Rogue Wave Standard C++ Library Iostreams and Locale User's Reference

(Part B)

Rogue Wave Software
Corvallis, Oregon USA

Rogue Wave Standard C++ Library Iostreams and Locale User's Guide and Reference

for

Rogue Wave's implementation of the Standard C++ Library.

Based on ANSI's Working Paper for Draft Proposed International Standard for Information Systems--Programming Language C++.

User's Guide and Tutorial Author: Timothy A. Budd

Internationalization and Locale User's Guide Author: Angelika Langer, with Elaine Cull

Class Reference Authors: Wendi Minne, Tom Pearson, and Randy Smithey

Product Team:

Development:	Anna Dahan, Angelika Langer, Philippe Le Mouel, Randy Smithey
Quality Engineering:	Kevin Djang, Randall Robinson
Manuals:	Elaine Cull, Wendi Minne, Julie Prince, Randy Smithey
Support:	North Krimsley

Significant contributions by: Joe Delaney

Part #	RW81-01-100096
Printing Date:	October, 1996

Rogue Wave Software, Inc., 850 SW 35th St., Corvallis, Oregon, 97333 USA

Product Information:	**(541) 754-3010**
	(800) 487-3217
Technical Support:	**(541) 754-2311**
FAX:	**(541) 757-6650**
World Wide Web:	**http://www.roguewave.com**

Iostreams and Locale Reference

Introduction

This reference guide is an alphabetical listing of the classes, facets, functions, and pre-defined streams found in the iostreams and locale portion of the Standard C++ Library.

Each entry in this reference begins with a banner that indicates either the category that an item belongs in (e.g., facet, function, pre-defined stream, etc.) or the inheritance for that class. In many cases a very brief description of the object follows the banner. Next, the reference includes a synopsis, which indicates the object's header file(s) and a class signature or function declaration.

For entries that are not classes the reference includes a text description of the item, followed by an example of its use, and references to other items related to this entry.

For classes, the reference includes a text description of the entry and a listing of the C++ code that describes the interface. Following the interface is a description of the types and methods in the class, organized into constructors, destructors, operators, member functions, and other categories. The categories are not a part of the C++ language, but do provide a way of organizing the methods. Finally, the reference includes an example of using the class, and a pointer to other entries related to the class.

Throughout the documentation, there are frequent references to "self," which should be understood to mean "`*this`".

Standards Conformance

The information presented in this reference conforms with the requirements of the ANSI X3J16/ISO WG21 Joint C++ Committee.

basic_filebuf

basic_filebuf → ***basic_streambuf***

Synopsis

```
#include <fstream>
template<class charT, class traits = char_traits<charT> >
class basic_filebuf
: public basic_streambuf<charT, traits>
```

Description

The template class ***basic_filebuf*** is derived from ***basic_streambuf***. It associates the input or output sequence with a file. Each object of type ***basic_filebuf<charT, traits>*** controls two character sequences:

- a character input sequence
- a character output sequence

The restrictions on reading and writing a sequence controlled by an object of class ***basic_filebuf<charT,traits>*** are the same as for reading and writing with the Standard C library files.

If the file is not open for reading the input sequence cannot be read. If the file is not open for writing the output sequence cannot be written. A joint file position is maintained for both the input and output sequences.

A file provides byte sequences. So the ***basic_filebuf*** class treats a file as the external source (or sink) byte sequence. In order to provide the contents of a file as wide character sequences, a wide-oriented file buffer called `wfilebuf` converts wide character sequences to multibytes character sequences (and vice versa) according to the current locale being used in the stream buffer.

Interface

```
template<class charT, class traits = char_traits<charT> >
class basic_filebuf
: public basic_streambuf<charT, traits> {

 public:

  typedef traits                         traits_type;
  typedef charT                          char_type;
  typedef typename traits::int_type      int_type;
  typedef typename traits::pos_type      pos_type;
  typedef typename traits::off_type      off_type;

  basic_filebuf();
  basic_filebuf(int fd);

  virtual ~basic_filebuf();

  bool is_open() const;
  basic_filebuf<charT, traits>* open(const char *s,
                                     ios_base::openmode,
                                     long protection = 0666);

  basic_filebuf<charT, traits>* open(int fd);

  basic_filebuf<charT, traits>* close();

 protected:

  virtual int      showmanyc();

  virtual int_type overflow(int_type c = traits::eof());

  virtual int_type pbackfail(int_type c = traits::eof());

  virtual int_type underflow();

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


  virtual pos_type seekoff(off_type off,
                          ios_base::seekdir way,
                          ios_base::openmode which =
                          ios_base::in | ios_base::out);

  virtual pos_type seekpos(pos_type sp,
                          ios_base::openmode which =
                          ios_base::in | ios_base::out);

  virtual int sync();

  virtual streamsize xsputn(const char_type* s, streamsize n);

};
```

Types

char_type
The type char_type is a synonym for the template parameter charT.

filebuf
The type filebuf is an instantiation of class basic_filebuf on type char:

```
typedef basic_filebuf<char> filebuf;
```

int_type
The type int_type is a synonym of type traits::in_type.

off_type
The type off_type is a synonym of type traits::off_type.

pos_type
The type pos_type is a synonym of type traits::pos_type.

traits_type

The type `traits_type` is a synonym for the template parameter `traits`.

wfilebuf

The type `wfilebuf` is an instantiation of class `basic_filebuf` on type `wchar_t`:

```
typedef basic_filebuf<wchar_t> wfilebuf;
```

Constructors

basic_filebuf();

Constructs an object of class `basic_filebuf<charT,traits>`, initializing the base class with `basic_streambuf<charT,traits>()`.

basic_filebuf(int fd);

Constructs an object of class `basic_filebuf<charT,traits>`, initializing the base class with `basic_streambuf<charT,traits>()`, then calls `open(fd)`. This function is not described in the C++ standard, and is provided as an extension in order to manipulate pipes, sockets or other UNIX devices, that can be accessed through file descriptors.

Destructor

virtual **~basic_filebuf**();

Calls `close()` and destroys the object.

Member Functions

basic_filebuf<charT,traits>*
close();

If `is_open() == false`, returns a null pointer. Otherwise, closes the file, and returns `*this`.

bool
is_open() const;

Returns true if the associated file is open.

basic_filebuf<charT,traits>*
open(const char* s, ios_base::openmode mode, long protection = 0666);

If `is_open() == true`, returns a null pointer. Otherwise opens the file, whose name is stored in the null-terminated byte-string `s`. The file open modes are given by their C-equivalent description (see the C function `fopen`):

in	"w"
in|binary	"rb"
out	"w"
out|app	"a"
out|binary	"wb"
out|binary|app	"ab"
out|in	"r+"
out|in|app	"a+"
out|in|binary	"r+b"
out|in|binary|app	"a+b"
trunc|out	"w"
trunc|out|binary	"wb"

```
trunc|out|in            "w+"
trunc|out|in|binary     "w+b"
```

The third argument, `protection`, is used as the file permission. It does not appear in the Standard C++ description of the function `open` and is provided as an extension. It determines the file read / write / execute permissions under UNIX. It is more limited under DOS since files are always readable and do not have special execute permission. If the `open` function fails, it returns a null pointer.

basic_filebuf<charT,traits>*
open(int fd);

Attaches the file previously opened and identified by its file descriptor `fd`, to the `basic_filebuf` object. This function is not described in the C++ standard, and is provided as an extension in order to manipulate pipes, sockets or other UNIX devices, that can be accessed through file descriptors.

int_type
overflow(int_type c = traits::eof());

If the output sequence has a put position available, and `c` is not `traits::eof()`, then write `c` into it. If there is no position available, the function output the content of the buffer to the associated file and then write `c` at the new current put position. If the operation fails, the function returns `traits::eof()`. Otherwise it returns `traits::not_eof(c)`. In wide characters file buffer, `overflow` converts the internal wide characters to their external multibytes representation by using the `locale::codecvt` facet located in the locale object imbued in the stream buffer.

int_type
pbackfail(int_type c = traits::eof());

Puts back the character designated by `c` into the input sequence. If `traits::eq_int_type(c,traits::eof())` returns true, move the input sequence one position backward. If the operation fails, the function returns `traits::eof()`. Otherwise it returns `traits::not_eof(c)`.

pos_type
seekoff(off_type off, ios_base::seekdir way,
ios_base::openmode which =   ios_base::in | ios_base::out);

If the open mode is `in | out`, alters the stream position of both the input and the output sequence. If the open mode is `in`, alters the stream position of only the input sequence, and if it is `out`, alters the stream position of only the output sequence. The new position is calculated by combining the two parameters `off` (displacement) and `way` (reference point). If the current position of the sequence is invalid before repositioning, the operation fails and the return value is `pos_type(off_type(-1))`. Otherwise the function returns the current new position. File buffers using `locale::codecvt` facet performing state dependent conversion, only support seeking to the beginning of the file, to the current position, or to a position previously obtained by a call to one of the iostreams seeking functions.

pos_type
seekpos(pos_type sp,ios_base::openmode which = ios_base::in | ios_base::out);

If the open mode is `in | out`, alters the stream position of both the input and the output sequence. If the open mode is `in`, alters the stream position of only the input sequence, and if it is `out`, alters the stream position of only the output sequence. If the current position of the sequence is invalid before repositioning, the operation fails and the return value is `pos_type(off_type(-1))`. Otherwise the function returns the current new position. File buffers using `locale::codecvt` facet performing state dependent conversion, only support seeking to the beginning of the file, to the current position, or to a position previously obtained by a call to one of the iostreams seeking functions.

basic_filebuf<charT,traits>*
setbuf(char_type*s, streamsize n);

If `s` is not a null pointer, output the content of the current buffer to the associated file, then delete the current buffer and replace it by `s`. Otherwise resize the current buffer to size `n` after outputting its content to the associated file if necessary.

int
sync();

Synchronizes the content of the external file, with its image maintained in memory by the file buffer. If the function fails, it returns -1, otherwise it returns 0.

int_type
underflow();

If the input sequence has a read position available, returns the content of this position. Otherwise fills up the buffer by reading characters from the associated file and if it succeeds, returns the content of the new current position. The function returns `traits::eof()` to indicate failure. In wide characters file buffer, `underflow` converts the external multibytes characters to their wide character representation by using the `locale::codecvt` facet located in the locale object imbued in the stream buffer.

streamsize
xsputn(const char_type* s, streamsize n);

Writes up to `n` characters to the output sequence. The characters written are obtained from successive elements of the array whose first element is designated by `s`. The function returns the number of characters written.

Examples

```
//
// stdlib/examples/manual/filebuf.cpp
//
#include<iostream>
#include<fstream>

void main ( )
{
  using namespace std;

  // create a read/write file-stream object on tiny char
  // and attach it to the file "filebuf.out"
  ofstream out("filebuf.out",ios_base::in |
ios_base::out);

  // tie the istream object to the ofstream object
  istream in(out.rdbuf());

  // output to out
  out << "Il errait comme un ame en peine";

  // seek to the beginning of the file
  in.seekg(0);

  // output in to the standard output
  cout << in.rdbuf() << endl;

  // close the file "filebuf.out"
  out.close();

  // open the existing file "filebuf.out"
  // and truncate it
  out.open("filebuf.out",ios_base::in |       ios_base::out |
ios_base::trunc);

  // set the buffer size
  out.rdbuf()->pubsetbuf(0,4096);

  // open the source code file
  ifstream ins("filebuf.cpp");

  //output it to filebuf.out
  out << ins.rdbuf();

  // seek to the beginning of the file
  out.seekp(0);

  // output the all file to the standard output
  cout << out.rdbuf();
}
```

See Also

char_traits(3C++), ***ios_base***(3C++), ***basic_ios***(3C++), ***basic_streambuf***(3C++), ***basic_ifstream***(3C++), ***basic_ofstream***(3C++), ***basic_fstream***(3C++)

Working Paper for Draft Proposed International Standard for Information Systems--Programming Language C++, Section 27.8.1.1

Standards Conformance

ANSI X3J16/ISO WG21 Joint C++ Committee

$$\begin{tikzcd}
 & & \textit{basic\_ostream} \arrow[dr] & & \\
\textit{basic\_fstream} \arrow[r] & \textit{basic\_iostream} \arrow[ur] \arrow[dr] & & \textit{basic\_ios} \arrow[r] & \textit{ios\_base} \\
 & & \textit{basic\_istream} \arrow[ur] & &
\end{tikzcd}$$

Synopsis

```
#include <fstream>
template<class charT, class traits = char_traits<charT> >
class basic_fstream
: public basic_iostream<charT, traits>
```

Description

The template class ***basic_fstream<charT,traits>*** supports reading and writing to named files or other devices associated with a file descriptor. It uses a `basic_filebuf` object to control the associated sequences. It inherits from ***basic_iostream*** and can therefore use all the formatted and unformatted input and output functions.

