_base& (*pf)(ios_base&));

Effects: Calls (*pf)(*this)
Returns: *this.

Also, several footnotes will need to be changed."

We need to change footnote 9 in 27.4.5.3 **basefield manipulators [lib.basefield.manip]** to:

"The function signature dec(ios_base& str) can be called by the function signature basic_ostream<charT,traits>& basic_ostream<charT,traits>::operator << (ios_base& (*)(ios_base&)) to permit expressions of the form cout << dec to change the format flags stored in cout."

Requestor: John Hinke (hinke@roguewave.com)

Issue Number: 27-602
Title: positional typedefs in istream/ostream derived classes
Section: 27
Status: **active**

Description:

Remove the positional typedefs from the following classes. The positional typedefs are:

typedef traits::pos_type pos_type;
typedef traits::off_type off_type;

They are not used in the following classes:

basic_istringstream
basic_ostringstream
basic_ifstream
basic_ofstream

Possible Resolution:

John wrote:

"Remove them. They are still inherited from the base classes."

I do not think that they are inherited from the base classes (see typename discussions).

Requestor: John Hinke (hinke@roguewave.com)

Issue Number: 27-603
Title: istream::read, ostream::write
Section: 27.6.1.3 **Unformatted input functions [lib.istream.unformatted]**, 27.6.2.5 **Unformatted output functions [lib.ostream.unformatted]**
Status: **active**
Description:

basic_istream<charT,traits>& basic_istream<charT,traits>::read(char_type *,streamsize);
basic_ostream<charT,traits>& basic_ostream<charT,traits>::write(const char_type *,streamsize);

These functions are typically used for binary data.

Possible Resolution:

John wrote:

"These functions should take a void * instead of char_type *. If these functions are changed, then perhaps we should add another function that replaces this behavior. basic_istream currently has a get function, which behaves like the read and write functions. It would make sense to add a corresponding put function in basic_ostream that parallels the behavior of get."

I think we should let these functions remain the way they are, because no other function performs the exact same task (see issue 27-103).The get function in basic_istream does not behave like the read function, it takes an extra parameter, and if this parameter is equal to the current read character, the function does not read any more characters. The question becomes, do we need to add functions taking a void* parameter ? They could be useful if you want to insert or extract binary data from a wide characters stream. In this case, the classic read and write functions are not sufficient, because the size of the data to be extracted or inserted has to be a multiple of the character size. The problem is that the underlying streambuf is using charT type and if you want to move inside the streambuf or perform read or write operations, they will have to be done by multiples of the charT size. The question therefore becomes, is the price to add these two functions too high ?

Requestor: John Hinke (hinke@roguewave.com)

Issue Number: 27-604
Title: Opening an istream without ios::in set? or an ostream without ios::out set?
Section: 27.6.1.1 **Template class basic_istream [lib.input.streams],**
27.6.2.1 **Template class basic_ostream [lib.output.streams]**
Status: **active**
Description:

Benedikt asks,

"Why can I open an istream without ios::in being set or an ostream without ios::out? I mean, I just did that by mistake with an ofstream and searched for quite a while to find out, why there were no actual writes to the newly created file.

"Or, even worse, why can I open an istream with ios::out (and no ios::in) being set and vice versa?

"Shouldn't the iostreams check whether the given mode flags make any sense, and maybe even add ios::in if you missed to set this in an istream, or ios::out if you used an ostream?"

Possible Resolution:

The only way to create an istream or ostream object is by calling the constructor "explicit basic_istream(basic_streambuf<charT,traits>* sb);" for istream and "explicit basic_ostream(basic_streambuf<charT,traits>* sb);" for ostream. At this point an implementation should do something like:

In basic_istream constructor:

```
if ( sb->which_open_mode() & ios_base::in )
  init(sb);
else
  init(0);
```

In basic_ostream constructor:

```
if ( sb->which_open_mode() & ios_base::out )
  init(sb);
else
  init(0);
```

But the actual open mode is really set up in the buffer, which can be basic_stringbuf, basic_filebuf or strstreambuf according to the kind of object you are using.
In the draft, it is clear that whenever you create an object of type basic_ifstream, basic_istringstream or istrstream the buffer's open mode is set to "in" and when you create an object of type basic_ofstream, basic_ostringstream, or ostrstream, the buffer's open mode is set to "out" (see constructor description for all these objects). Therefore a correct implementation will not allow the behavior described above by Benedikt.

Requestor: Benedikt Erik Heinen (beh@tequila.oche.de)

Issue Number: 27-605
Title: get/put type functions should be able to use iterators.
Section: 27.6.1.3 **Unformatted input functions [lib.istream.unformatted]**
27.6.2.5 **Unformatted output functions [lib.ostream.unformatted]**
Status: **active**
Description:

Several functions in istream and ostream take a pointer and a length and optionally a delimiter. It would be nice to add overloaded functions that take either InputIterators, or OutputIterators. These new functions would look like:

For basic_istream:

template<class OutputIterator>
istream& get(OutputIterator *begin*, OutputIterator *end*, char_type *delim*);

The *begin* and *end* iterators define where the characters will be written. Characters will be read from the sequence until the *end* iterator is reached, or the next character is *delim*.

For basic_ostream:

template<class InputIterator>
ostream& write(InputIterator *begin*, InputIterator *end*);

The *begin* and *end* iterators define the sequence of characters to be written.

These functions would be added to the current implementation. The current set of functions should not be removed. They are very commonly used. There are several functions which are candidates for these *begin* and *end* iterators. These functions are:

For basic_istream:

istream& get(char_type *, streamsize, char_type);
istream& getline(char_type *, streamsize, char_type);
istream& read(char_type *, streamsize);

For basic_ostream:

ostream& put(char_type *, streamsize);
ostream& write(void *, streamsize);

Possible Resolution:

I do not think it is really necessary. We should have a vote to decide if we want to adopt this change or not.

Requestor: Nathan Myers (ncm@cantrip.org)

Standard manipulators issues

Issue Number: 27-651
Title: setfill description is wrong
Section: 27.6.3 Standard manipulators **[lib.std.manip]**
Status: **active**
Description:

P.J. Plauger wrote: "Setfill description is nonsense, since a fill character is now a charT, which cannot necessarily be represented as type int. Nor can it be applied to ios_base, since the fill character now inhabits basic_ios."

Possible Resolution:

setfill should be changed to:

```
template <class charT>
smanip<charT> setfill ( charT c );
```

Returns: smanip<charT>(f,c) where f can be defined as:

```
template <class charT>
basic_ios<charT,ios_traits<charT>>& f ( basic_ios<charT,ios_traits<charT>>& str, charT c)
{  // set fill character
   str.fill ( c );
   return str;
}
```

Requestor: P.J. Plauger (plauger!pjp@uunet.uu.net)
Philippe Le Mouël (philippe@roguewave.com)

Issue Number: 27-652
Title: smanip is not a single type
Section: 27.6.3 Standard manipulators **[lib.std.manip]**
Status: **active**
Description:

P.J. Plauger wrote: "Description of manipulators strongly suggests that smanip is a single type. It was supposed to make clear that each manipulator can return a different type, as needed. (And more than one type is certainly needed here.)"

