X3J16/95-0194
WG21/N0794

IOStreams Issues List
Library Clause 27

By: John Hinke, Quantitative Data Systems
jhinke@qds.com

History

Pre-Tokyo	X3J16/95-0194 WG21/N0794
Pre-Monterey	X3J16/95-0089 WG21/N0689
Pre-Austin	X3J16/95-0034 WG21/N0634

Summary of Issues

27.4.2 `ios_traits`

Active 27-001 Making `newline` locale aware
Active 27-002 `is_whitespace` is inconsistent
Active 27-003 Mention of base struct `string_char_traits`
Active 27-004 example of changing the behavior of `is_whitespace` is incorrect.
Active 27-005 `not_eof` specification
Active 27-006 `streamsize` should be ***SZ_T*** not ***INT_T***
Active 27-007 `ios_traits` typedefs are '`char`' oriented
Active 27-008 `ios_traits::length` is missing **Returns:** clause
Active 27-009 `ios_traits::get_state` should be specified
Active 27-010 `ios_traits::get_pos` should be specified
Active 27-011 Return type for `ios_traits::copy` is incorrect

27.4.3 `ios_base`

Active 27-101 `ios_base` manipulators

27.4.4 `basic_ios`

Active 27-201 missing throw specifications for `clear` and `setstate`
Active 27-202 `tie` not required to be associated with an input sequence
Active 27-203 `operator bool()` needs to be fixed

27.5.2 `basic_streambuf`

Active 27-301 imbuing on streambufs. When, how often, etc...
Active 27-302 `sungetc` has an incorrect return type

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

27.6.1 `basic_istream`

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

27.6.2 `basic_ostream`

Active 27-501 `op<<(char)` needs to be consistant with the other formatted inserters
Open 27-502 `op<<(void *)` should it be `const volatile void *`
Active 27-503 `ostream` functions need to check for `NULL` streambuf

27.6.1-27.6.2 `basic_istream`, `basic_ostream`

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 27-605 Should get/put use iterators?

27.7 `basic_stringbuf`, `basic_istringstream`, `basic_ostringstream`

Active 27-701 `str()` needs to clarify return value on else clause
Active 27-702 string stream classes need to have `string_char_traits` and `allocator` parameters

27.8.2 `basic_filebuf`, `basic_ifstream`, `basic_ofstream`

Active 27-801 `filebuf::underflow` example is incorrect
Active 27-802 `filebuf::is_open` is a bit confusing

Miscellaneous

Active 27-901 input/output of `unsigned char`, and `signed char`
Active 27-902 default locale arguments for stream constructors
Active 27-903 `ipfx`/`opfx`/`isfx`/`osfx` not compatible with exceptions.
Active 27-904 `iosfwd` declarations incomplete
Active 27-905 `iostream` type classes are missing.
Active 27-906 add a typedef to access the traits parameter in each stream class
Active 27-907 Use of "instance of" vs. "version of" in descriptions of class `ios`

Active 27-908 unnecessary ‘;’ (semicolons) in tables
Active 27-909 Editorial issues (typo’s)
Active 27-910 remove `streampos` in favor of `pos_type`
Active 27-911 stdio synchronization
Active 27-912 removing **Notes:** from the text
Active 27-913 Incorporating **Notes:** into the text
Active 27-914 rethrowing exceptions

[The editorial boxes were not added as issues to this list. However, some of them are very important and need to be discussed at the Tokyo meeting. Some of the important issues are: not_eof specification, newline specification and the editorial proposal for moving some functionality from basic_ios to ios_base. --John Hinke]

[The Public Comments that are not included here as issues will be available at the Tokyo meeting. --John Hinke]

`ios_traits` issues

Issue Number: 27-001
Title: changing `traits::newline` to be locale aware
Section: 27.4.2.2 **`ios_traits` value functions**
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. This would require that the functions that have `newline()` as a default value will need to 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(getloc().template
        use<ctype<char_type> >()));
   }
```

The functions that would 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 (myersn@roguewave.com),
John Hinke(jhinke@qds.com)

Issue Number: 27-002

Title: `traits::is_whitespace()` is inconsistent
Section: 27.4.2.3 **`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(const ctype<char_type>&, char_type);
```

27.4.2.3 `ios_traits` test functions [lib.ios.traits.tests]

```
bool is_whitespace(char_type, const ctype<char_type>&);
```

27.6.1.1.2 `basic_istream::ipfx` [lib.istream.prefix]

Notes: ...uses the function

```
bool traits::is_whitespace(charT, const locale *)
```

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 which signature is correct. The purpose of this function is to check for whitespace characters. It will most commonly be used inside a tight loop where the lookup of the `ctype` facet could be very expensive. I propose the following option:

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

Returns: `true` if *`c`* represents one of the white space characters. The default definition is as if it returns *`ct`*`.is(`*`ct`*`.space, `*`c`*`)`.