Interface

```
template<class charT, class traits = char_traits<charT> >
class basic_fstream
: public basic_iostream<charT, traits> {

 public:

  typedef basic_ios<charT, traits>   ios_type;

  typedef charT                       char_type;
  typedef traits                      traits_type;
  typedef typename traits::int_type  int_type;
  typedef typename traits::pos_type  pos_type;
  typedef typename traits::off_type  off_type;

  basic_fstream();

  explicit basic_fstream(const char *s, ios_base::openmode
                         mode = ios_base::in | ios_base::out,
                         long protection = 0666);

  explicit basic_fstream(int fd);

  basic_fstream(int fd, char_type *buf, int len);

  virtual ~basic_fstream();

  basic_filebuf<charT, traits> *rdbuf() const;

  bool is_open();

  void open(const char *s, ios_base::openmode mode =
            ios_base::in | ios_base::out,
            long protection = 0666);

  void close();
};
```

Types

`char_type`
The type `char_type` is a synonym for the template parameter `charT`.

`fstream`
The type `fstream` is an instantiation of class `basic_fstream` on type `char`:

```
typedef basic_fstream<char> fstream;
```

`int_type`
The type `int_type` is a synonym of type `traits::in_type`.

`ios_type`
The type `ios_type` is an instantiation of class `basic_ios` on type `charT`.

`off_type`
The type `off_type` is a synonym of type `traits::off_type`.

`pos_type`
The type `pos_type` is a synonym of type `traits::pos_type`.

`traits_type`
The type `traits_type` is a synonym for the template parameter `traits`.

`wfstream`
The type `wfstream` is an instantiation of class `basic_fstream` on type `wchar_t:`

```
typedef basic_fstream<wchar_t> wfstream;
```

Constructors

`basic_fstream``();`
Constructs an object of class ***basic_fstream<charT,traits>***, initializing the base class ***basic_iostream*** with the associated file buffer, which is initialized by calling the `basic_filebuf` constructor `basic_filebuf<charT,traits>()`. After construction, a file can be attached to the ***basic_fstream*** object by using the `open()` member function.

```
basic_fstream(const char* s,
              ios_base::openmode mode=
              ios_base::in | iosw_base::out,
              long protection= 0666);
```
Constructs an object of class ***basic_fstream<charT,traits>***, initializing the base class ***basic_iostream*** with the associated file buffer, which is initialized by calling the `basic_filebuf` constructor `basic_filebuf<charT,traits>()`. The constructor then calls the open function `open(s,mode,protection)` in order to attach the file, whose name is pointed at by `s`, to the ***basic_fstream*** object. The third argument, `protection`, is used as the file permission. It does not appear in the

Standard C++ description and is provided as an extension. It determines the file read / write / execute permissions under UNIX. It is more limited under DOS since files are always readable and do not have special execute permission.

`explicit` **`basic_fstream`**`(int fd);`

Constructs an object of class ***basic_fstream<charT,traits>***, initializing the base class ***basic_iostream*** with the associated file buffer, which is initialized by calling the `basic_filebuf` constructor `basic_filebuf<charT,traits>()`. The constructor then calls the `basic_filebuf` open function `open(fd)` in order to attach the file descriptor `fd` to the ***basic_fstream*** object. This constructor is not described in the C++ standard, and is provided as an extension in order to manipulate pipes, sockets or other UNIX devices, that can be accessed through file descriptors. If the function fails, it sets `ios_base::failbit.`

`basic_fstream``(int fd, char_type* buf,int len);`

Constructs an object of class ***basic_fstream<charT,traits>***, initializing the base class ***basic_iostream*** with the associated file buffer, which is initialized by calling the `basic_filebuf` constructor `basic_filebuf<charT,traits>()`. The constructor then calls the `basic_filebuf` open function `open(fd)` in order to attach the file descriptor `fd` to the ***basic_fstream*** object. The underlying buffer is then replaced by calling the `basic_filebuf` member function, `setbuf()`, with parameters `buf` and `len`. This constructor is not described in the C++ standard, and is provided as an extension in order to manipulate pipes, sockets, or other UNIX devices that can be accessed through file descriptors. It also maintains compatibility with the old iostreams library. If the function fails, it sets `ios_base::failbit.`

Destructor

`virtual` **`~basic_fstream`**`();`

Destroys an object of class `basic_fstream`.

Member Functions

`void`
`close``();`

Calls the associated `basic_filebuf` function `close()` and if this function fails, it calls the `basic_ios` member function `setstate(failbit)`.

`bool`
`is_open``();`

Calls the associated `basic_filebuf` function `is_open()` and return its result.

```
void
open(const char* s,ios_base::openmode =
     ios_base::out | ios_base::in,
     long protection = 0666);
```

Calls the associated `basic_filebuf` function `open(s,mode,protection)` and, if this function fails at opening the file, calls the `basic_ios` member function `setstate(failbit)`. The third argument `protection` is used as the file permissions. It does not appear in the Standard C++ description and is provided as an extension. It determines the file read / write / execute permissions under UNIX. It is more limited under DOS since files are always readable and do not have special execute permission.

```
basic_filebuf<charT,traits>*
rdbuf() const;
```

Returns a pointer to the `basic_filebuf` associated with the stream.

Examples

```
//
// stdlib/examples/manual/fstream.cpp
//
#include<iostream>
#include<bidirec>
void main ( )
{
    using namespace std;

    // create a bi-directional fstream object
    fstream inout("fstream.out");

    // output characters
    inout << "Das ist die rede von einem man" << endl;
    inout << "C'est l'histoire d'un home" << endl;
    inout << "This is the story of a man" << endl;

    char p[100];

    // seek back to the beginning of the file
    inout.seekg(0);

    // extract the first line
    inout.getline(p,100);

    // output the first line to stdout
    cout << endl << "Deutch :" << endl;
    cout << p;

    fstream::pos_type pos = inout.tellg();

    // extract the second line
    inout.getline(p,100);

    // output the second line to stdout
    cout << endl << "Francais :" << endl;
    cout << p;

    // extract the third line
    inout.getline(p,100);

    // output the third line to stdout
    cout << endl << "English :" << endl;
    cout << p;

    // move the put sequence before the second line
    inout.seekp(pos);

    // replace the second line
    inout << "This is the story of a man" << endl;

    // replace the third line
    inout << "C'est l'histoire d'un home";

    // seek to the beginning of the file
    inout.seekg(0);

    // output the all content of the fstream object to stdout
    cout << endl << endl << inout.rdbuf();
}
```

See Also

char_traits(3C++), ***ios_base***(3C++), ***basic_ios***(3C++), ***basic_filebuf***(3C++), ***basic_ifstream***(3C++), ***basic_ofstream***(3C++)

Working Paper for Draft Proposed International Standard for Information Systems--Programming Language C++, Section 27.8.1.11

Standards Conformance

ANSI X3J16/ISO WG21 Joint C++ Committee

basic_ifstream → ***basic_istream*** → ***basic_ios*** → ***ios_base***

Synopsis

```
#include <fstream>
template<class charT, class traits = char_traits<charT> >
class basic_ifstream
: public basic_istream<charT, traits>
```

Description

The template class ***basic_ifstream<charT,traits>*** supports reading from named files or other devices associated with a file descriptor. It uses a ***basic_filebuf*** object to control the associated sequences. It inherits from ***basic_istream*** and can therefore use all the formatted and unformatted input functions.

Interface

```
template<class charT, class traits = char_traits<charT> >
class basic_ifstream
: public basic_istream<charT, traits> {

 public:

  typedef basic_ios<charT, traits>    ios_type;

  typedef traits                      traits_type;
  typedef charT                       char_type;
  typedef typename traits::int_type   int_type;
  typedef typename traits::pos_type   pos_type;
  typedef typename traits::off_type   off_type;

  basic_ifstream();

  explicit basic_ifstream(const char *s,
                          ios_base::openmode mode =
                          ios_base::in,
                          long protection = 0666);

  explicit basic_ifstream(int fd);

  basic_ifstream(int fd, char_type* buf, int len);

  virtual ~basic_ifstream();

  basic_filebuf<charT, traits> *rdbuf() const;

  bool is_open();

  void open(const char *s, ios_base::openmode mode =
            ios_base::in, long protection = 0666);

  void close();

};
```

Types

`char_type`

The type `char_type` is a synonym for the template parameter `charT`.

`ifstream`

The type `ifstream` is an instantiation of class `basic_ifstream` on type `char`:

```
typedef basic_ifstream<char> ifstream;
```

`int_type`

The type `int_type` is a synonym of type `traits::in_type`.

`ios_type`

The type `ios_type` is an instantiation of class `basic_ios` on type `charT`.

`off_type`

The type `off_type` is a synonym of type `traits::off_type`.

`pos_type`

The type `pos_type` is a synonym of type `traits::pos_type`.

`traits_type`

The type `traits_type` is a synonym for the template parameter `traits`.

`wifstream`

The type `wifstream` is an instantiation of class `basic_ifstream` on type `wchar_t`:

```
typedef basic_ifstream<wchar_t> wifstream;
```

Constructors

```
basic_ifstream();
```

Constructs an object of class ***basic_ifstream<charT,traits>***, initializing the base class ***basic_istream*** with the associated file buffer, which is initialized by calling the `basic_filebuf` constructor `basic_filebuf<charT,traits>()`. After construction, a file can be attached to the ***basic_ifstream*** object by using the `open` member function.

```
basic_ifstream(const char* s,
               ios_base::openmode mode= ios_base::in,
               long protection= 0666);
```

Constructs an object of class ***basic_ifstream<charT,traits>***, initializing the base class ***basic_istream*** with the associated file buffer, which is initialized by calling the `basic_filebuf` constructor `basic_filebuf<charT,traits>()`. The constructor then calls the open function `open(s,mode,protection)` in order to attach the file whose name is pointed at by `s`, to the ***basic_ifstream*** object. The third argument `protection` provides file permissions. It does not appear in the Standard C++ description and is provided as an extension. It determines the file read / write / execute permissions under UNIX. It is more limited under

DOS since files are always readable and do not have special execute permission.

`explicit` **`basic_ifstream`**`(int fd);`

Constructs an object of class ***basic_ifstream<charT,traits>***, initializing the base class ***basic_istream*** with the associated file buffer, which is initialized by calling the `basic_filebuf` constructor `basic_filebuf<charT,traits>()`. The constructor then calls the `basic_filebuf` open function `open(fd)` in order to attach the file descriptor `fd` to the ***basic_ifstream*** object. This constructor is not described in the C++ standard, and is provided as an extension in order to manipulate pipes, sockets or other UNIX devices, that can be accessed through file descriptors. If the function fails, it sets `ios_base::failbit.`

`basic_ifstream``(int fd, char_type* buf,int len);`

Constructs an object of class ***basic_ifstream<charT,traits>***, initializing the base class ***basic_istream*** with the associated file buffer, which is initialized by calling the `basic_filebuf` constructor `basic_filebuf<charT,traits>()`. The constructor then calls the `basic_filebuf` open function `open(fd)` in order to attach the file descriptor `fd` to the ***basic_ifstream*** object. The underlying buffer is then replaced by calling the `basic_filebuf` member function `setbuf` with parameters `buf` and `len`. This constructor is not described in the C++ standard, and is provided as an extension in order to manipulate pipes, sockets or other UNIX devices, that can be accessed through file descriptors. It also maintains compatibility with the old iostreams library. If the function fails, it sets `ios_base::failbit.`

Destructor

`virtual` **`~basic_ifstream`**`();`

Destroys an object of class `basic_ifstream`.