Possible Resolution:

27.6.3 standard manipulators paragraph 2 says: "The type **smanip** is an implementation-defined type (_dcl.fct_) returned by the standard manipulators.". We need to rewrite this sentence to make it clear that smanip is not restrained to one physical type.

Requestor: P.J. Plauger (plauger!pjp@uunet.uu.net)
Philippe Le Mouël (philippe@roguewave.com)

---

string stream issues

Issue Number: 27-701
Title: basic_stringbuf::str() needs to clarify return value on else clause
Section: 27.7.1.2 **Member functions [lib.stringbuf.members]**
Status: **active**
Description:

“Table 15 in [lib.stringbuf.members] describes the return values of basic_stringbuf::str(). What does the "otherwise" mean?. Does it mean neither ios_base::in nor ios_base::out is set? What is the return value supposed to be if _both_ bits are set?”

Possible Resolution:

My understanding is that if both ios_base::in and ios_base::out are set, you should return basic_string<char_type>(eback(),egptr()-eback()). I propose to change the **Returns:** clause to clarify this fact.

Returns: The return values of this function are indicated in Table 15 and the test that determine these values are carried out in the order shown in Table 15.

Requestor: Angelika Langer (langer@roguewave.com)
Bernd Eggink (admin@rrz.uni-hamburg.de)

Issue Number: 27-702
Title: string streams need allocator and string_char_traits parameters
Section: 27.7.1 **Template class basic_stringbuf [lib_stringbuf]**
Status: **active**
Description:

The string streams are currently templatized on the character type (charT) and the traits type (ios_traits). String template parameters need to be added.

Possible Resolution:

John wrote:

“I propose to change the template parameters of the string streams from:

```
    template<class charT, class traits = ios_traits<charT> >
to:
    template<class charT, class IOS_traits = ios_traits<charT>,
        class STRING_traits = string_char_traits<charT>,
        class Allocator = allocator>
```

All references to basic_string, or any of the string stream classes will need to be fixed.

All references to traits should be replaced by either IOS_traits or STRING_traits.”

I do not see a good reason for this change and anyway, isn’t it too late ?

Requestor: John Hinke (hinke@roguewave.com)

Issue Number: 27-703
Title: stringbuf postconditions
Section: 27.7.1.2 **Member functions [lib.stringbuf.members]**
Status: **active**
Description:

basic_stringbuf::str(basic_string s) Postconditions requires that str() == s. This is true only if which had in set at construction time. Condition should be restated.

Possible Resolution:

I think the real problem is in "Table 16 - str get/set areas". Its second line says:

(which & ios_base::out) != 0 setp(str(),str(),str()+str.size())

First, the function setp takes only two parameters. Furthermore it should say:

(which & ios_base::out)!= 0 setp(str(),str()+s.size())
then if: (which & ios_base::app)!=0 pbump(s.size())

Then the postcondition requiring that str() == s in the function void str(const basic_string<char_type>& s) will be valid if "in" or "out" and "app" are set at construction time.

Table 16 should be changed to:

Table 16—str get/set areas

Condition	Setting
(which & ios_base::in)!=0	setg(str(), str(), str()+s.size())
(which & ios_base::out)!=0	setp(str(),str()+s.size())
(which & ios_base::app)!=0	pbump(s.size())

The postcondition should be changed to:

Postcondition: if ios_base::in, or ios_base::out and ios_base::app are set at construction time, then str()==s. Otherwise str() == basic_string<char_type>(). If s.size() >0, set the get and/or put pointers as indicated in Table 16.

Requestor: Public Comment

Issue Number: 27-704
Title: stringbuf::stringbuf constructor
Section: 27.7.1.1 **basic_stringbuf constructors [lib.stringbuf.cons]**
Status: **active**
Description:

basic_stringbuf::basic_stringbuf(basic_string str, openmode which) Postconditions requires that str() == str. This is true only if which has in set. Condition should be restated.

Possible Resolution:

This is the same problem described in issue 27-703.

The real problem is in “Table 14 - str get/set areas”. The second line says:

(which & ios_base::out) != 0 setp(str(),str(),str()+str.size())

First, the function setp takes only two parameters. Furthermore it should say:

(which & ios_base::out)!= 0 setp(str(),str()+str.size())
then if: (which & ios_base::app)!=0 pbump(str.size())

Then the postcondition requiring that str() == str in the function basic_stringbuf:: basic_stringbuf(basic_string str, openmode which) will be valid if “in” or “out” and “app” are set.

Table 14 should be changed to:

Table 14—str get/set areas

Condition	Setting
(which & ios_base::in)!=0	setg(str(), str(), str()+str.size())
(which & ios_base::out)!=0	setp(str(),str()+str.size())
(which & ios_base::app)!=0	pbump(str.size())

The postcondition should be changed to:

Postcondition: if ios_base::in, or ios_base::out and ios_base::app are set at construction time, then str()==str. Otherwise str() == basic_string<char_type>(). If str.size() >0, set the get and/or put pointers as indicated in Table 14.

Requestor: Public Comment

Issue Number: 27-705
Title: Incorrect calls to setg and setp in Table 14
Section: 27.7.1.1 **basic_stringbuf constructors [lib.stringbuf.cons]**
Status: **active**
Description:

Table 14 describes calls to setg and setp with string arguments, for which no signature exists. Needs to be recast.

Possible Resolution:

Possible Resolution of issue 27-704 solves this problem.

Requestor: Public Comment

Issue Number: 27-706
Title: Incorrect calls to setg and setp in table 16
Section: 27.7.1.2 **Member functions [lib.stringbuf.members]**
Status: **active**
Description:

Table 16 describes calls to setg and setp with string arguments, for which no signature exists. Needs to be recast.

Possible Resolution:

Possible Resolution of issue 27-703 solves this problem

Requestor: Public Comment

Issue Number: 27-707
Title: setbuf function is missing
Section: 27.7.1 **Template class basic_stringbuf [lib.stringbuf]**
Status: **active**
Description:

Steve Clamage wrote: “Section 27.7.1.3 should have a basic_stringbuf override of the base class setbuf() function, but it is missing.”

Possible Resolution:

Add the following description in 27.7.1 **Template class basic_stringbuf [lib.stringbuf]** and 27.7.1.3 **Overridden virtual functions [lib.stringbuf.virtuals]**:

basic_streambuf<charT,traits>* setbuf(char_type* s, int n);

Effects: If *(mode & ios_base::out)* is true, proceed as follows:
If *s* is not a null pointer, and n > pptr() - pbase(), replace the current buffer (copy its contents and deallocate it) by the buffer of size *n* pointed at by *s*.
In the case where *s* is a null pointer, and n > pptr() - pbase() resize the current buffer to size *n*.
If the function fails, it returns a null pointer.
Returns: (basic_streambuf<charT,traits>*)(this)

I am not qualified enough to decide if the return type should be changed to basic_stringbuf<charT,traits>* as proposed by Steve Clamage in issue 27-809. I tried it with several compilers, and the results were just error messages. Basically, the compilers were complaining about the fact that the base class virtual function and the overridden virtual function should have the same return type.