Side note: 27.4.2.3 `ios_traits::is_whitespace`: The returns paragraph calls a method of `ctype` that does not exist.

Requestor: John Hinke (jhinke@qds.com)

Issue Number: 27-003
Title: mention of base struct `string_char_traits`
Section: 27.4.2.3 **`ios_traits` test functions**
paragraph 1 **[lib.ios.traits.tests]**
Status: **active**
Description:

27.1.2.1 **Type *`CHAR_T`*** paragraph 2:
"The base class (or struct), `string_char_traits` provides the definitions common between the string class templates and the iostream class templates."

`ios_traits` is not derived from `string_char_traits`.

Possible Resolution:

Remove the sentence from 27.1.2.1.

Requestor: John Hinke (jhinke@qds.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:

Requestor: John Hinke (jhinke@qds.com)

Issue Number: 27-005
Title: `not_eof` specification
Section: 27.4.2.2 `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: 27-006
Title: `streamsize` should be ***SZ_T*** not ***INT_T***
Section: 27
Status: **active**
Description:

The current description for `streamsize` is:

```
typedef INT_T streamsize;
```

It should be:

```
typedef SZ_T streamsize;
```

Possible Resolution:

Requestor: John Hinke (jhinke@qds.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>`. It would lead to the restriction upon implementations that all of the `charT` have to be converted in '`int`' range. The restriction is too heave to 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(char_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(const ctype<char_type>
                                            ctype&, char_type);

      // 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* 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(char_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(const ctype<char_type>
                                           ctype&, char_type);

      // 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* 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 my mistakes made in my document for Austin (X1J16/95-0064) caused to introduce such the inapproplate 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 the 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 enough to 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`.

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

Issue Number: 27-008
Title: `ios_traits::length` is missing **Returns:** clause
Section: 27.4.2.2
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:

Change **Effects:** to **Returns:** and remove "Determines".

Requestor: Public Comment

Issue Number: 27-009
Title: `ios_traits::get_state` should be specified
Section: 27.4.2.4
Status: **active**
Description:

ios_traits::get_state should be specified to do more than return zero.˝Semantics are inadequate. A pos_type conceptually has three components: an off_type (streamsize), an fpos_t, and˝a state_type (mbstate_t, which may be part of fpos_t). It must be possible to compose a pos_type from these˝elements, in various combinations, and to decompose them into their three parts.

Possible Resolution:

Requestor: Public Comment

Issue Number: 27-010
Title: `ios_traits::get_pos` should be specified
Section: 27.4.2.4
Status: **active**
Description:

ios_traits::get_pos should be specified to do more than return˝pos_type(pos). Semantics are inadequate. See comments on get_state. above.

Possible Resolution:

Requestor: Public Comment

Issue Number: 27-011
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:

Requestor: John Hinke (jhinke@qds.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:

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.

Requestor: John Hinke (jhinke@qds.com)

`basic_ios` issues

Issue Number: 27-201
Title: missing throw specifications for `clear` and `setstate`
Section: 27.4.4 [lib.ios]
Status: **active**
Description:

The synopsis of `clear` and `setstate` are missing `throw(failure)`. They have the throw specification in the descriptions of the functions.

Possible Resolution:

Requestor: John Hinke (jhinke@qds.com)

Issue Number: 27-202
Title: `tie` not required to be associated with an input sequence
Section: 27.4.4.2
Status: **active**
Description:

basic_ios::tie is not necessarily synchronized with an *input* sequence. Can also be used with an output sequence.

Possible Resolution:

Change “an input” to “the”.

Requestor: Public Comment

Issue Number: 27-203
Title: `operator bool()` needs to be fixed
Section: 27.4.4
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 just 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]*

Requestor: Nathan Myers (myersn@roguewave.com)

`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 (myersn@roguewave.com)

Issue Number: 27-302
Title: `int streambuf::sungetc()`
Section: 27.5.2.2.4 **Putback [lib.streambuf.pub.pback]**
Status: **active**
Description:

The function `int basic_streambuf::sungetc()` has a return type that should be `int_type`.

Possible Resolution:

Change **27.5.2: Template `class` `basic_streambuf<charT, traits>` [lib.streambuf]**

```
int_type sungetc();
```

Change **27.5.2.2.4: `basic_streambuf::sungetc` [lib.streambuf.pub.pback]**

```
int_type sungetc();
```

Requestor: John Hinke (jhinke@qds.com)

Issue Number: 27-303
Title: `not_eof` needs to be used where appropriate
Section: 27
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())`..."