Member Functions

```
void
close();
```

Calls the associated `basic_filebuf` function `close()` and if this function fails, it calls the `basic_ios` member function `setstate(failbit)`.

```
bool
is_open();
```

Calls the associated `basic_filebuf` function `is_open()` and return its result.

```
void
open(const char* s,ios_base::openmode =
     ios_base::in, long protection = 0666);
```

Calls the associated `basic_filebuf` function `open(s,mode,protection)`. If this function fails opening the file, it calls the `basic_ios` member function `setstate(failbit)`. The third argument `protection` provides file permissions. It does not appear in the Standard C++ description and is provided as an extension. It determines the file read / write / execute permissions under UNIX. It is more limited under DOS since files are always readable and do not have special execute permission.

```
basic_filebuf<charT,traits>*
rdbuf() const;
```

Returns a pointer to the `basic_filebuf` associated with the stream.

Examples

```
//
// stdlib/examples/manual/ifstream.cpp
//
#include<iostream>
#include<fstream>
#include<iomanip>

void main ( )
{
  using namespace std;

  long   l= 20;
  char   *ntbs="Le minot passait la piece a frotter";
  char   c;
  char   buf[50];

try {

  // create a read/write file-stream object on char
  // and attach it to an ifstream object
  ifstream in("ifstream.out",ios_base::in |
             ios_base::out | ios_base::trunc);

  // tie the ostream object to the ifstream object
  ostream out(in.rdbuf());

  // output ntbs in out
  out << ntbs << endl;

  // seek to the beginning of the file
  in.seekg(0);

  // output each word on a separate line
  while ( in.get(c) )
   {
     if ( char_traits<char>::eq(c,' ') )
      cout << endl;
     else
      cout << c;
   }
  cout << endl << endl;

  // move back to the beginning of the file
  in.seekg(0);

  // clear the state flags
  in.clear();

  // does the same thing as the previous code
  // output each word on a separate line
  while ( in >> buf )
   cout << buf << endl;

  cout << endl << endl;

  // output the base info before each integer
  out << showbase;

  ostream::pos_type pos= out.tellp();

  // output l in hex with a field with of 20
  out << hex << setw(20) << l << endl;

  // output l in oct with a field with of 20
  out << oct << setw(20) << l << endl;

  // output l in dec with a field with of 20
  out << dec << setw(20) << l << endl;

  // move back to the beginning of the file
  in.seekg(0);

  // output the all file
  cout << in.rdbuf();

  // clear the flags
  in.clear();

  // seek the input sequence to pos
  in.seekg(pos);

  int a,b,d;

  // read the previous outputted integer
  in >> a >> b >> d;

  // output 3 times 20
  cout << a << endl << b << endl << d << endl;

}
  catch( ios_base::failure& var )
   {
      cout << var.what();
   }

}
```

See Also

char_traits(3C++), ***ios_base***(3C++), ***basic_ios***(3C++), ***basic_filebuf***(3C++), ***basic_ofstream***(3C++), ***basic_fstream***(3C++)

Working Paper for Draft Proposed International Standard for Information Systems--Programming Language C++, Section 27.8.1.5

Standards Conformance

ANSI X3J16/ISO WG21 Joint C++ Committee

basic_Ios → ***ios_base***

Synopsis

```
#include <ios>
template<class charT, class traits = char_traits<charT> >
class basic_ios : public ios_base
```

Description

The class ***basic_ios*** is a base class that provides the common functionality required by all streams. It maintains state information that reflects the integrity of the stream and stream buffer. It also maintains the link between the stream classes and the stream buffers classes via the `rdbuf` member functions. Classes derived from ***basic_ios*** specialize operations for input, and output.

Interface

```
 template<class charT, class traits = char_traits<charT> >
 class basic_ios : public ios_base {

    public:

     typedef basic_ios<charT, traits>           ios_type;
     typedef basic_streambuf<charT, traits>     streambuf_type;
     typedef basic_ostream<charT, traits>       ostream_type;

     typedef typename traits::char_type         char_type;
     typedef traits                             traits_type;

     typedef typename traits::int_type          int_type;
     typedef typename traits::off_type          off_type;
     typedef typename traits::pos_type          pos_type;


     explicit basic_ios(basic_streambuf<charT, traits> *sb_arg);
     virtual ~basic_ios();

     char_type fill() const;
     char_type fill(char_type ch);

     void exceptions(iostate except);
     iostate exceptions() const;

     void clear(iostate state = goodbit);

     void setstate(iostate state);
     iostate rdstate() const;

     operator void*() const;
     bool operator!() const;

     bool good() const;
     bool eof() const;
     bool fail() const;
```

```
       bool bad() const;

       ios_type& copyfmt(const ios_type& rhs);

       ostream_type *tie() const;
       ostream_type *tie(ostream_type *tie_arg);

       streambuf_type *rdbuf() const;
       streambuf_type *rdbuf( streambuf_type *sb);

       locale imbue(const locale& loc);

       char narrow(charT, char) const;
       charT widen(char) const;

     protected:

       basic_ios();

       void init(basic_streambuf<charT, traits> *sb);
};
```

Types

char_type
The type `char_type` is a synonym of type `traits::char_type`.

ios
The type `ios` is an instantiation of `basic_ios` on `char`:

```
typedef basic_ios<char> ios;
```

int_type
The type `int_type` is a synonym of type `traits::in_type`.

ios_type
The type `ios_type` is a synonym for `basic_ios<charT, traits>` .

off_type
The type `off_type` is a synonym of type `traits::off_type`.

ostream_type
The type `ostream_type` is a synonym for `basic_ostream<charT, traits>` .

pos_type
The type `pos_type` is a synonym of type `traits::pos_type`.

streambuf_type
The type `streambuf_type` is a synonym for `basic_streambuf<charT, traits>` .

traits_type
The type `traits_type` is a synonym for the template parameter `traits`.

`wios`

The type `wios` is an instantiation of `basic_ios` on `wchar_t`:

```
typedef basic_ios<wchar_t> wios;
```

Public Constructors

`explicit` **`basic_ios`**`(basic_streambuf<charT, traits>* sb);`

Constructs an object of class `basic_ios`, assigning initial values to its member objects by calling `init(sb)`. If `sb` is a null pointer, the stream is positioned in error state, by triggering its `badbit`.

`basic_ios``();`

Constructs an object of class `basic_ios` leaving its member objects uninitialized. The object must be initialized by calling the `init` member function before using it.

Public Destructor

`virtual` **`~basic_ios`**`();`

Destroys an object of class `basic_ios`.

Public Member Functions

`bool`
`bad``() const;`

Returns true if badbit is set in `rdstate().`

`void`
`clear``(iostate state = goodbit);`

If `(state & exception()) == 0`, returns. Otherwise, the function throws an object of class `ios_base::failure`. After the call returns `state == rdstate()`.

`basic_ios&`
`copyfmt``(const basic_ios& rhs);`

Assigns to the member objects of `*this` the corresponding member objects of `rhs`, with the following exceptions:

- `rdstate()` and `rdbuf()` are left unchanged
- calls `ios_base::copyfmt`
- `exceptions()` is altered last by calling exceptions(`rhs.exceptions()`)

`bool`
`eof``() const;`

Returns true if `eofbit` is set in `rdstate().`

`iostate`
`exceptions``() const;`

Returns a mask that determines what elements set in `rdstate()` cause exceptions to be thrown.

`void`
`exceptions``(iostate except);`

Set the exception mask to `except` then calls `clear(rdstate())`.

`bool`
`fail``() const;`

Returns true if failbit or badbit is set in `rdstate().`

`char_type`
`fill``() const;`

Returns the character used to pad (fill) an output conversion to the specified field width.

`char_type`
`fill``(char_type fillch);`

Saves the field width value then replaces it by `fillch` and returns the previously saved value.

`bool`
`good``() const;`

Returns `rdstate() == 0.`

`locale`
`imbue``(const locale& loc);`

Saves the value returned by `getloc()` then assigns `loc` to a private variable and if `rdbuf() != 0` calls `rdbuf()->pubimbue(loc)` and returns the previously saved value.

`void`
`init``(basic_streambuf<charT,traits>* sb);`

Performs the following initialization:

`rdbuf()`	`sb`
`tie()`	`0`
`rdstate()`	`goodbit if sb is not null otherwise badbit`
`exceptions()`	`goodbit`
`flags()`	`skipws | dec`
`width()`	`0`
`precision()`	`6`
`fill()`	`the space character`
`getloc()`	`locale::locale()`

`char`
`narrow``(charT c, char dfault) const;`

Uses the stream's locale to convert the wide character `c` to a tiny character, and then returns it. If no conversion exists, it returns the character `dfault`.

`bool`
`operator!``() const;`

Returns `fail() ? 1 : 0;`

```
streambuf_type*
rdbuf() const;
```
Returns a pointer to the stream buffer associated with the stream.

```
streambuf_type*
rdbuf(streambuf_type* sb);
```
Associates a stream buffer with the stream. All the input and output will be directed to this stream buffer. If `sb` is a null pointer, the stream is positioned in error state, by triggering its `badbit`.

```
iostate
rdstate() const;
```
Returns the control state of the stream.

```
void
setstate(iostate state);
```
Calls `clear(rdstate() | state)`.

```
ostream_type*
tie() const;
```
Returns an output sequence that is tied to (synchronized with) the sequence controlled by the stream buffer.

```
ostream_type*
tie(ostream_type* tiestr);
```
Saves the `tie()` value then replaces it by `tiestr` and returns the value previously saved.

```
operator
void*() const;
```
Returns `fail() ? 0 : 1;`

```
charT
widen(char c) const;
```
Uses the stream's locale to convert the tiny character `c` to a wide character, then returns it.

See Also

ios_base(3C++), ***basic_istream***(3C++), ***basic_ostream***(3C++), ***basic_streambuf***(3C++), ***char_traits***(3C++)

Working Paper for Draft Proposed International Standard for Information Systems--Programming Language C++.

Standards Conformance

ANSI X3J16/ISO WG21 Joint C++ Committee

$$\begin{tikzcd} & \textit{basic\_ostream}\ldots \arrow[dr] & & \\ \textit{basic\_iostream}\ldots \arrow[ur] \arrow[dr] & & \textit{basic\_ios} \arrow[r] & \textit{ios\_base} \\ & \textit{basic\_istream}\ldots \arrow[ur] & & \end{tikzcd}$$

Synopsis

```
#include <istream>
template<class charT, class traits = char_traits<charT> >
class basic_iostream
: public basic_istream<charT, traits>,
public basic_ostream<charT, traits>
```

Description

The class ***basic_iostream*** inherits a number of functions from classes ***basic_ostream<charT, traits>*** and ***basic_istream<charT, traits>.*** They assist in formatting and interpreting sequences of characters controlled by a stream buffer. Two groups of functions share common properties, the formatted functions and the unformatted functions.

Interface

```
template<class charT, class traits>
class basic_iostream
: public basic_istream<charT, traits>,
  public basic_ostream<charT, traits>

{

 public:

  explicit basic_iostream(basic_streambuf<charT, traits> *sb);
  virtual ~basic_iostream();

 protected:

  explicit basic_iostream();

};
```

Public Constructors

`explicit` **`basic_iostream`**`(basic_streambuf<charT, traits>* sb);`

Constructs an object of class `basic_iostream,` assigning initial values to the base class by calling `basic_istream<charT, traits>(sb)` and `basic_ostream<charT, traits>(sb)`.