Requestor: Steve Clamage (stephen.clamage@eng.sun.com)

---

file stream issues

Issue Number: 27-801
Title: filebuf::underflow example
Section: 27.8.1.4 **Overridden virtual functions [lib.filebuf.virtuals]**
Status: **active**
Description:

The "as if" example for basic_filebuf::underflow has several "typos". It should say:

```
char    from_buf[FSIZE};
char*   from_end;
char    to_buf[TSIZE};
char*   to_end;
typename traits::state_type st;

codecvt_base::result r =
      getloc().template use<codecvt<char, charT,
      typename traits::state_type> >().convert
      (st, from_buf, from_buf+FSIZE, from_end,
      to_buf, to_buf+TSIZE, to_end);
```

Possible Resolution:

We should correct the example as follows, and not as described above:

```
char    from_buf[FSIZE};
char*   from_end;
charT   to_buf[TSIZE};
charT*  to_end;
typename traits::state_type st;

codecvt_base::result r=
   use_facet<codecvt<char,charT,typename ios_traits::state_type> >(getloc()).
      convert(st,from_buf,from_buf+FSIZE,from_end,to_buf,to_buf+to_size,to_end);
```

Requestor: John Hinke (hinke@roguewave.com)

Issue Number: 27-802
Title: filebuf::is_open is a bit confusing
Section: 27.8.1.3 **Member functions [lib.filebuf.members]**
Status: **active**
Description:

It says, "**Returns:** true if the associated file is available and open." What is the meaning of available? This seems a bit confusing.

Possible Resolution:

Change the **Returns:** statement to:

Returns: true after a successful call to the member function open, and before a successful call to member function close, otherwise false.

Requestor: John Hinke (hinke@roguewave.com), Bob Kline (bkline@cortex.nlm.nih.gov)

Issue Number: 27-803
Title: ofstream constructor missing trunc as openmode
Section: 27.8.1.9 **basic_ofstream constructors [lib.ofstream.cons]**
Status: **active**
Description:

basic_ofstream::basic_ofstream(const char *s, openmode mode = out) has wrong default second argument. It should be `out | trunc', the same as for basic_ofstream::open (in the definition at least).

Possible Resolution:

In my version of the WP (30 November 1995) both basic_ofstream::basic_ofstream(const char *s, openmode mode = out | trunc) and basic_ofstream::open(const char* s, openmode mode=out | trunc) take the same second argument default value out | trunc. Therefore we should close the issue.

Requestor: Public Comment

Issue Number: 27-804
Title: ofstream::open missing trunc in openmode
Section: 27.8.1.10 **Member functions [lib.ofstream.members]**
Status: **active**
Description:

basic_ofstream::open(const char *s, openmode mode = out) has wrong default second argument. It should be `out | trunc', the same as for basic_ofstream::open in the definition.

Possible Resolution:

See issue 27-803.

Requestor: Public Comment

Issue Number: 27-805
Title: filebuf::imbue semantics
Section: 27.8.1.4 **Overridden virtual functions [lib.filebuf.virtuals]**
Status: **active**
Description:

basic_filebuf::imbue has silly semantics. Whether or not sync() succeeds has little bearing on whether you can safely change the working codecvt facet. The most sensible thing is to establish this facet at construction. (Then pubimbue and imbue can be scrubbed completely.) Next best is while is_open() is false. (Then imbue can be scrubbed, since it has nothing to do.) Next best is to permit any imbue that doesn't change the facet or is at beginning of file. Next best is to permit change of facet any time provided either the current or new facet does not mandate state-dependent conversions. (See comments under seekoff.)

Possible Resolution:

Requestor: Public Comment

Issue Number: 27-806
Title: filebuf::seekoff **Effects:** clause needs work
Section: 27.8.1.4 **Overridden virtual functions [lib.filebuf.virtuals]**
Status: **active**
Description:

basic_filebuf::seekoff Effects is an interesting exercise in creative writing. It should simply state that if the stream is opened as a text file or has state-dependent conversions, the only permissible seeks are with zero offset relative to the beginning or current position of the file. (How to determine that predicate is another matter -- should state for codecvt that even a request to convert zero characters will return noconv.) Otherwise, behavior is largely the same as for basic_stringstream, from whence the words should be cribbed. The problem of saving the stream state in a traits::pos_type object remains unsolved. The primitives described for ios_traits are inadequate.

Possible Resolution:

Requestor: Public Comment

Issue Number: 27-807
Title: filebuf::underflow performance questions
Section: 27.8.1.4 **Overridden virtual functions [lib.filebuf.virtuals]**
Status: **active**
Description:

basic_filebuf::underflow is defined unequivocally as the function that calls codecvt, but there are performance advantages to having this conversion actually performed in uflow. If the specification cannot be broadened sufficiently to allow either function to do the translation, then uflow loses its last rationale for being added in the first place. Either the extra latitude should be granted implementors or uflow should be removed from basic_streambuf and all its derivatives.

Possible Resolution:

To me underflow is also called by uflow, so it is simple to make the actual call to the codecvt facet in underflow.

Requestor: Public Comment

Issue Number: 27-808
Title: Editorial fixes in wording for fstreams
Section: 27.8.1 **File streams [lib.fstreams]**
Status: **active**
Description:

27.8.1 [lib.fstreams], paragraph 2: "... the type name FILE is a synonym for the type FILE." This seems like an odd sort of synonym, doesn't it? Also, the last sentence of this subsection, "Because of necessity of the conversion between the external source/sink streams and wide character sequences." is incomplete.

Possible Resolution:

Requestor: Public Comment

Issue Number: 27-809
Title: Description of function setbuf is missing
Section: 27.8.1.4 **Overridden virtual functions [lib.filebuf.virtuals]**
Status: **active**
Description:

Steve Clamage wrote: “basic_filebuf version of setbuf( ) needs a description, and the return type shown in the draft is basic_streambuf*, which is probably wrong. It was correct before covariant return types were added to the draft. Now it should probably return basic_filebuf*.”

Possible Resolution:

Add the following description in 27.8.1.4 **Overridden virtual functions [lib.filebuf.virtuals]**:

basic_streambuf<charT,traits>* setbuf(char_type* s, int n);

Effects: If *s* is not a null pointer, flush the buffer by calling *overflow(traits::eof())* and if the return value is not *traits::eof()*, deallocate the current buffer and replace it by the buffer of size *n* pointed at by *s*.
In the case where *s* is a null pointer, resize the current buffer to size *n* (this can result in flushing the buffer).
If the function fails, it returns a null pointer.
Returns: (basic_streambuf<charT,traits>*)(this)

I am not qualified enough to decide if the return type should be changed or not as proposed by Steve Clamage. I tried it with several compilers, and the results were just error messages. Basically, the compilers were complaining about the fact that the base class virtual function and the overridden virtual function should have the same return type.