Possible Resolution:
Requestor: Per Bothner (bothner@cygnus.com)

Issue Number: 27-304
Title: `uflow` needs editing
Section: 27
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())`"

Possible Resolution:
Requestor: Per Bothner (bothner@cygnus.com)

Issue Number: 27-305
Title: `basic_streambuf::showmanyc` Incorrect return clause
Section: 27.5.2.4.3
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:
Restore original wording from the editorial box. Leave the footnote. Remove˝the Note about using `traits::eof()`.

Requestor: Public Comment

Issue Number: 27-306
Title: `basic_streambuf::uflow` has incorrect default behavior
Section: 27.5.2.4.3
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:

Requestor: Public Comment

Issue Number: 27-307
Title: `basic_streambuf::uflow` has nonsense returns clause
Section: 27.5.2.4.3
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.”

Requestor: Public Comment

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

Nathan Myers (myersn@roguewave.com) 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:

In **[lib.streambuf.get.area],** replace the description of setg as follows:

Precondition: `(gnext==0)==(gend==0) && (gnext==0)==(gbeg==0) && gbeg<=gnext && gnext<=gend.`
Postconditions: `gptr()==gnext && eback()==gbeg && egptr()==gend.`

and in **[lib.streambuf.put.area]**

Preconditions: `(pbeg==0)==(pend==0) && pbeg<=pend.`
Postconditions: `pptr()==pbeg && pbase()==pbeg && epptr()==pend.`

I believe this reflects the behavior of existing implementations.

Requestor: Nathan Myers (myersn@roguewave.com)

`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:

This function should be deprecated in favor of 27-908

Requestor: John Hinke (jhinke@qds.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:

This function should be fixed and deprecated in favor of 27-907

Requestor: John Hinke (jhinke@qds.com)

Issue Number: 27-403
Title: Clarification of exceptions thrown
Section: 27.6.1.1 **Template `class basic_istream` [lib.istream]**
Status: **active**

Description:

27.6.1.1 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:

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.

Possible Resolution:

For the functions that use `rdbuf()`, they need to check whether it is a valid pointer or not and do something appropriate. Another option would be to guarantee that the˝streambuf is never `NULL`.

Here's a list of the individual changes: (all are for `basic_istream`)

- `int_type peek();`

  "**Returns:** If `good() == false`, returns `traits::eof()`. If `rdbuf()` is a null pointer, returns `traits::eof()`, otherwise returns `rdbuf()->sgetc()`."

- `pos_type tellg();`

"**Returns:** if `fail() == true`, returns `streampos(-1)` to indicate failure. Otherwise, if `rdbuf()` is a null pointer, returns `streampos(-1)`, otherwise, returns `rdbuf()->pubseekoff(0, cur, in)`."

- `basic_istream& seekg(pos_type&);`

  "**Effects:** If `fail() != true` and if `rdbuf()` is not a null pointer, exceutes `rdbuf()->pubseekpos(`*`pos`*`)`, otherwise calls `setstate(failbit)` (which may throw `ios_base::failure`)."

- `basic_istream& seekg(off_type, ios_base::seekdir);`

  "**Effects:** If `fail() != true` and `rdbuf()` is not a null pointer, exceutes `rdbuf()->pubseekoff(`*`off, dir`*`)`, otherwise calls `setstate(failbit)` (which may throw `ios_base::failure`)."

Requestor: John Hinke (jhinke@qds.com)

`basic_ostream` issues

Issue Number: 27-501
Title: `ostream<<(char)` : formatting, padding, width
Section: 27.6.2.4.2 `basic_ostream::operator<<`
Status: active
Description:

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

Possible Resolution:

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.

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

Issue Number: 27-502
Title: `ostream::operator<<(void *)`

Section: 27.2.4.1
Status: **Open**
Description:

`ostream& 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.

Requestor: Fergus Henderson (fjh@munta.cs.mu.oz.au)

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.

Possible Resolution:

For the functions that use `rdbuf()`, they need to check whether it is a valid pointer or not and do something appropriate.

Here's a list of the individual changes: (all are for `basic_ostream`)

- `pos_type tellp();`

  "**Returns:** If `fail() == true`, returns `streampos(-1)` to indicate failure. If `rdbuf()` is a null pointer, returns `streampos(-1)`, otherwise returns `rdbuf()->pubseekoff(0, cur, out)`."

- `basic_ostream& seekp(pos_type&);`

"**Effects:** If `fail() != true` and `rdbuf()` is not a null pointer, executes `rdbuf()->pubseekpos(pos)`, otherwise calls `setstate(failbit)` (which may throw `ios_base::failure`)."

- `basic_ostream& seekp(off_type&, ios_base::seekdir);`

  "**Effects:** If `fail() != true` and `rdbuf()` is not a null pointer, executes `rdbuf()->pubseekoff(off, dir)`, otherwise calls `setstate(failbit)` (which may throw `ios_base::failure`)."