`explicit` **`basic_iostream`**`();`

Constructs an object of class `basic_iostream`, assigning initial values to the base class by calling `basic_istream<charT, traits>()` and `basic_ostream<charT, traits>()`. After construction the object has its badbit set.

Destructor

`virtual` **`~basic_iostream`**`();`

Destroys an object of class `basic_iostream`.

Examples

See ***basic_istream*** and ***basic_ostream*** examples.

See Also

char_traits(3C++), ***ios_base***(3C++), ***basic_ios***(3C++), ***basic_streambuf***(3C++), ***basic_istream***(3C++), ***basic_ostream***(3C++)

Working Paper for Draft Proposed International Standard for Information Systems--Programming Language C++, Section 27.6.1.4.1

Standards Conformance

ANSI X3J16/ISO WG21 Joint C++ Committee

basic_istream ⟶ **basic_ios** ⟶ **ios_base**

Synopsis

```
#include <istream>
template<class charT, class traits = char_traits<charT> >
class basic_istream
: virtual public basic_ios<charT, traits>
```

Description

The class ***basic_istream*** defines a number of member function signatures that assist in reading and interpreting input from sequences controlled by a stream buffer.

Two groups of member function signatures share common properties: the formatted input functions (or extractors) and the unformatted input functions. Both groups of input functions obtain (or extract) input characters by calling ***basic_streambuf*** member functions. They both begin by constructing an object of class ***basic_istream::sentry*** and, if this object is in `good` state after construction, the function endeavors to obtain the requested input. The `sentry` object performs exception safe initialization, such as controlling the status of the stream or locking it in multithread environment.

Some formatted input functions parse characters extracted from the input sequence, converting the result to a value of some scalar data type, and storing the converted value in an object of that scalar type. The conversion behavior is locale dependent, and directly depend on the locale object imbued in the stream.

Interface

```
template<class charT, class traits = char_traits<charT> >
class basic_istream
: virtual public basic_ios<charT, traits> {

public:

 typedef basic_istream<charT, traits>    istream_type;
 typedef basic_ios<charT, traits>        ios_type;
 typedef basic_streambuf<charT, traits>  streambuf_type;

 typedef traits                         traits_type;
 typedef charT                          char_type;
 typedef typename traits::int_type      int_type;
 typedef typename traits::pos_type      pos_type;
 typedef typename traits::off_type      off_type;

 explicit basic_istream(basic_streambuf<charT, traits> *sb);
 virtual ~basic_istream();
```

```
class sentry
 {
    public:

     inline explicit sentry(basic_istream<charT,traits>&,
                           bool noskipws = 0);
      ~sentry();
      operator bool ();
 };

istream_type& operator>>(istream_type& (*pf)(istream_type&));
istream_type& operator>>(ios_base& (*pf)(ios_base&));
istream_type& operator>>(ios_type& (*pf)(ios_type&));

istream_type& operator>>(bool& n);
istream_type& operator>>(short& n);
istream_type& operator>>(unsigned short& n);
istream_type& operator>>(int& n);
istream_type& operator>>(unsigned int& n);
istream_type& operator>>(long& n);
istream_type& operator>>(unsigned long& n);
istream_type& operator>>(float& f);
istream_type& operator>>(double& f);
istream_type& operator>>(long double& f);

istream_type& operator>>(void*& p);

istream_type& operator>>(streambuf_type& sb);
istream_type& operator>>(streambuf_type *sb);

int_type get();

istream_type& get(char_type *s, streamsize n, char_type delim);
istream_type& get(char_type *s, streamsize n);

istream_type& get(char_type& c);

istream_type& get(streambuf_type& sb,char_type delim);
istream_type& get(streambuf_type& sb);

istream_type& getline(char_type *s, streamsize n,char_type delim);
istream_type& getline(char_type *s, streamsize n);

istream_type& ignore(streamsize n = 1,
                     int_type delim = traits::eof());

istream_type& read(char_type *s, streamsize n);

streamsize readsome(char_type *s, streamsize n);

int peek();

pos_type tellg();

istream_type& seekg(pos_type&);
istream_type& seekg(off_type&, ios_base::seekdir);

istream_type& putback(char_type c);
```

```
  istream_type& unget();

  streamsize gcount() const;

  int sync();

 protected:

  explicit basic_istream( );

 };

 //global function

 template<class charT, class traits>
 basic_istream<charT, traits>&
 ws(basic_istream<charT, traits>&);


 template<class charT, class traits>
 basic_istream<charT, traits>&
 operator>> (basic_istream<charT, traits>&, charT&);


 template<class charT, class traits>
 basic_istream<charT, traits>&
 operator>> (basic_istream<charT, traits>&, charT*);


 template<class traits>
 basic_istream<char, traits>&
 operator>> (basic_istream<char, traits>&, unsigned char&);


 template<class traits>
 basic_istream<char, traits>&
 operator>> (basic_istream<char, traits>&, signed char&);


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


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

Types

char_type

The type `char_type` is a synonym for them template parameter `charT`.

int_type

The type `int_type` is a synonym of type `traits::in_type`.

ios_type

The type `ios_type` is a synonym for `basic_ios<charT, traits>` .

`istream`

The type `istream` is an instantiation of class `basic_istream` on type `char`:

```
typedef basic_istream<char> istream;
```

`istream_type`

The type `istream_type` is a synonym for `basic_istream<charT, traits>`.

`off_type`

The type `off_type` is a synonym of type `traits::off_type`.

`pos_type`

The type `pos_type` is a synonym of type `traits::pos_type`.

`streambuf_type`

The type `streambuf_type` is a synonym for `basic_streambuf<charT, traits>` .

`traits_type`

The type `traits_type` is a synonym for the template parameter `traits`.

`wistream`

The type `wistream` is an instantiation of class `basic_istream` on type `wchar_t`:

```
typedef basic_istream<wchar_t> wistream;
```

Public Constructor

explicit **basic_istream**(basic_streambuf<charT, traits>* sb);

Constructs an object of class `basic_istream`, assigning initial values to the base class by calling `basic_ios::init(sb)`.

Public Destructor

virtual **~basic_istream**();

Destroys an object of class `basic_istream`.

Sentry Class

explicit **sentry**(basic_istream<charT,traits>&, bool noskipws=0);

Prepares for formatted or unformatted input. First if the `basic_ios` member function `tie()` is not a null pointer, the function synchronizes the output sequence with any associated stream. If `noskipws` is zero and the `ios_base` member function `flags() & skipws` is nonzero, the function extracts and discards each character as long as the next available input character is a white space character. If after any preparation is completed the `basic_ios` member function `good()` is true, the sentry conversion function operator `bool()` will return `true`. Otherwise it will return `false`. In multithread environment the sentry object constructor is responsible for locking the stream and the stream buffer associated with the stream.

```
~sentry();
```

Destroys an object of class ***sentry***. In multithread environment, the sentry object destructor is responsible for unlocking the stream and the stream buffer associated with the stream.

```
operator bool();
```

If after any preparation is completed the `basic_ios` member function `good()` is true, the sentry conversion function operator `bool()` will return `true` else it will return `false`.

Extractors

```
istream_type&
operator>>(istream_type&
           (*pf) (istream_type&));
```

Calls `pf(*this)`, then return `*this`. See, for example, the function signature `ws(basic_istream&)`.

```
istream_type&
operator>>(ios_type& (*pf) (ios_type&));
```

Calls `pf(*this)`, then return `*this`.

```
istream_type&
operator>>(ios_base& (*pf) (ios_base&));
```

Calls `pf(*this)`, then return `*this`. See, for example, the function signature `dec(ios_base&)`.

```
istream_type&
operator>>(bool& n);
```

Converts a Boolean value, if one is available, and stores it in `n`. If the `ios_base` member function `flag() & ios_base::boolalpha` is false it tries to read an integer value, which if found must be 0 or 1. If the `boolalpha` flag is true, it reads characters until it determines whether the characters read are correct according to the locale function `numpunct<>::truename()` or `numpunct<>::falsename()`. If no match is found, it calls the `basic_ios` member function `setstate(failbit)`, which may throw `ios_base::failure`.

```
istream_type&
operator>>(short& n);
```

Converts a `signed short` integer, if one is available, and stores it in `n`, then returns `*this`.

```
istream_type&
operator>>(unsigned short& n);
```

Converts an `unsigned short` integer, if one is available, and stores it in `n`, then return `*this`.

`istream_type&`
`operator>>``(int& n);`

Converts a `signed integer`, if one is available, and stores it in `n`, then return `*this`.

`istream_type&`
`operator>>``(unsigned int& n);`

Converts an `unsigned integer`, if one is available, and stores it in `n`, then return `*this`.

`istream_type&`
`operator>>``(long& n);`

Converts a `signed long` integer, if one is available, and stores it in `n`, then returns `*this`.

`istream_type&`
`operator>>``(unsigned long& n);`

Converts an `unsigned long` integer, if one is available, and stores it in `n`, then returns `*this`.

`istream_type&`
`operator>>``(float& f);`

Converts a `float`, if one is available, and stores it in `f`, then returns `*this`.

`istream_type&`
`operator>>``(double& f);`

Converts a `double`, if one is available, and stores it in `f`, then returns `*this`.

`istream_type&`
`operator>>``(long double& f);`

Converts a `long double`, if one is available, and stores it in `f`, then returns `*this`.

`istream_type&`
`operator>>``(void*& p);`

Extracts a `void` pointer, if one is available, and stores it in p, then return `*this`.

`istream_type&`
`operator>>``(streambuf_type* sb);`

If `sb` is null, calls the `basic_ios` member function `setstate(badbit)`, which may throw `ios_base::failure`. Otherwise extracts characters from `*this` and inserts them in the output sequence controlled by `sb`. Characters are extracted and inserted until any of the following occurs:

- end-of-file occurs on the input sequence
- inserting in the output sequence fails
- an exception occurs

If the function stores no characters, it calls the `basic_ios` member function `setstate(failbit)`, which may throw `ios_base::failure`. If failure was due to catching an exception thrown while extracting characters from `sb` and `failbit` is on in `exception()`, then the caught exception is rethrown.

```
istream_type&
operator>>(streambuf_type& sb);
```

Extracts characters from `*this` and inserts them in the output sequence controlled by `sb`. Characters are extracted and inserted until any of the following occurs:

- end-of-file occurs on the input sequence
- inserting in the output sequence fails
- an exception occurs

If the function stores no characters, it calls the `basic_ios` member function `setstate(failbit)`, which may throw `ios_base::failure`. If failure was due to catching an exception thrown while extracting characters from `sb` and `failbit` is on in `exception()`, then the caught exception is rethrown.

Unformatted Functions

```
streamsize
gcount() const;
```

Returns the number of characters extracted by the last unformatted input member function called.

```
int_type
get();
```

Extracts a character, if one is available. Otherwise, the function calls the `basic_ios` member function `setstate(failbit)`, which may throw `ios_base::failure`. Returns the character extracted or `traits::eof()` if none is available.

```
istream_type&
get(char_type& c);
```

Extracts a character, if one is available, and assigns it to `c`. Otherwise, the function calls the `basic_ios` member function `setstate(failbit)`, which may throw `ios_base::failure`.