Requestor: Steve Clamage (stephen.clamage@eng.sun.com)

Issue Number: 27-810
Title: Openmode notation is not consistent in basic_ifstream and basic_ofstream
Section: 27.8.1.5 **Template class basic_ifstream [lib.ifstream]**
27.8.1.8 **Template class basic_ofstream [lib.ofstream]**
Status: **active**
Description:

basic_ifstream, basic_ofstream *constructors* and member functions *open* describe the type ios_base::openmode as openmode and its values as *in* and *out* rather than *ios_base::in* and *ios_base::out* as everywhere else in the library.

Possible Resolution:

In 27.8.1.5 **Template class basic_ifstream [lib.ifstream]** , 27.8.1.6 **basic_ifstream constructors [lib.ifstream.cons]** and 27.8.1.7 **member functions [lib.ifstream.members]** change the following functions:

explicit basic_ifstream(const char* s, openmode mode = in);

to:

explicit basic_ifstream(const char* s, ios_base::openmode mode = ios_base::in);

void open(const char* s, openmode mode = in);

to:

void open(const char* s, ios_base::openmode mode = ios_base::in);

In 27.8.1.8 **Template class basic_ofstream [lib.ofstream]** , 27.8.1.9 **basic_ofstream constructors [lib.ofstream.cons]** and 27.8.1.10 **member functions [lib.ofstream.members]** change the following functions:

explicit basic_ofstream(const char* s, openmode mode = out | trunc);

to:

explicit basic_ofstream(const char* s, ios_base::openmode mode = ios_base::out | ios_base::trunc);

void open(const char* s, openmode mode = out | trunc);

to:

void open(const char* s, ios_base::openmode mode = ios_base::out | ios_base::trunc);

Requestor: Philippe Le Mouël (philippe@roguewave.com)

Issue Number: 27-811
Title: Description of function sync is missing
Section: 27.8.1.4 **Overridden virtual functions [lib.filebuf.virtuals]**
Status: **active**
Description:

Description of the overridden sync( ) function in class basic_filebuf is missing.

Possible Resolution:

Add the following description in 27.8.1.4 **Overridden virtual functions [lib.filebuf.virtuals]**:

int sync( );

Effects: If *pbase( )* is non-null calls *overflow(traits::eof())*, which outputs the content of the buffer to the associated file.
Returns: If the call to *overflow* returns *traits::eof()*, returns -1 to indicate failure, otherwise returns 0.

Requestor: Philippe Le Mouël (philippe@roguewave.com)

Miscellaneous issues

Issue Number: 27-901
Title: input/output of unsigned charT
Section: 27
Status: **active**
Description:

NOTE: istream here means basic_istream.
ostream here means basic_ostream.

This issue details all of the issues with inserting or extracting characters.

Currently, IOStreams does not allow the insertion/extraction of unsigned charT or signed charT. There are two types of functions that could insert or extract these character types: formatted IO, and unformatted IO. Formatted IO use overloaded operators. Example:

```
istream& istream::operator>>(charT&);
ostream& ostream::operator<<(charT);
```

Examples of unformatted IO are:

```
istream& istream::get(charT *, streamsize, charT);
int_type ostream::put(charT);
```

This does not allow us to overload on unsigned charT. We can make the formatted operators global, and then overload ("specialize") on char, and wchar_t, but that doesn't solve the unformatted problem.

There is also a problem of inserting or extracting wide-characters from a skinny stream or skinny characters from a wide-stream:

```
char    c;
wchar_t wc;

cout << wc;
wcout << c;
```

Possible Resolution:

I propose two different solutions. Both of them solve the problem.

Solution #1

I propose to change the current member functions that "use" charT's as the argument type to char and wchar_t. For example:

replace:

```
istream& istream::operator>>(charT&);
```

with:

```
istream& istream::operator>>(char&);
```

```
istream& istream::operator>>(signed char&);
istream& istream::operator>>(unsigned char&);
istream& istream::operator>>(wchar_t&);
```

Users can easily add a new global insertion/extraction operator for their new character type. They can also derive from istream or ostream and add their own unformatted IO functions for their new character type.

This would also solve the problem of inserting skinny characters into a wide stream or wide characters into a skinny stream.

For the unformatted IO functions, we replace:

```
istream& istream::get(charT *, streamsize, charT);
```

with:

```
istream& istream::get(char *, streamsize, char);
istream& istream::get(unsigned char *, streamsize, unsigned char);
istream& istream::get(signed char *, streamsize, signed char);
istream& istream::get(wchar_t *, streamsize, wchar_t);
```

We would also need to replace the other members that make sense reading or writing unsigned char, or signed char values.

This would still allow users to have streams of unsigned char, or any other type.

Solution #2

Leave the classes as they are, but add several new member functions. For example:

Leave this member function:

```
istream& istream::operator>>(charT&);
```

and add these member functions:

```
istream& istream::operator>>(unsigned char&);
istream& istream::operator>>(signed char&);
```

For the unformatted IO functions we leave this member function:

```
istream& istream::get(charT *, streamsize, charT);
```

and add these member functions:

```
istream& istream::get(unsigned char *, streamsize, unsigned char);
istream& istream::get(signed char *, streamsize, signed char);
```

This would still allow users to create their own character type class and also provide backward compatibility. However, this would mean that users could not have istream<unsigned char>, which I think is a resonable restriction.

This would not solve the skinny-character-on-wide-stream problem, though. To solve this problem, we can overload the formatted functions:

We can define global inserters/extractors for these special cases:

```
namespace std {
 ostream&  operator<<(ostream&, wchar_t);
 wostream& operator<<(wostream&, char);

 istream&  operator>>(istream&, wchar_t&);
```

```
 wistream& operator>>(wistream&, char&);
}
```

This would still not allow us to insert a skinny-character-on-wide-stream using the unformatted IO routines. I'm not sure if that is a real problem or not. If you need to use the unformatted operations, you could easily use either read or write.

The following functions would need to be changed for either solution:

```
istream& operator>>(char_type *);
istream& operator>>(char_type&);
istream& get(char_type *, streamsize, char_type);
istream& getline(char_type *, streamsize, char_type);

ostream& operator<<(char_type *);
ostream& operator<<(char_type);
```

Requestor: John Hinke (hinke@roguewave.com)

Issue Number: 27-902
Title: default locale
Section: 27
Status: **active**
Description:

In order to coordinate the C-language locale model, I believe that the default locale value should not be 'locale::classic ()', what we call "C" locale, but be 'locale::global()', the current global locale.

Possible Resolution:

In 27.4.4.1 **basic_ios constructors [lib.basic.ios.cons]** change the following line of "Table 8-ios_base() effects" from:

getloc()   locale::classic()

to:

getloc()   locale::global()

The Tokyo meeting recommended acceptance of this issue.