Requestor: John Hinke (jhinke@qds.com)

`basic_istream/basic_ostream` issues

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

The `ios_base` manipulators 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:

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.

Requestor: John Hinke (jhinke@qds.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:

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

Requestor: John Hinke (jhinke@qds.com)

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

```
istream& istream::read(char_type *,streamsize);
ostream& ostream::write(const char_type *,streamsize);
```

These functions are typically used for binary data.

Possible Resolution:

These function 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` which parallels the behavior of `get`.

Requestor: John Hinke (jhinke@qds.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, 27.6.2
[lib.input.streams], [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:

Should we enfore this policy? Does it ever make sense to open an `istream` for writing or an `ostream` for reading?

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

Issue Number: 27-605
Title: get/put type functuions should be able to use iterators.
Section: 27
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 took 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:
Requestor: Nathan Myers (myersn@roguewave.com)

`basic_stringbuf` 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 75 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:
Requestor: Angelika Langer (Angelika.Langer@mch.sni.de)
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`
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:

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`.

Requestor: John Hinke (jhinke@qds.com)

`basic_filebuf` issues

Issue Number: 27-801
Title: `filebuf::underflow` example
Section: 27
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:

Requestor: John Hinke (jhinke@qds.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:

Requestor: John Hinke (jhinke@qds.com),
Bob Kline (bkline@cortex.nlm.nih.gov**)**

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 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 (jhinke@qds.com)

Issue Number: 27-902
Title: default locale arguments
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::classic());
```

or perhaps:

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

Norihiro Kumagai <kuma@slab.tnr.sharp.co.jp> replies:

> In order to cordinate 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.

Most likely, it should probably be `locale::global()`.

The next issue is when can the locale change? There is nothing that says a user cannot change the current locale. In fact, an interface exists in both `ios_base` and `basic_streambuf` for changing the locale at any time. If we were to use `locale::transparent`, the locale could change even if the user didn't want it to. This isn't to say that the user couldn't imbue `locale::transparent`.

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
```

Question: Should we say anything about `str` streams?

Requestor: Nathan Myers (myersn@roguewave.com)
Norihiro Kumagai (kuma@slab.tnr.sharp.co.jp)

Issue Number: 27-903
Title: [io]{pfs|sfx} and exceptions
Section: 27.2.2.1, 27.2.4.1
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 to introduce 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(bool noskipws = false);
      ~sentry();

      operator bool();
};
```

Add the following class to `basic_ostream`:

```
class sentry {
      bool ok_; // exposition only
```

```
   public:
      explicit sentry();
      ~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(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(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();`

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.

Requestor: Nathan Myers (myersn@roguewave.com),
John Hinke (jhinke@qds.com),
Jerry Schwarz (jss@declarative.com)

Issue Number: 27-904
Title: iosfwd declarations: incomplete
Section: 27.2 Forward declarations
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:

Requestor: John Hinke (jhinke@qds.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 this 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:

Option 1:

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

Requestor: John Hinke (jhinke@qds.com)

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

Some classes like `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(cin.getloc().
   template use<ctype<cin.char_type> >()));
```

There is no way to get the traits type without saying something like: `ios_traits<cin.char_type>` which is almost resonable, 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:

Add the following to each templatized class:

```
typedef traits traits_type;
```

Where `traits` is the template parameter

Requestor: John Hinke (jhinke@qds.com)

Issue Number: 27-907
Title: Use of "instance of" vs. "version of" in descriptions of class `ios`
Section: 27.2 [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:

Requestor: John Hinke (jhinke@qds.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:

Requestor: John Hinke (jhinke@qds.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:

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`

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

Description of type POS_T contains many awkward phrases. Needs rewriting for˝clarity.

Editorial Issue #4

Description:

27.1.2.4

Footnote 207 should say ``for one of'' instead of ``for one if.'' Also, it˝should``whose representation has at least'' instead of ``whose representation at least.''

Editorial Issue #5

Description:

27.4.2.1

Remove extra **Returns:** clause from `not_eof`.

Editorial Issue #6

Description:

27.4.3

Argument types for `ios_base::precision` and `ios_base::width` should be `streamsize`.

Possible Resolution:

Requestor: John Hinke (jhinke@qds.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 that no resolution on what to do with functions that use it as an argument type.

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`.

Possible Resolution:

Requestor: John Hinke (jhinke@qds.com)

Issue Number: 27-911
Title: stdio synchronization
Section: 27
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.sgetc()) != 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:

One possibility would be to reintroduce "`sync_with_stdio`" but give it a boolean argument. `sync_with_stdio(true)` would cause syncronization `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 (myersn@roguewave.com)

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`*`)`

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

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 (jhinke@qds.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 (jhinke@qds.com)

Issue Number: 27-914
Title: rethrowing exceptions
Section: 27
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:

Requestor: Nathan Myers (myersn@roguewave.com)