```
istream_type&
get(char_type* s, streamsize n,char_type delim);
```

Extracts characters and stores them into successive locations of an array whose first element is designated by `s`. Characters are extracted and stored until any of the following occurs:

- `n`-1 characters are stored
- end-of-file occurs on the input sequence

- the next available input character `== delim`.

If the function stores no characters, it calls the `basic_ios` member function `setstate(failbit)`, which may throw `ios_base::failure`. In any case, it stores a null character into the next successive location of the array.

`istream_type&`
`get``(char_type* s, streamsize n);`
Calls `get(s,n,widen('\n')).`

`istream_type&`
`get``(streambuf_type& sb,char_type delim);`
Extracts characters and inserts them in the output sequence controlled by `sb`. Characters are extracted and inserted until any of the following occurs:

- end-of-file occurs on the input sequence
- inserting in the output sequence fails
- the next available input character `== delim`.
- an exception occurs

If the function stores no characters, it calls the `basic_ios` member function `setstate(failbit)`, which may throw `ios_base::failure`. If failure was due to catching an exception thrown while extracting characters from `sb` and `failbit` is on in `exception()`, then the caught exception is rethrown.

`istream_type&`
`get``(streambuf_type& sb);`
Calls `get(sb,widen('\n')).`

`istream_type&`
`getline``(char_type* s, streamsize n, char_type delim);`
Extracts characters and stores them into successive locations of an array whose first element is designated by `s`. Characters are extracted and stored until any of the following occurs:

- `n`-1 characters are stored
- end-of-file occurs on the input sequence
- the next available input character `== delim`.

If the function stores no characters, it calls the `basic_ios` member function `setstate(failbit)`, which may throw `ios_base::failure`. In any case, it stores a null character into the next successive location of the array.

`istream_type&`
`getline``(char_type* s, streamsize n);`
Calls `getline(s,n,widen('\n')).`

`istream_type&`
`ignore``(streamsize n=1, int_type delim=traits::eof());`

Extracts characters and discards them. Characters are extracted until any of the following occurs:

- `n` characters are extracted
- end-of-file occurs on the input sequence
- the next available input character `== delim`.

`int_type`
`peek``();`

Returns `traits::eof()` if the `basic_ios` member function `good()` returns false. Otherwise, returns the next available character. Does not increment the current get pointer.

`istream_type&`
`putback``(char_type c);`

Insert `c` in the putback sequence.

`istream_type&`
`read``(char_type* s, streamsize n);`

Extracts characters and stores them into successive locations of an array whose first element is designated by `s`. Characters are extracted and stored until any of the following occurs:

- `n` characters are stored
- end-of-file occurs on the input sequence

If the function does not store `n` characters, it calls the `basic_ios` member function `setstate(failbit)`, which may throw `ios_base::failure`.

`streamsize`
`readsome``(char_type* s, streamsize n);`

Extracts characters and stores them into successive locations of an array whose first element is designated by `s`. If `rdbuf()->in_avail() == -1`, calls the `basic_ios` member function `setstate(eofbit)`.

- If `rdbuf()->in_avail() == 0`, extracts no characters
- If `rdbuf()->in_avail() > 0`, extracts `min( rdbuf()->in_avail(), n)`

In any case the function returns the number of characters extracted.

`istream_type&`
`seekg``(pos_type& pos);`

If the `basic_ios` member function `fail()` returns `false`, executes `rdbuf()->pubseekpos(pos)`, which will position the current pointer of the input sequence at the position designated by `pos`.

```
istream_type&
seekg(off_type& off, ios_base::seekdir dir);
```

If the `basic_ios` member function `fail()` returns `false`, executes `rdbuf()->pubseekpos(off,dir)`, which will position the current pointer of the input sequence at the position designated by `off` and `dir`.

```
int
sync();
```

If `rdbuf()` is a null pointer, return -1. Otherwise, calls `rdbuf()->pubsync()` and if that function returns -1 calls the `basic_ios` member function `setstate(badbit)`. The purpose of this function is to synchronize the internal input buffer, with the external sequence of characters.

```
pos_type
tellg();
```

If the `basic_ios` member function `fail()` returns `true`, `tellg()` returns `pos_type(off_type(-1))` to indicate failure. Otherwise it returns the current position of the input sequence by calling `rdbuf()->pubseekoff(0,cur,in)`.

```
istream_type&
unget();
```

If `rdbuf()` is not null, calls `rdbuf()->sungetc()`. If `rdbuf()` is null or if `sungetc()` returns `traits::eof()`, calls the `basic_ios` member function `setstate(badbit)`.

Non Member Functions

```
template<class charT, class traits>
basic_istream<charT, traits>&
operator>>(basic_istream<charT, traits>& is, charT& c);
```

Extracts a character if one is available, and stores it in `c`. Otherwise the function calls the `basic_ios` member function `setstate(failbit)`, which may throw `ios_base::failure`.

```
template<class charT, class traits>
basic_istream<charT, traits>&
operator>>(basic_istream<charT, traits>& is, charT* s);
```

Extracts characters and stores them into successive locations of an array whose first element is designated by `s`. If the `ios_base` member function `is.width()` is greater than zero, then `is.width()` is the maximum number of characters stored. Characters are extracted and stored until any of the following occurs:

- if `is.witdh()>0`, `is.witdh()-1` characters are extracted
- end-of-file occurs on the input sequence
- the next available input character is a white space.

If the function stores no characters, it calls the `basic_ios` member function `setstate(failbit)`, which may throw `ios_base::failure`. In any case, it then stores a null character into the next successive location of the array and calls `width(0)`.

```
Template<class traits>
basic_istream<char, traits>&
operator>>(basic_istream<char, traits>& is, unsigned char& c);
```

Returns `is >> (char&)c`.

```
Template<class traits>
basic_istream<char, traits>&
operator>>(basic_istream<char, traits>& is, signed char& c);
```

Returns `is >> (char&)c`.

```
Template<class traits>
basic_istream<char, traits>&
operator>>(basic_istream<char, traits>& is, unsigned char* c);
```

Returns `is >> (char*)c`.

```
Template<class traits>
basic_istream<char, traits>&
operator>>(basic_istream<char, traits>& is, signed char* c);
```

Returns `is >> (char*)c`.

```
template<class charT, class traits>
basic_istream<charT, traits>&
ws(basic_istream<charT, traits>& is);
```

Skips any white space in the input sequence and returns `is`.

Examples

```
//
// stdlib/examples/manual/istream1.cpp
//
#include<iostream>
#include<istream>
#include<fstream>

void main ( )
{
  using namespace std;

  float f= 3.14159;
  int   i= 3;
  char  s[200];

  // open a file for read and write operations
  ofstream out("example", ios_base::in | ios_base::out
               | ios_base::trunc);

  // tie the istream object to the ofstream filebuf
  istream  in (out.rdbuf());

  // output to the file
  out << "Annie is the Queen of porting" << endl;
  out << f << endl;
  out << i << endl;

  // seek to the beginning of the file
  in.seekg(0);

  f = i = 0;

  // read from the file using formatted functions
  in >> s >> f >> i;

  // seek to the beginning of the file
  in.seekg(0,ios_base::beg);

  // output the all file to the standard output
  cout << in.rdbuf();

  // seek to the beginning of the file
  in.seekg(0);

  // read the first line in the file
  // "Annie is the Queen of porting"
  in.getline(s,100);

  cout << s << endl;

  // read the second line in the file
  // 3.14159
  in.getline(s,100);

  cout << s << endl;

  // seek to the beginning of the file
  in.seekg(0);

  // read the first line in the file
  // "Annie is the Queen of porting"
  in.get(s,100);

  // remove the newline character
  in.ignore();

  cout << s << endl;

  // read the second line in the file
  // 3.14159
  in.get(s,100);

  cout << s << endl;

  // remove the newline character
  in.ignore();

  // store the current file position
  istream::pos_type position = in.tellg();

  out << "replace the int" << endl;
```

```
  // move back to the previous saved position
  in.seekg(position);

  // output the remain of the file
  // "replace the int"
  // this is equivalent to
  // cout << in.rdbuf();
  while( !char_traits<char>::eq_int_type(in.peek(),
char_traits<char>::eof()) )
   cout << char_traits<char>::to_char_type(in.get());

  cout << "\n\n\n" << flush;
}

//
// istream example two
//
#include <iostream>

void main ( )
{
  using namespace std;

  char p[50];

  // remove all the white spaces
  cin >> ws;

  // read characters from stdin until a newline
  // or 49 characters have been read
  cin.getline(p,50);

  // output the result to stdout
  cout << p;
}
```

See Also

char_traits(3C++), ***ios_base***(3C++), ***basic_ios***(3C++), ***basic_streambuf***(3C++), ***basic_iostream***(3C++)

Working Paper for Draft Proposed International Standard for Information Systems--Programming Language C++, Section 27.6.1

Standards Conformance

ANSI X3J16/ISO WG21 Joint C++ Committee

basic_istringstream → ***basic_istream*** → ***basic_ios*** → ***ios_base***

Synopsis

```
#include <sstream>
template<class charT, class traits = char_traits<charT>,
         class Allocator = allocator<void> >
class basic_istringstream
: public basic_istream<charT, traits>
```

Description

The template class ***basic_istringstream<charT,traits,Allocator>*** provides functionality to read from an array in memory. It supports reading objects of class ***basic_string<charT,traits,Allocator>***. It uses a `basic_stringbuf` object to control the associated storage. It inherits from ***basic_istream*** and therefore can use all the formatted and unformatted input functions.

Interface

```
template<class charT, class traits = char_traits<charT>,
         class Allocator = allocator<void> >
class basic_istringstream
: public basic_istream<charT, traits> {

 public:

  typedef basic_stringbuf<charT, traits, Allocator>  sb_type;
  typedef basic_ios<charT, traits>                   ios_type;

  typedef basic_string<charT, traits, Allocator>     string_type;

  typedef traits                           traits_type;
  typedef charT                            char_type;
  typedef typename traits::int_type        int_type;
  typedef typename traits::pos_type        pos_type;
  typedef typename traits::off_type        off_type;

  explicit basic_istringstream(ios_base::openmode which =
                               ios_base::in);

  explicit basic_istringstream(const string_type& str,
                               ios_base::openmode which =
                               ios_base::in);

  virtual ~basic_istringstream();

  basic_stringbuf<charT,traits,Allocator> *rdbuf() const;
  string_type str() const;

  void str(const string_type& str);

};
```

Types

char_type
The type char_type is a synonym for the template parameter charT.

int_type
The type int_type is a synonym of type traits::in_type.

ios_type
The type ios_type is an instantiation of class basic_ios on type charT.

istringstream
The type istringstream is an instantiation of class basic_istringstream on type char:

```
typedef basic_istringstream<char> istringstream;
```

off_type
The type off_type is a synonym of type traits::off_type.

pos_type
The type pos_type is a synonym of type traits::pos_type.

sb_type
The type sb_type is an instantiation of class basic_stringbuf on type charT.

string_type
The type string_type is an instantiation of class basic_string on type charT.

traits_type
The type traits_type is a synonym for the template parameter traits.

wistringstream
The type wistringstream is an instantiation of class basic_istringstream on type wchar_t:

```
typedef basic_istringstream<wchar_t> wistringstream;
```

Constructors

```
explicit basic_istringstream(ios_base::openmode which =
                    ios_base::in);
```

Constructs an object of class basic_istringstream, initializing the base class basic_istream with the associated string buffer. The string buffer is initialized by calling the basic_stringbuf constructor basic_stringbuf<charT,traits,Allocator>(which).

```
explicit basic_istringstream(const string_type& str,
                    ios_base::openmode which =
                    ios_base::in);
```

Constructs an object of class basic_istringstream, initializing the base class basic_istream with the associated string buffer. The string buffer is initialized by calling the basic_stringbuf constructor basic_stringbuf<charT,traits,Allocator>(str,which).

Destructor

```
virtual ~basic_istringstream();
```

Destroys an object of class `basic_istringstream`.

Member Functions

```
basic_stringbuf<charT,traits,Allocator>*
rdbuf() const;
```

Returns a pointer to the `basic_stringbuf` associated with the stream.

```
string_type
str() const;
```

Returns a string object of type `string_type`, which contains a copy of the underlying buffer contents.