Requestor: Nathan Myers (ncm@cantrip.org)
Norihiro Kumagai (kuma@slab.tnr.sharp.co.jp)

Issue Number: 27-903
Title: [io]{pfs|sfx} and exceptions
Section: 27.6.1.1.2 **basic_istream prefix and suffix [lib.istream.prefix]**
27.6.2.3 **basic_ostream prefix and suffix functions [lib.ostream.prefix]**
Status: **active**
Description:

The members ipfx()/opfx and isfx()/osfx() of the streams are not compatible with exceptions. We need to eliminate them in favor of member classes whose constructor/destructor perfom the same actions, in the manner of custodian classes.

Possible Resolution:

In order for istream/ostream to be safe with exceptions, the *pfx and *sfx functions need to be called in pairs. I propose introducing a new class in basic_istream and basic_ostream. This class will be responsible for "doing" *pfx type operations in the constructor and *sfx type operations in the destructor. This will guarantee that *pfx and *sfx will be called in pairs even if an exception is thrown.

Add the following class to basic_istream:

```
class sentry {
        bool ok_; // exposition only
    public:
        explicit sentry(basic_istream<charT,traits>& is,bool noskipws = false);
        ~sentry();

        operator bool();
};
```

Add the following class to basic_ostream:

```
class sentry {
        bool ok_; // exposition only
    public:
        explicit sentry(basic_ostream<charT,traits>& os);
        ~sentry();

        operator bool();
};
```

Typical usage will be something like:

```
template<class charT, class traits>
basic_istream<charT, traits>&
basic_istream<charT, traits>::
operator>>(short& s)
{
    if(sentry cerberus(*this,false)) {
        // read in short
    }

    return *this;
}
```

Class basic_istream::sentry

The class sentry defines a class that is responsible for doing ipfx and isfx type operations. This class makes prefix and suffix operations exception safe.

explicit sentry(basic_istream<charT,traits>& is, bool *noskipws* = false);

Effects: *Same as ipfx(), except that the return value is stored in* ok_.

~sentry();

Effects: *Same as isfx()*

operator bool();

Effects: Returns ok_.

Class basic_ostream::sentry

The class sentry defines a class that is responsible for doing opfx and osfx type operations. This class makes prefix and suffix operations exception safe.

explicit sentry(basic_ostream<charT,traits>& os);

Effects: *Same as opfx(), except that the return value is stored in* ok_.

~sentry();

Effects: *Same as osfx()*

operator bool();

Effects: Returns ok_.

Deprecate ipfx/opfx/isfx/osfx in favor of this technique.

The Tokyo meeting recommended approval of this with a note indicating Bill's objection, who says that we need to be cautious about infinite loops in osfx. Box 35 in 27.6.1.1 **Template class basic_istream [lib.istream]** and Box 44 in **27.6.2.1 Template class basic_ostream** need to be corrected, the constructor of the class sentry does not take the right first parameters.

Requestor: Nathan Myers (ncm@cantrip.org),
John Hinke (hinke@roguewave.com),
Jerry Schwarz (jss@declarative.com)

Issue Number: 27-904
Title: iosfwd declarations: incomplete
Section: 27.2 **Forward declarations [lib.iostream.forward]**
Status: **active**
Description:

The list of forward declarations is incomplete. Should it contain all of the forward declarations available? Forward declarations for template classes basic_ios, basic_istream, and basic_ostream should have two class parameters, not one. It is equally dicey to define ios, istream, etc. by writing just one parameter for the defining classes. All should have the second parameter supplied, which suggests the need for a forward reference to template class ios_char_traits as well, or at least the two usual specializations of that class.

Possible Resolution:

Replace “**Header <iosfwd> synopsis”** by:

```
namespace std {
    template<class charT> struct ios_traits;
    template<class charT, class traits = ios_traits<charT> > class basic_ios;
    template<class charT, class traits = ios_traits<charT> > class basic_streambuf;
    template<class charT, class traits = ios_traits<charT> > class basic_istream;
    template<class charT, class traits = ios_traits<charT> > class basic_ostream;
    template<class charT, class traits = ios_traits<charT> > class basic_stringbuf;
    template<class charT, class traits = ios_traits<charT> > class basic_istringstream;
    template<class charT, class traits = ios_traits<charT> > class basic_ostringstream;
    template<class charT, class traits = ios_traits<charT> > class basic_filebuf;
    template<class charT, class traits = ios_traits<charT> > class basic_ifstream;
    template<class charT, class traits = ios_traits<charT> > class basic_ofstream;
    template<class charT, class traits=ios_traits<charT> >   class ostreambuf_iterator;
    template<class charT, class traits=ios_traits<charT> >   class istreambuf_iterator;

    typedef basic_ios<char>                     ios;
    typedef basic_streambuf<char>               streambuf;
    typedef basic_istream<char>                 istream;
    typedef basic_ostream<char>                 ostream;
    typedef basic_stringbuf<char>               stringbuf;
    typedef basic_istringstream<char>           istringstream;
    typedef basic_ostringstream<char>           ostringstream;
    typedef basic_filebuf<char>                 filebuf;
    typedef basic_ifstream<char>                ifstream;
    typedef basic_ofstream<char>                ofstream;
    typedef basic_ios<wchar_t>                  wios;
    typedef basic_streambuf<wchar_t>            wstreambuf;
    typedef basic_istream<wchar_t>              wistream;
    typedef basic_ostream<wchar_t>              wostream;
    typedef basic_stringbuf<wchar_t>            wstringbuf;
    typedef basic_istringstream<wchar_t>                wistringstream;
    typedef basic_ostringstream<wchar_t>                wostringstream;
    typedef basic_filebuf<wchar_t>              wfilebuf;
    typedef basic_ifstream<wchar_t>             wifstream;
    typedef basic_ofstream<wchar_t>             wofstream;
}
```

Requestor: John Hinke (hinke@roguewave.com)

Issue Number: 27-905
Title: Add iostream, fstream, stringstream, and strstream
Section: 27
Status: **active**
Description:

These classes were removed from the WP (date unknown). Users will complain about this. Library vendors will probably add them back to make their users happy. There has been some discussion of this on comp.std.c++.

Add the classes back to the WP. There is a way around this problem, but it requires users to change more of their code. If at all possible, I think it would be excellent if we could reduce the amount of code that users will have to change.

Without these classes, code such as:

```
fstream   inout("test.txt");
```

Would have to be replaced by code such as:

```
filebuf   fb("test.txt");
istream   in(&fb);
ostream   out(&fb);
```

The problem with this is that there would still be code like:

```
inout << "Something";
inout >> someVar;
```

That would have to be changed and that could be a lot of work.

Possible Resolution:

John wrote:

"Add the classes back following the original AT&T implementation."

See Bi-directional Iostreams Proposal (doc n° X3J16/96-0010).

Requestor: John Hinke (hinke@roguewave.com)

Issue Number: 27-906
Title: add a typedef to access the traits parameter for a class.
Section: 27
Status: **active**
Description:

Some classes; such as istream don't have access to the traits template parameter. Perhaps each class should provide a typedef for the traits parameter.