```
void
str(const string_type& str);
```

Clears the string buffer and copies the string object `str` into it. If the opening mode is `in`, initialize the input sequence to point at the first character of the buffer. If the opening mode is `out`, initialize the output sequence to point at the first character of the buffer. If the opening mode is `out | app`, initialize the output sequence to point at the last character of the buffer.

Examples

```
//
// stdlib/examples/manual/istringstream.cpp
//
#include<iostream>
#include<sstream>
#include<string>
#include<iomanip>

void main ( )
{
  using namespace std;

  long   l= 20;
  wchar_t *ntbs=L"Il avait l'air heureux";
  wchar_t c;
  wchar_t buf[50];

  // create a read/write string-stream object on wide char
  // and attach it to an wistringstream object
  wistringstream in(ios_base::in | ios_base::out);

  // tie the ostream object to the wistringstream object
  wostream out(in.rdbuf());

  // output ntbs in out
  out << ntbs;

  // output each word on a separate line
  while ( in.get(c) )
   {
     if ( c == L' ' )
      wcout << endl;
     else
      wcout << c;
   }
```

```
wcout << endl << endl;

// move back the input sequence to the beginning
in.seekg(0);

// clear the state flags
in.clear();

// does the same thing as the previous code
// output each word on a separate line
while ( in >> buf )
 wcout << buf << endl;

wcout << endl << endl;

// create a tiny string object
string test_string("Il dormait pour l'eternite");

// create a read/write string-stream object on char
// and attach it to an istringstream object
istringstream in_bis(ios_base:: in | ios_base::out |
                     ios_base::app );

// create an ostream object
ostream out_bis(in_bis.rdbuf());

// initialize the string buffer with test_string
in_bis.str(test_string);

out_bis << endl;

// output the base info before each integer
out_bis << showbase;

ostream::pos_type pos= out_bis.tellp();

// output l in hex with a field with of 20
out_bis << hex << setw(20) << l << endl;

// output l in oct with a field with of 20
out_bis << oct << setw(20) << l << endl;

// output l in dec with a field with of 20
out_bis << dec << setw(20) << l << endl;

// output the all buffer
cout << in_bis.rdbuf();

// seek the input sequence to pos
in_bis.seekg(pos);

int a,b,d;

// read the previous outputted integer
in_bis >> a >> b >> d;

// output 3 times 20
cout << a << endl << b << endl << d << endl;
```

```
}
```

See Also

char_traits(3C++), ***ios_base***(3C++), ***basic_ios***(3C++), ***basic_stringbuf***(3C++), ***basic_string***(3C++), ***basic_ostringstream***(3C++), ***basic_stringstream***(3C++)

Working Paper for Draft Proposed International Standard for Information Systems--Programming Language C++, Section 27.7.2

Standards Conformance

ANSI X3J16/ISO WG21 Joint C++ Committee

basic_ofstream → ***basic_ostream*** → ***basic_ios*** → ***ios_base***

Synopsis

```
#include <fstream>
template<class charT, class traits = char_traits<charT> >
class basic_ofstream
: public basic_ostream<charT, traits>
```

Description

The template class ***basic_ofstream<charT,traits>*** supports writing into named files or other devices associated with a file descriptor. It uses a `basic_filebuf` object to control the associated sequences. It inherits from ***basic_ostream*** and can therefore use all the formatted and unformatted output functions.

Interface

```
 template<class charT, class traits = char_traits<charT> >
 class basic_ofstream
 : public basic_ostream<charT, traits> {

  public:

   typedef basic_ios<charT, traits>    ios_type;

   typedef charT                       char_type;
   typedef traits                      traits_type;
   typedef typename traits::int_type   int_type;
   typedef typename traits::pos_type   pos_type;
   typedef typename traits::off_type   off_type;

   basic_ofstream();

   explicit basic_ofstream(const char *s,
                           ios_base::openmode mode =
                           ios_base::out,
                           long protection = 0666);

   explicit basic_ofstream(int fd);

   basic_ofstream(int fd, char_type* buf, int len);

   virtual ~basic_ofstream();

   basic_filebuf<charT, traits> *rdbuf() const;

   bool is_open();

   void open(const char *s, ios_base::openmode mode =
             ios_type::out, long protection = 0666);

   void close();

 };
```

Types

`char_type`

The type `char_type` is a synonym for the template parameter `charT`.

`off_type`

The type `off_type` is a synonym of type `traits::off_type`.

`ofstream`

The type `ofstream` is an instantiation of class `basic_ofstream` on type `char`:

```
typedef basic_ofstream<char> ofstream;
```

`int_type`

The type `int_type` is a synonym of type `traits::in_type`.

`ios_type`

The type `ios_type` is an instantiation of class `basic_ios` on type `charT`.

`pos_type`

The type `pos_type` is a synonym of type `traits::pos_type`.

`traits_type`

The type `traits_type` is a synonym for the template parameter `traits`.

`wofstream`

The type `wofstream` is an instantiation of class `basic_ofstream` on type `wchar_t`:

```
typedef basic_ofstream<wchar_t> wofstream;
```

Constructors

```
basic_ofstream();
```

Constructs an object of class ***basic_ofstream<charT,traits>***, initializing the base class ***basic_ostream*** with the associated file buffer, which is initialized by calling the `basic_filebuf` constructor `basic_filebuf<charT,traits>()`. After construction a file can be attached to the ***basic_ofstream*** object using the `open` member function.

```
basic_ofstream(const char* s,
               ios_base::openmode mode= ios_base::in,
               long protection= 0666);
```

Constructs an object of class ***basic_ofstream<charT,traits>***, initializing the base class ***basic_ostream*** with the associated file buffer, which is initialized by calling the `basic_filebuf` constructor `basic_filebuf<charT,traits>()`. The constructor then calls the open function `open(s,mode,protection)` in order to attach the file whose name is pointed at by `s`, to the ***basic_ofstream*** object. The third argument, `protection`, is used as the file permissions. It does not appear in the Standard C++ description and is provided as an extension. It determines the file read / write / execute permissions under UNIX. It is more limited under DOS since files are always readable and do not have special execute permission.

`explicit` **`basic_ofstream`**`(int fd);`

Constructs an object of class ***basic_ofstream<charT,traits>***, initializing the base class ***basic_ostream*** with the associated file buffer, which is initialized by calling the `basic_filebuf` constructor `basic_filebuf<charT,traits>()`. The constructor then calls the `basic_filebuf` open function `open(fd)` in order to attach the file descriptor `fd` to the ***basic_ofstream*** object. This constructor is not described in the C++ standard, and is provided as an extension in order to manipulate pipes, sockets or other UNIX devices, that can be accessed through file descriptors. If the function fails, it sets `ios_base::failbit.`

`basic_ofstream``(int fd, char_type* buf,int len);`

Constructs an object of class ***basic_ofstream<charT,traits>***, initializing the base class ***basic_ostream*** with the associated file buffer, which is initialized by calling the `basic_filebuf` constructor `basic_filebuf<charT,traits>()`. The constructor then calls the `basic_filebuf` open function `open(fd)` in order to attach the file descriptor `fd` to the ***basic_ofstream*** object. The underlying buffer is then replaced by calling the `basic_filebuf` member function `setbuf` with parameters `buf` and `len`. This constructor is not described in the C++ standard, and is provided as an extension in order to manipulate pipes, sockets or other UNIX devices, that can be accessed through file descriptors. It also maintains compatibility with the old iostreams library. If the function fails, it sets `ios_base::failbit.`

Destructor

`virtual` **`~basic_ofstream`**`();`

Destroys an object of class `basic_ofstream`.

Member Functions

```
void
close();
```

Calls the associated `basic_filebuf` function `close()` and if this function fails, it calls the `basic_ios` member function `setstate(failbit)`.

```
bool
is_open();
```

Calls the associated `basic_filebuf` function `is_open()` and return its result.

```
void
open(const char* s,ios_base::openmode =
          ios_base::out, long protection = 0666);
```

Calls the associated `basic_filebuf` function `open(s,mode,protection)` and, if this function fails opening the file, calls the `basic_ios` member function `setstate(failbit)`. The third argument, `protection`, is used as the file permissions. It does not appear in the Standard C++ description and is provided as an extension. It determines the file read / write / execute permissions under UNIX, and is more limited under DOS since files are always readable and do not have special execute permission.

```
basic_filebuf<charT,traits>*
rdbuf() const;
```

Returns a pointer to the `basic_filebuf` associated with the stream.

Examples

See ***basic_fstream***, ***basic_ifstream*** and ***basic_filebuf*** examples.

See Also

char_traits(3C++), ***ios_base***(3C++), ***basic_ios***(3C++), ***basic_filebuf***(3C++), ***basic_ifstream***(3C++), ***basic_fstream***(3C++)

Working Paper for Draft Proposed International Standard for Information Systems--Programming Language C++, Section 27.8.1.8

Standards Conformance

ANSI X3J16 / ISO WG21 Joint C++ Committee

basic_ostream

basic_ostream → ***basic_ios*** → ***ios_base***

Synopsis

```
#include <ostream>
template<class charT, class traits = char_traits<charT> >
class basic_ostream
: virtual public basic_ios<charT, traits>
```

Description

The class ***basic_ostream*** defines a number of member function signatures that assist in formatting and writing output to sequences controlled by a stream buffer.

Two groups of member function signatures share common properties: the formatted output functions (or insertors) and the unformatted output functions. Both groups of functions insert characters by calling ***basic_streambuf*** member functions. They both begin by constructing an object of class ***basic_ostream::sentry*** and, if this object is in good state after construction, the function tries to perform the requested output. The `sentry` object performs exception-safe initialization, such as controlling the status of the stream or locking it in multithread environment.

Some formatted output functions generate the requested output by converting a value from some scalar to text form and inserting the converted text in the output sequence. The conversion behavior is locale dependent, and directly depend on the locale object imbued in the stream.

Interface

```
template<class charT, class traits = char_traits<charT> >
class basic_ostream
:virtual public basic_ios<charT, traits> {

 public:

  typedef basic_ostream<charT, traits>   ostream_type;
  typedef basic_ios<charT, traits>       ios_type;

  typedef traits                         traits_type;
  typedef charT                          char_type;

  typedef typename traits::int_type      int_type;
  typedef typename traits::pos_type      pos_type;
  typedef typename traits::off_type      off_type;

  explicit basic_ostream(basic_streambuf<charT, traits> *sb);
  virtual ~basic_ostream();

 class sentry {

  public:
     explicit sentry(basic_ostream<charT,traits>&);
     ~sentry();
     operator bool ();

  };

  ostream_type& operator<<(ostream_type& (*pf)(ostream_type&));
  ostream_type& operator<<(ios_base& (*pf)(ios_base&));
  ostream_type& operator<<(ios_type& (*pf)(ios_type&));

  ostream_type& operator<<(bool n);
  ostream_type& operator<<(short n);
  ostream_type& operator<<(unsigned short n);
  ostream_type& operator<<(int n);
  ostream_type& operator<<(unsigned int n);
  ostream_type& operator<<(long n);
  ostream_type& operator<<(unsigned long n);
  ostream_type& operator<<(float f);
  ostream_type& operator<<(double f);
  ostream_type& operator<<(long double f);

  ostream_type& operator<<(void *p);

  ostream_type& operator<<(basic_streambuf<char_type, traits>&
sb);
  ostream_type& operator<<(basic_streambuf<char_type, traits>
*sb);

  ostream_type& put(char_type c);

  ostream_type& write(const char_type *s, streamsize n);

  ostream_type& flush();

  ostream_type& seekp(pos_type );
  ostream_type& seekp(off_type , ios_base::seekdir );
  pos_type tellp();

 protected:

  basic_ostream();

};

//global functions

template <class charT, class traits>
basic_ostream<charT,traits>&
operator<<(basic_ostream<charT,traits>&, charT);

template <class charT, class traits>
basic_ostream<charT,traits>&
operator<<(basic_ostream<charT,traits>&, char);

template <class traits>
basic_ostream<char,traits>&
operator<<(basic_ostream<char,traits>&, char);
```

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

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

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

template <class traits>
basic_ostream<char,traits>&
operator<<(basic_ostream<char,traits>&, unsigned char);

template <class traits>
basic_ostream<char,traits>&
operator<<(basic_ostream<char,traits>&, signed char);

template <class traits>
basic_ostream<char,traits>&
operator<<(basic_ostream<char,traits>&, const unsigned char*);

template <class traits>
basic_ostream<char,traits>&
operator<<(basic_ostream<char,traits>&, const signed char*);

template<class charT, class traits>
basic_ostream<charT, traits>&
endl(basic_ostream<charT, traits>& os);

template<class charT, class traits>
basic_ostream<charT, traits>&
ends(basic_ostream<charT, traits>& os);

template<class charT, class traits>
basic_ostream<charT, traits>&
flush(basic_ostream<charT, traits>& os);
```

Types

char_type
The type `char_type` is a synonym for the template parameter `charT`.

int_type
The type `int_type` is a synonym of type `traits::in_type`.

ios_type
The type `ios_type` is a synonym for `basic_ios<charT, traits>`.

off_type
The type `off_type` is a synonym of type `traits::off_type`.

ostream
The type `ostream` is an instantiation of class `basic_ostream` on type `char`:

```
typedef basic_ostream<char> ostream;
```

`ostream_type`

The type `ostream_type` is a synonym for `basic_ostream<charT, traits>`.

`pos_type`

The type `pos_type` is a synonym of type `traits::pos_type`.

`traits_type`

The type `traits_type` is a synonym for the template parameter `traits`.

`wostream`

The type `wostream` is an instantiation of class `basic_ostream` on type `wchar_t`:

```
typedef basic_ostream<wchar_t> wostream;
```

Constructor

`explicit` **`basic_ostream`**`(basic_streambuf<charT, traits>* sb);`

Constructs an object of class `basic_ostream`, assigning initial values to the base class by calling `basic_ios<charT,traits>::init(sb)`.

Destructor

`virtual` **`~basic_ostream`**`();`

Destroys an object of class `basic_ostream`.

Sentry Class

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

Prepares for formatted or unformatted output. First if the `basic_ios` member function `tie()` is not a null pointer, the function synchronizes the output sequence with any associated stream. If after any preparation is completed the `basic_ios` member function `good()` is true, the sentry conversion function operator `bool ()` will return `true`. Otherwise it will return `false`. In multithread environment the sentry object constructor is responsible for locking the stream and the stream buffer associated with the stream.

`~sentry``();`

Destroys an object of class `sentry`. If the `ios_base` member function `flags() & unitbuf == true`, then flush the buffer. In multithread environment the sentry object destructor is responsible for unlocking the stream and the stream buffer associated with the stream.

`operator` **`bool`**`();`

If after any preparation is completed the `ios_base` member function `good()` is true, the sentry conversion function operator `bool()` will return true else it will return false.

Insertors

`ostream_type&`
`operator<<``(ostream_type& (*pf) (ostream_type&));`
Calls `pf(*this)`, then return `*this`. See, for example, the function signature `endl(basic_ostream&)`.

`ostream_type&`
`operator<<``(ios_type& (*pf) (ios_type&));`
Calls `pf(*this)`, then return `*this`.

`ostream_type&`
`operator<<``(ios_base& (*pf) (ios_base&));`
Calls `pf(*this)`, then return `*this`. See, for example, the function signature `dec(ios_base&)`.

`ostream_type&`
`operator<<``(bool n);`
Converts the boolean value `n`, and outputs it into the `basic_ostream` object's buffer. If the `ios_base` member function `flag() & ios_base::boolalpha` is false it tries to write an integer value, which must be 0 or 1. If the `boolalpha` flag is true, it writes characters according to the locale function `numpunct<>::truename()` or `numpunct<>::falsename()`.

`ostream_type&`
`operator<<``(short n);`
Converts the `signed short` integer `n`, and output it into the stream buffer, then return `*this`.

`ostream_type&`
`operator<<``(unsigned short n);`
Converts the `unsigned short` integer `n`, and output it into the stream buffer, then return `*this`.

`ostream_type&`
`operator<<``(int n);`
Converts the `signed integer n`, and output it into the stream buffer, then return `*this`.

`ostream_type&`
`operator<<``(unsigned int n);`
Converts the `unsigned integer n`, and output it into the stream buffer, then return `*this`.

`ostream_type&`
`operator<<``(long n);`
Converts the `signed long` integer `n`, and output it into the stream buffer, then return `*this`.

`ostream_type&`
`operator<<``(unsigned long n);`

Converts the `unsigned long` integer n, and output it into the stream buffer, then return `*this`.

`ostream_type&`
`operator<<``(float f);`

Converts the `float f` and output it into the stream buffer, then return `*this`.

`ostream_type&`
`operator<<``(double f);`

Converts the `double f` and output it into the stream buffer, then return `*this`.

`ostream_type&`
`operator<<``(long double f);`

Converts the `long double f` and output it into the stream buffer, then return `*this`.

`ostream_type&`
`operator<<``(void *p);`

Converts the `pointer p`, and output it into the stream buffer, then return `*this`.

`ostream_type&`
`operator<<``(basic_streambuf<charT,traits> *sb);`

If `sb` is null calls the `basic_ios` member function `setstate(badbit)`. Otherwise gets characters from `sb` and inserts them into the stream buffer until any of the following occurs:

- end-of-file occurs on the input sequence.
- inserting in the output sequence fails
- an exception occurs while getting a character from `sb`

If the function inserts no characters or if it stopped because an exception was thrown while extracting a character, it calls the `basic_ios` member function `setstate(failbit)`. If an exception was thrown while extracting a character it is rethrown.

`ostream_type&`
`operator<<``(basic_streambuf<charT,traits>& sb);`

Gets characters from `sb` and inserts them into the stream buffer until any of the following occurs:

- end-of-file occurs on the input sequence.
- inserting in the output sequence fails
- an exception occurs while getting a character from `sb`

If the function inserts no characters or if it stopped because an exception was thrown while extracting a character, it calls the `basic_ios` member function `setstate(failbit)`. If an exception was thrown while extracting a character it is rethrown.

Unformatted Functions

`ostream_type&`
`flush``();`

If `rdbuf()` is not a null pointer, calls `rdbuf()->pubsync()` and returns `*this`. If that function returns -1 calls `setstate(badbit)`.

`ostream_type&`
`put``(char_type c);`

Inserts the character `c`. If the operation fails, calls the `basic_ios` member function `setstate(badbit)`.

`ostream_type&`
`seekp``(pos_type pos);`

If the `basic_ios` member function `fail()` returns `false`, executes `rdbuf()->pubseekpos(pos)`, which will position the current pointer of the output sequence at the position designated by `pos`.

`ostream_type&`
`seekp``(off_type off, ios_base::seekdir dir);`

If the `basic_ios` member function `fail()` returns `false`, executes `rdbuf()->pubseekpos(off,dir)`, which will position the current pointer of the output sequence at the position designated by `off` and `dir`.

`pos_type`
`tellp``();`

If the `basic_ios` member function `fail()` returns `true`, `tellp()` returns `pos_type(off_type(-1))` to indicate failure. Otherwise it returns the current position of the output sequence by calling `rdbuf()-> pubseekoff(0,cur, out)`.

`ostream_type&`
`write``(const char_type* s, streamsize n);`

Obtains characters to insert from successive locations of an array whose first element is designated by `s`. Characters are inserted until either of the following occurs:

- n characters are inserted
- inserting in the output sequence fails

In the second case the function calls the `basic_ios` member function `setstate(badbit)`. The function returns `*this`.

Non Member Functions

```
template<class charT, class traits>
basic_ostream<charT, traits>&
endl(basic_ostream<charT, traits>& os);
```

Outputs a `newline` character and flush the buffer, then returns `os`.

```
template<class charT, class traits>
basic_ostream<charT, traits>&
ends(basic_ostream<charT, traits>& os);
```

Inserts a null character into the output sequence, then returns `os`.

```
template<class charT, class traits>
basic_ostream<charT, traits>&
flush(basic_ostream<charT, traits>& os);
```

Flushes the buffer, then returns `os`.

```
template<class charT, class traits>
basic_ostream<charT, traits>&
operator<<(basic_ostream<charT, traits>& os, charT c);
```

Outputs the character `c` of type `charT` into the `basic_ostream` object's buffer. Both the stream and the stream buffer are instantiated on type `charT`. Padding is not ignored.

```
template<class charT, class traits>
basic_ostream<charT, traits>&
operator<<(basic_ostream<charT, traits>& os, char c);
```

Outputs the character `c` of type `char` into the `basic_ostream` object's buffer. Both the stream and the stream buffer are instantiated on type `charT`. Conversion from characters of type `char` to characters of type `charT` is performed by the `basic_ios` member function `widen`. padding is not ignored.

```
template<class traits>
basic_ostream<char, traits>&
operator<<(basic_ostream<char, traits>& os, char c);
```

Outputs the character `c` of type `char` into the `basic_ostream` object's buffer. Both the stream and the stream buffer are instantiated on type `char`. Padding is not ignored.

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

Outputs the null-terminated-byte-string `s` of type `charT*` into the `basic_ostream` object's buffer. Both the stream and the stream buffer are instantiated on type `charT`.

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

Outputs the null-terminated-byte-string `s` of type `char*` into the `basic_ostream` object's buffer. Both the stream and the stream buffer are instantiated on type `charT`. Conversion from characters of type `char` to characters of type charT is performed by the basic_ios member function widen.

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

Outputs the null-terminated-byte-string s of type char* into the basic_ostream object's buffer. Both the stream and the stream buffer are instantiated on type char.

```
template<class traits>
basic_ostream<char, traits>&
operator<<(basic_ostream<char, traits>& os, unsigned char c);
```

Returns os << (char)c.

```
template<class traits>
basic_ostream<char, traits>&
operator<<(basic_ostream<char, traits>& os, signed char c);
```

Returns os << (char)c.

```
template<class traits>
basic_ostream<char, traits>&
operator<<(basic_ostream<char, traits>& os, unsigned char* c);
```

Returns os << (char*)c.

```
template<class traits>
basic_ostream<char, traits>&
operator<<(basic_ostream<char, traits>& os, signed char* c);
```

Returns os << (char*)c.

Formatting

The formatting is done through member functions or manipulators.

Manipulators	Member Functions
showpos	setf(ios_base::showpos)
noshowpos	unsetf(ios_base::showpos)
showbase	setf(ios_base::showbase)
noshowbase	unsetf(ios_base::showbase)
uppercase	setf(ios_base::uppercase)
nouppercase	unsetf(ios_base::uppercase)
showpoint	setf(ios_base::showpoint)
noshowpoint	unsetf(ios_base::showpoint)
boolalpha	setf(ios_base::boolalpha)
noboolalpha	unsetf(ios_base::boolalpha)
unitbuf	setf(ios_base::unitbuf)
nounitbuf	unsetf(ios_base::unitbuf)
internal	setf(ios_base::internal, ios_base::adjustfield)
left	setf(ios_base::left, ios_base::adjustfield)

Manipulators	Member Functions
`right`	`setf(ios_base::right, ios_base::adjustfield)`
`dec`	`setf(ios_base::dec, ios_base::basefield)`
`hex`	`setf(ios_base::hex, ios_base::basefield)`
`oct`	`setf(ios_base::oct, ios_base::basefield)`
`fixed`	`setf(ios_base::fixed, ios_base::floatfield)`
`scientific`	`setf(ios_base::scientific, ios_base::floatfield)`
`resetiosflags (ios_base::fmtflags flag)`	`setf(0,flag)`
`setiosflags (ios_base::fmtflags flag)`	`setf(flag)`
`setbase(int base)`	see above
`setfill(char_type c)`	`fill(c)`
`setprecision(int n)`	`precision(n)`
`setw(int n)`	`width(n)`

Description

`showpos`	Generates a + sign in non-negative generated numeric output
`showbase`	Generates a prefix indicating the numeric base of generated integer output
`uppercase`	Replaces certain lowercase letters with their uppercase equivalents in generated output
`showpoint`	Generates a decimal-point character unconditionally in generated floating-point output
`boolalpha`	Insert and extract bool type in alphabetic format
`unitbuf`	Flushes output after each output operation
`internal`	Adds fill characters at a designated internal point in certain generated output, or identical to right if no such point is designated
`left`	Adds fill characters on the right (final positions) of certain generated output
`right`	Adds fill characters on the left (initial positions) of certain generated output