You need the traits parameter when you want to say stuff like:

```
cin.ignore(100, traits::newline(use_facet<ctype<cin.char_type> >(cin.getloc()) )
```

There is no way to get the traits type without saying something like: ios_traits<cin.char_type> which is almost reasonable, but it would be nicer to say something like: cin::traits_type. There are some cases where ios_traits is not the traits used to instantiate the stream.

Possible Resolution:

The Tokyo meeting recommends acceptance of the following:

Add "typedef traits traits_type;" to basic_ios and basic_streambuf.

Where traits is the template parameter

Requestor: John Hinke (hinke@roguewave.com)

Issue Number: 27-907
Title: Use of “instance of” vs. “version of” in descriptions of class ios
Section: 27.2 **Forward declarations [lib.iostream.forward]**
Status: **active**
Description:

Paragraph 2 and 3 describe the class ios and the class wios. One is described as “an instance of the template...” the other is described as “a version of the template...”.

Possible Resolution:

Change paragraph 3 to:

“The class wios is an instance of the template class basic_ios, specialized by the type wchar_t”

Requestor: John Hinke (hinke@roguewave.com)

Issue Number: 27-908
Title: unnecessary ‘;’ (semicolons) in tables
Section: 27
Status: **active**
Description:

There are unnecessary semicolons in tables in chapter 27. These probably should be removed.

Possible Resolution:

Remove unnecessary semicolon in section 27.1.2.6 **Type POS_T [lib.iostreams.pos.t]** “Table 2-Position type requirements”.

Requestor: John Hinke (hinke@roguewave.com)

Issue Number: 27-909
Title: Editorial issues (typo’s)
Section: 27
Status: **active**
Description:

Here are a list of “typo’s” and other possible editorial issues.

Editorial Issue #1
Description:
27.4.4.3 basic_ios iostate flags functions [lib.iostate.flags]
The description of ios_base::exceptions is listed under the basic_ios clause.

Possible Resolution:
This needs to be moved back to the ios_base clause.

Editorial Issue #2
Description:
27.4.2 Template struct ios_traits [lib.ios.traits]

The template declaration is incorrect C++ code.

Possible Resolution:
Change the template declaration to:
template <class charT> struct ios_traits {

by removing the <charT>.

Editorial Issue #3
Description:
27.1.2.4 **Type POS_T [lib.iostreams.pos.t]**
Description of type POS_T contains many awkward phrases. Needs rewriting for clarity.

Editorial Issue #4
Description:
27.1.2.3 **Type OFF_T [lib.iostreams.off.t]**
Footnote 1 should say ``for one of'' instead of ``for one if.'' Also, it should``whose representation has at least'' instead of ``whose representation at least.''

Requestor: John Hinke (hinke@roguewave.com)

Issue Number: 27-910
Title: remove streampos in favor of pos_type
Section: 27
Status: **active**
Description:

There are editorial boxes in Chapter 27 that say that streampos was deprecated but no resolution on what to do with functions that use it as an argument type has been offered.

Possible Resolution:

Change all references to streampos as an argument type to pos_type. Each class in Chapter 27 has a typedef for, or access to, pos_type.

Requestor: John Hinke (hinke@roguewave.com)

Issue Number: 27-911
Title: stdio synchronization
Section: 27.3.1 Narrow stream objects [**lib.narrow.stream.objects]**
Status: **active**
Description:

Doing measurements on the performance of streambufs attached to stdin on a variety of systems, I found that the performance of a simple loop:

```
while ((c = cin.get()) != EOF) ...
```

was from 5 to 20 times slower than the equivalent

```
while ((c = getc(stdin)) != EOF) ...
```

To my horror, I found that this is a result of a mandate in the WP, that stdin and cin (and also stdout and cout) must be synchronized. As a goal this seems laudable, but if the consequence in many (most) environments is either:

1. an order of magnitude slower input, or
2. breaking link compatibility with C,

maybe we should reconsider this choice, and instead allow-but-not-require that the two be synchronized.

Possible Resolution:

Nathan wrote:

"One possibility would be to reintroduce "sync_with_stdio" but give it a boolean argument. sync_with_stdio(true) would cause syncronization, while sync_with_stdio(false) would cause unsyncronization.

This would be agreeable to me. I take it this would be a static member of ios_base? How would it default? I assume that the call with false could be a no-op."

Requestor: Nathan Myers (ncm@cantrip.org)

Issue Number: 27-912
Title: removing **Notes:** from the text
Section: 27
Status: **active**
Description:

This issue is in response to Mats Meta list. It is an attempt to remove normative text from the WP. This issue removes **Notes:** from the text. Some **Notes:** clauses that need to be incorporated into the text will be handled in another issue.

Remove all **Notes:** clauses from the following:

27.4.2.1 ios_traits value functions [lib.ios.traits.values]
int_type not_eof(char_type c)

27.4.2.1 ios_traits value functions [lib.ios.traits.values]
char_type newline()

27.4.3.4 ios_base storage functions [lib.ios.base.storage]
void * & pword(int idx)

27.5.2.2.3 Get area [lib.streambuf.pub.get]
int_type snextc()

27.5.2.4.3 Get area [lib.streambuf.virt.get]
int showmanyc()

27.5.2.4.3 Get area [lib.streambuf.virt.get]
streamsize xsgetn(char_type **s*, streamsize *n*)

27.6.1.2.2 basic_istream::operator>> [lib.istream::extractors]
basic_istream<charT, traits>& operator>>(char_type *s)

27.7.1.3 Overridden virtual functions [lib.stringbuf.virtuals]
int_type pbackfail(int_type *c*)

27.7.1.3 Overridden virtual functions [lib.stringbuf.virtuals]
int_type overflow(int_type *c*)

27.8.1.4 Overridden virtual functions [lib.filebuf.virtuals]
int showmanyc()

Possible Resolution:

Requestor: John Hinke (hinke@roguewave.com)

Issue Number: 27-913
Title: Incorporating **Notes:** into the text
Section: 27
Status: **active**
Description:

The following **Notes:** clauses need to be incorporated into the WP text:

27.5.2.1 basic_streambuf constructors [lib.streambuf.cons]
basic_streambuf()

27.5.2.4.1 Locales [lib.streambuf.virt.locales]
void imbue(const locale&)

27.5.2.4.3 Get area [lib.streambuf.virt.get]
int_type underflow()

27.5.2.4.4 Putback [lib.streambuf.virt.pback]
int_type pbackfail(int *c*)

27.5.2.4.5 Put area [lib.streambuf.virt.put]
int_type overflow(int_type *c*)

27.6.1.1.1 basic_istream constructors [lib.basic.istream.cons]
virtual ~basic_istream()

27.6.1.1.2 basic_istream prefix and suffix [lib.istream.prefix]
bool ipfx(bool *noskipws*)

27.6.1.2.2 basic_istream::operator>> [lib.istream::extractors]
basic_istream<charT, traits>& operator>>(bool& *n*)

27.6.1.3 Unformatted input functions [lib.istream.unformatted]
basic_istream<charT, traits>& ignore(int *n*, int_type *delim*)

27.6.2.2 basic_ostream constructors [lib.ostream.cons]
virtual ~basic_ostream()

27.6.2.4.2 basic_ostream::operator<< [lib.ostream.inserters]
basic_ostream<charT, traits>& operator<<(char_type c)
Change this **Notes:** clause to a **Requires:** clause.

27.7.1.1 basic_stringbuf constructors [lib.stringbuf.cons]
explicit basic_stringbuf(ios_base::openmode)

27.8.1.4 Overridden virtual functions [lib.filebuf.virtuals]
int_type pbackfail(int_type *c*)

Possible Resolution:

Requestor:	John Hinke (hinke@roguewave.com)

Issue Number:	27-914
Title:	rethrowing exceptions
Section:	27.6.2.4.1 Common requirements **[lib.ostream.formatted.reqmts]**
Status:	**active**
Description:	

[NOTE: This follows directly with 27-903 --John Hinke]

The typical operator<< looks like this, given current semantics for exceptions:

```
{
  sentry cerberos(*this); if (!cerberos) return;
  iostate save = exceptions(); exceptions(0);

  try {
    if (use_facet< num_put<charT,ostreambuf_iterator<charT,traits> >(
          getloc()).put(*this,*this,fill(),getloc(),val).failed())
      setstate(failbit); // won't throw
  }
  catch (...) { exceptions(save); setstate(badbit); throw; }
  exceptions(save); setstate(rdstate());
}
```

If we change exception semantics so that ios_base::failure just gets rethrown, without setting badbit, we have instead:

```
{
  sentry cerberos(*this);
  if (!cerberos) return;
  try {
    if (use_facet< num_put<charT,ostreambuf_iterator<charT,traits> >(
          getloc()).put(*this,*this,fill(),getloc(),val).failed())
      setstate(failbit);  // might throw
  }
  catch (const ios_base::failure&) { throw; }
  catch (...)                      { setstate(badbit); throw; }
}
```

The examples don't constitute an argument for or against the change, but rather are suggestions for the example code that should appear in **[lib.ostream.formatted.reqmts]** according to what is decided.

For the record, I am in favor of the change.

Possible Resolution:

This issue is related to issue 27-504, in which another scheme is proposed.

Requestor: Nathan Myers (ncm@cantrip.org)

Issue Number: 27-915
Title: The use of specialization
Section: 27
Status: **active**
Description:

There is wording in Clause 27 such as:

"...iostream classes are the instantiations of the..."
"...class ios is an instance of the..."
"...class wios is a version of the..."

This wording needs to be consistent with the rest of the document.

Possible Resolution:

Make the following changes to be consistent:

27.1.1 Definitions [lib.iostreams.definitions]
Replace: "**-- narrow-oriented iostream classes** ...iostream classes are the instantiations of the..."
With: "**--narrow-oriented iostream classes** ...iostream classes are specializations of the..."

27.1.1 Definitions [lib.iostreams.definitions]
Replace: "**-- wide-oriented iostream classes** ...iostream classes are the instantiations of the..."
With: "**-- wide-oriented iostream classes** ...iostream classes are specializations of the..."

27.2 Forward declarations [lib.iostream.forward] paragraph 2
Replace: "The class ios is an instance of the template..."
With: "The class ios is a specialization of the template..."

27.2 Forward declarations [lib.iostream.forward] paragraph 3
Replace: "The class wios is a version of the template..."
With: "The class wios is a specialization of the template..."

27.4.2 Template struct ios_traits [lib.ios.traits] paragraph 2
Replace: "An implementation shall provide the following two instantiations of ios_traits:"
With: "An implementation shall provide the following two specializations of ios_traits:"

27.5.2 Templates class basic_streambuf<charT, traits> [lib.streambuf] paragraph 2

Replace: “The class streambuf is an instantiation of the template...”
With: “The class streambuf is a specialization of the template...”

27.5.2 Templates class basic_streambuf<charT, traits> [lib.streambuf] paragraph 3
Replace: “The class wstreambuf is an instantiation of the template...”
With: “The class wstreambuf is a specialization of the template...”

Requestor: John Hinke (hinke@roguewave.com)

Issue Number: 27-916
Title: missing descriptions of specializations
Section: 27
Status: **active**
Description:

For compatibility, each templatized class has two specializations. One for skinny characters and one for wide characters. For example:

```
template<class charT, class traits>
class basic_ios : public ios_base {
  //...
};
```

Class ios is a specialization of...
Class wios is a specialization of...

These descriptions are missing for some of the classes. This proposal adds these missing descriptions.

Possible Resolution:

Add the following descriptions to the appropriate sections:

For class basic_ios:

27.4.4 Template class basic_ios [lib.ios]
The class ios is a specialization of the template class basic_ios specialized by the type char.

The class wios is a specialization of the template class basic_ios specialized by the type wchar_t.

For class basic_istream:

27.6.1.1 Template class basic_istream [lib.istream]
The class istream is a specialization of the template class basic_istream specialized by the type char.

The class wistream is a specialization of the template class basic_istream specialized by the type wchar_t.

For class basic_ostream:

27.6.2.1 Template class basic_ostream [lib.ostream]
The class ostream is a specialization of the template class basic_ostream specialized by the type char.

The class wostream is a specialization of the template class basic_ostream specialized by the type wchar_t.

For class basic_stringbuf:

27.7.1 Template class basic_stringbuf [lib.stringbuf]

The class stringbuf is a specialization of the template class basic_stringbuf specialized by the type char.

The class wstringbuf is a specialization of the template class basic_stringbuf specialized by the type wchar_t.

For class basic_istringstream:

27.7.2 Template class basic_istringstream [lib.istringstream]

The class istringstream is a specialization of the template class basic_istringstream specialized by the type char.

The class wistringstream is a specialization of the template class basic_istringstream specialized by the type wchar_t.

For class basic_ostringstream:

27.7.2.3 Template class basic_ostringstream [lib.ostringstream]

The class ostringstream is a specialization of the template class basic_ostringstream specialized by the type char.

The class wostringstream is a specialization of the template class basic_ostringstream specialized by the type wchar_t.

For class basic_filebuf:

27.8.1.1 Template class basic_filebuf [lib.filebuf]

The class filebuf is a specialization of the template class basic_filebuf specialized by the type char.

The class wfilebuf is a specialization of the template class basic_filebuf specialized by the type wchar_t.

For class basic_ifstream:

27.8.1.5 Template class basic_ifstream [lib.ifstream]

The class ifstream is a specialization of the template class basic_ifstream specialized by the type char.

The class wifstream is a specialization of the template class basic_ifstream specialized by the type wchar_t.

For class basic_ofstream:

27.8.1.8 Template class basic_ofstream [lib.ofstream]

The class ofstream is a specialization of the template class basic_ofstream specialized by the type char.

The class wofstream is a specialization of the template class basic_ofstream specialized by the type wchar_t.

Requestor: John Hinke (hinke@roguewave.com)

Issue Number: 27-917
Title: Editorial changes
Section: 27.1.2 Type requirements **[lib.iostreams.type.reqmts]**
Status: **active**
Description:

27.1.2 **[lib.iostreams.type.reqmts]**: Last sentence: "... expects to the character container class." should read "... expects of the character container class."

Possible Resolution:

Requestor: Public Comment

Issue Number: 27-918
Title: Validity of OFF_T to POS_T conversion
Section: 27.1.2.3 Type OFF_T **[lib.iostreams.off.t]**
Status: **active**
Description:

27.1.2.3 **[lib.iostreams.off.t]**: Paragraph 4: "Type OFF_T is convertible to type POS_T. But no validity of the resulting POS_T value is ensured, whether or not the OFF_T value is valid." Of what use is the conversion, then?

Possible Resolution:

Requestor: Public Comment

Issue Number: 27-919
Title: Question on Table 2 assertions
Section: 27.1.2.4 Table2 Position type requirements **[lib.iostreams.pos.t]**
Status: **active**
Description:

27.1.2.4 **[lib.iostreams.pos.t]**: table 2: first row has assertion "p == P(i)" but p does not appear in the expression for that row; also, that row has the note "a destructor is assumed" -- what does this mean?

Possible Resolution:

The first row of table 2 should be deleted. The second row already specifies the construction and assignment from an integer value.

Requestor: Public Comment

Issue Number: 27-920
Title: destination of clog and wclog
Section: 27.3.1 Narrow stream objects **[lib.narrow.stream.objects]**,
27.3.2 Wide stream objects **[lib.wide.stream.objects]**
Status: **active**
Description:

There is currently an editorial box concerning the destination of clog and wclog. I would like to propose the following resolution:

Possible Resolution:

Change **27.3.1 Narrow stream objects [lib.narrow.stream.objects]** paragraph 6 to:
The object clog controls output to an implementation defined stream buffer.

Change **27.3.2 Wide stream objects [lib.wide.stream.objects]** paragraph 6 to:
The object wclog controls output to an implementation defined stream buffer.

Requestor: John Hinke (hinke@roguewave.com)

Issue Number: 27-921
Title: default locale argument to constructor
Section: 27
Status: **active**
Description:

Default locale arguments for stream constructors.

istream and ostream constructors (and all derivations) should have a default locale argument, in the manner of

```
obogusstream(const char *name,const locale& l = locale());
```

Possible Resolution:

Add a new argument to the standard stream constructors:

```
const locale& l = locale::global()
```

Add this new argument to the following classes' constructors:

basic_istream,
basic_ostream,
basic_istringstream,
basic_ostringstream,
basic_ifstream,
basic_ofstream
istrstrem
ostrstream

Requestor: Nathan Myers (ncm@cantrip.org)
Norihiro Kumagai (kuma@slab.tnr.sharp.co.jp)

Annex D issues

Issue Number: 27-1001
Title: description of function setbuf is not sufficient
Section: D.6.1.3 **strstreambuf overridden virtual functions [depr.strstreambuf.virtuals]**
Status: **active**
Description:

Description of the overridden setbuf(char* s,streamsize n) function in class strstreambuf is not sufficient.

Possible Resolution:

Change the current description of function setbuf(char* s, streamsize n) in D.6.1.3 **strstreambuf overridden virtual functions [depr.strstreambuf.virtuals]** to:

streambuf* setbuf(char* s, streamsize n);

and not:

streambuf<char>* setbuf(char* s, streamsize n);

Effects: If *s* is not a null pointer, and n > pptr() - pbase(), replace the current buffer (copy its contents and deallocate it) by the buffer of size *n* pointed at by *s*.
In the case where *s* is a null pointer, and n > pptr() - pbase() resize the current buffer to size *n*.
If the function fails, it returns a null pointer.
Returns: (streambuf*)(this)

I am not qualified enough to decide if the return type should be changed to strstreambuf* as proposed by Steve Clamage in issue 27-809. I tried it with several compilers, and the results were just error messages. Basically, the compilers were complaining about the fact that the base class virtual function and the overridden virtual function should have the same return type.

Requestor: philippe Le Mouël (philippe@roguewave.com)

Issue Number: 27-1002
Title: strstreambuf Editorial issues (typos)
Section: D.6.1 **Class strstreambuf [depr.strstreambuf]**
Status: **active**
Description:

Class strstreambuf contains several typos and is also missing some typedefs.

Possible Resolution:

The following typedefs need to be added to class strstreambuf (D.6.1 **Class strstreambuf [depr.strstreambuf]**) :

- typedef ios_traits<char>::int_type int_type;

This typedef is used in the strstreambuf overridden virtual functions *overflow , pbackfail* and *underflow*.

- typedef ios_traits<char>::pos_type pos_type;

This typedef is used in the strstreambuf overridden virtual functions *seekoff* and *seekpos.*

- typedef ios_traits<char>::off_type off_type;

This typedef is used in the strstreambuf overridden virtual function *seekoff.*

In D.6.1 **Class strstreambuf [depr.strstreambuf]** the notation of the strstreambuf base class is wrong it should say:

class strstreambuf : public basic_streambuf<char>

and not:

class strstreambuf : public streambuf<char> // does not exist

In D.6.1 **Class strstreambuf [depr.strstreambuf]** the declaration of function freeze is missing the argument name. It should say:

void freeze(bool freezefl =1);

and not:

void freeze(bool = 1);

Requestor: Philippe Le Mouël (philippe@roguewave.com)

Issue Number: 27-1003
Title: istrstream Editorial issues (typos)
Section: D.6.2 **Template class istrstream [depr.istrstream]**
Status: **active**
Description:

Class istrstream contents several typos.

Possible Resolution:

In D.6.2 **Template Class istrstream [depr.istrstream]** the previous title should be changed to "D.6.2 Class istrstream", because the class is not a template class.

In D.6.2 **Template Class istrstream [depr.istrstream]** the notation of the istrstream base class is wrong. It should say:

class istrstream : public basic_istream<char>

and not:

class istrstream : public istream<char> // does not exist

Requestor: Philippe Le Mouël (philippe@roguewave.com)

Issue Number: 27-1004

Title: ostrstream Editorial issues (typos)
Section: D.6.3 **Template class ostrstream [depr.ostrstream]**
Status: **active**
Description:

Class ostrstream contents several typos.

Possible Resolution:

In D.6.3 **Template Class ostrstream [depr.ostrstream]** the previous title should be changed to "D.6.3 Class ostrstream", because the class is not a template class.

In D.6.3 **Template Class ostrstream [depr.ostrstream]** the notation of the ostrstream base class is wrong. It should say:

class ostrstream : public basic_ostream<char>

and not:

class ostrstream : public ostream<char> // does not exist

In D.6.3 **Template Class ostrstream [depr.ostrstream]** and D.6.3.2 **Member functions[depr.ostrstream.members]** the declaration of function *void freeze(int freezefl = 1)* is not consistent with the declaration in D.6.1 **Class strstreambuf [depr.strstreambuf],** which is *void freeze(bool freezefl =1).* The argument should be *bool* or *int*, but not *bool* in one and *int* in the other.

Requestor: Philippe Le Mouël (philippe@roguewave.com)