`dec`	Converts integer input or generates integer output in decimal base
`hex`	Converts integer input or generates integer output in hexadecimal base
`oct`	Converts integer input or generates integer output in octal base
`fixed`	Generates floating-point output in fixed-point notation
`scientific`	Generates floating-point output in scientific notation
`resetiosflagss (ios_base::fmtflags flag)`	Resets the `fmtflags` field `flag`
`setiosflags (ios_base::fmtflags flag)`	Set up the flag `flag`
`setbase(int base)`	Converts integer input or generates integer output in base `base`. The parameter `base` can be 8, 10 or 16.
`setfill(char_type c)`	Set the character used to pad (fill) an output conversion to the specified field width
`setprecision(int n)`	Set the precision (number of digits after the decimal point) to generate on certain output conversions
`setw(int n)`	Set the field with (number of characters) to generate on certain output conversions

Examples

```
//
// stdlib/examples/manual/ostream1.cpp
//
#include<iostream>
#include<ostream>
#include<sstream>
#include<iomanip>

void main ( )
{
   using namespace std;

   float f= 3.14159;
   int   i= 22;
   char* s= "Randy is the king of stdlib";

   // create a read/write stringbuf object on tiny char
   // and attach it to an istringstream object
   istringstream in( ios_base::in | ios_base::out );

   // tie the ostream object to the istringstream object
   ostream out(in.rdbuf());

   out << "test beginning !" << endl;

   // output i in hexadecimal
   out << hex << i <<endl;

   // set the field width to 10
   // set the padding character to '@'
   // and output i in octal
   out << setw(10) << oct << setfill('@') << i << endl;

   // set the precision to 2 digits after the separator
   // output f
   out << setprecision(3) << f << endl;

   // output the 17 first characters of s
   out.write(s,17);

   // output a newline character
   out.put('\n');

   // output s
   out << s << endl;

   // output the all buffer to standard output
   cout << in.rdbuf();
}


//
// stdlib/examples/manual/ostream2.cpp
//
#include<iostream>
#include<ostream>
#include<sstream>

void main ( )
{
   using namespace std;

   float f= 3.14159;
   wchar_t* s= L"Kenavo !";

   // create a read/write stringbuf object on wide char
   // and attach it to an wistringstream object
   wistringstream in( ios_base::in | ios_base::out );

   // tie the wostream object to the wistringstream object
   wostream out(in.rdbuf());

   out << L"test beginning !" << endl;

   // output f in scientific format
   out << scientific << f <<endl;

   // store the current put-pointer position
   wostream::pos_type pos = out.tellp();

   // output s
   out << s << endl;

   // output the all buffer to standard output
```

```
     wcout << in.rdbuf() << endl;

     // position the get-pointer
     in.seekg(pos);

     // output s
     wcout << in.rdbuf() << endl;
  }
```

See Also

char_traits(3C++), ***ios_base***(3C++), ***basic_ios***(3C++), ***basic_streambuf***(3C++), ***basic_iostream***(3C++)

Working Paper for Draft Proposed International Standard for Information Systems--Programming Language C++, Section 27.6.2.1

Standards Conformance

ANSI X3J16/ISO WG21 Joint C++ Committee

basic_ostringstream → ***basic_ostream*** → ***basic_ios*** → ***ios_base***

Synopsis

```
#include <sstream>
template<class charT, class traits = char_traits<charT>,
         class Allocator = allocator<void> >
class basic_ostringstream
: public basic_ostream<charT, traits>
```

Description

The template class ***basic_ostringstream<charT,traits,Allocator>*** provides functionality to write to an array in memory. It supports writing objects of class ***basic_string<charT,traits,Allocator>***. It uses a ***basic_stringbuf*** object to control the associated storage. It inherits from ***basic_ostream*** and therefore can use all the formatted and unformatted output functions.

Interface

```
template<class charT, class traits = char_traits<charT>,
         class Allocator = allocator<void> >
class basic_ostringstream
: public basic_ostream<charT, traits> {

 public:

  typedef basic_stringbuf<charT, traits, Allocator> sb_type;
  typedef basic_ios<charT, traits>                  ios_type;

  typedef basic_string<charT, traits, Allocator>    string_type;

  typedef traits                           traits_type;
  typedef charT                            char_type;
  typedef typename traits::int_type        int_type;
  typedef typename traits::pos_type        pos_type;
  typedef typename traits::off_type        off_type;

  explicit basic_ostringstream(ios_base::openmode which =
                               ios_base::out);

  explicit basic_ostringstream(const string_type& str,
                               ios_base::openmode which =
                               ios_base::out);

  virtual ~basic_ostringstream();

  basic_stringbuf<charT,traits,Allocator> *rdbuf() const;
  string_type str() const;

  void str(const string_type& str);

};
```

Types

`char_type`
The type `char_type` is a synonym for the template parameter `charT`.

`int_type`
The type `int_type` is a synonym of type `traits::in_type`.

`ios_type`
The type `ios_type` is an instantiation of class `basic_ios` on type `charT`.

`off_type`
The type `off_type` is a synonym of type `traits::off_type`.

`ostringstream`
The type `ostringstream` is an instantiation of class `basic_ostringstream` on type `char`:

```
typedef basic_ostringstream<char> ostringstream;
```

`pos_type`
The type `pos_type` is a synonym of type `traits::pos_type`.

`sb_type`
The type `sb_type` is an instantiation of class `basic_stringbuf` on type `charT`.

`string_type`
The type `string_type` is an instantiation of class `basic_string` on type `charT`.

`traits_type`
The type `traits_type` is a synonym for the template parameter `traits`.

`wostringstream`
The type `wostringstream` is an instantiation of class `basic_ostringstream` on type `wchar_t`:

```
typedef basic_ostringstream<wchar_t> wostringstream;
```

Constructors

```
explicit basic_ostringstream(ios_base::openmode which =
                              ios_base::out);
```

Constructs an object of class `basic_ostringstream`, initializing the base class `basic_ostream` with the associated string buffer. The string buffer is initialized by calling the `basic_stringbuf` constructor `basic_stringbuf<charT,traits,Allocator>(which)`.

```
explicit basic_ostringstream(const string_type& str,
                              ios_base::openmode which =
                              ios_base::out);
```

Constructs an object of class `basic_ostringstream`, initializing the base class `basic_ostream` with the associated string buffer. The string buffer is initialized by calling the `basic_stringbuf` constructor `basic_stringbuf<charT,traits,Allocator>(str,which)`.

Destructor

```
virtual ~basic_ostringstream();
```

Destroys an object of class `basic_ostringstream`.

Member Functions

```
basic_stringbuf<charT,traits,Allocator>*
rdbuf() const;
```

Returns a pointer to the `basic_stringbuf` associated with the stream.

```
string_type
str() const;
```

Returns a string object of type `string_type` whose contents is a copy of the underlying buffer contents.

```
void
str(const string_type& str);
```

Clears the underlying string buffer and copies the string object `str` into it. If the opening mode is `in`, initialize the input sequence to point at the first character of the buffer. If the opening mode is `out`, initialize the output sequence to point at the first character of the buffer. If the opening mode is `out | app`, initialize the output sequence to point at the last character of the buffer.

Examples

See ***basic_stringstream***, ***basic_istringstream*** and ***basic_stringbuf*** examples.

See Also

char_traits(3C++), ***ios_base***(3C++), ***basic_ios***(3C++), ***basic_stringbuf***(3C++), ***basic_string***(3C++), ***basic_istringstream***(3C++), ***basic_stringstream***(3C++)

Working Paper for Draft Proposed International Standard for Information Systems--Programming Language C++, Section 27.7.2.3

Standards Conformance

ANSI X3J16/ISO WG21 Joint C++ Committee

basic_streambuf

Base Class

Synopsis

```
#include <streambuf>
template<class charT, class traits = char_traits<charT> >
class basic_streambuf;
```

Description

The class template ***basic_streambuf<charT,traits>*** serves as an abstract base class for deriving various stream buffers whose objects each control two character sequences:

- A character input sequence;
- A character output sequence.

Each sequence is characterized by three pointers which, if non-null, all point into the same `charT` array object. The array object represents, at any moment, a subsequence of characters from the sequence. Operations performed on a sequence alter the values stored in these pointers, perform reads and writes directly to or from associated sequences, and alter "the stream position" and conversion state as needed to maintain this subsequence relationship. The three pointers are:

- The beginning pointer, or lowest element address in the array;
- The next pointer, or next element address that is a current candidate for reading or writing;
- The end pointer, or first element address beyond the end of the array.

Stream buffers can impose various constraints on the sequences they control, including:

- The controlled input sequence may be unreadable;
- The controlled output sequence may be unwritable;
- The controlled sequences can be associated with the contents of other representations for character sequences, such as external files;
- The controlled sequences can impose limitations on how the program can read characters from a sequence, write characters to a sequence, put characters back into an input sequence, or alter the stream position.

Interface

```
template<class charT, class traits = char_traits<charT> >
class basic_streambuf {

public:
 typedef charT                       char_type;
 typedef traits                      traits_type;

 typedef typename traits::int_type    int_type;
 typedef typename traits::pos_type    pos_type;
 typedef typename traits::off_type    off_type;


 virtual ~basic_streambuf();

 locale pubimbue( const locale& loc);
 locale getloc() const;

 basic_streambuf<char_type, traits> *
   pubsetbuf(char_type *s, streamsize n);

 pos_type pubseekoff(off_type off, ios_base::seekdir way,
                     ios_base::openmode which =
                     ios_base::in | ios_base::out);

 pos_type pubseekpos(pos_type sp, ios_base::openmode which =
                     ios_base::in | ios_base::out);

 int pubsync();

 ios_base::openmode which_open_mode();

 streamsize   in_avail();

 int_type snextc();

 int_type sbumpc();

 int_type sgetc();

 streamsize sgetn(char_type *s, streamsize n);

 int_type sputbackc(char_type c);

 int sungetc();

 int_type sputc(char_type c);

 streamsize sputn(const char_type *s, streamsize n);

protected:

 basic_streambuf();

 char_type *eback() const;
 char_type *gptr() const;
 char_type *egptr() const;

 void gbump(int n);

 void setg(char_type *gbeg_arg,char_type *gnext_arg,
           char_type *gend_arg);
```

```
    char_type *pbase() const;
    char_type *pptr() const;
    char_type *epptr() const;

    void pbump(int n);

    void setp(char_type *pbeg_arg,char_type *pend_arg);

    virtual void imbue( const locale& loc);

    virtual int_type overflow(int_type c = traits::eof());

    virtual int_type pbackfail(int_type c = traits::eof());

    virtual int showmanyc();

    virtual int_type underflow();

    virtual int_type uflow();

    virtual streamsize xsgetn(char_type *s, streamsize n);

    virtual streamsize xsputn(const char_type *s, streamsize n);

    virtual pos_type seekoff(off_type off,ios_base::seekdir way,
                             ios_base::openmode which =
                             ios_base::in | ios_base::out);

    virtual pos_type seekpos(pos_type sp,ios_base::openmode which =
                             ios_base::in | ios_base::out);

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

    virtual int sync();

  };
```

Types

char_type
The type `char_type` is a synonym for the template parameter `charT`.

int_type
The type `int_type` is a synonym of type `traits::in_type`.

off_type
The type `off_type` is a synonym of type `traits::off_type`.

pos_type
The type `pos_type` is a synonym of type `traits::pos_type`.

`streambuf`

The type `streambuf` is an instantiation of class `basic_streambuf` on type `char`:

```
typedef basic_streambuf<char> streambuf;
```

`traits_type`

The type `traits_type` is a synonym for the template parameter `traits`.

`wstreambuf`

The type `wstreambuf` is an instantiation of class `basic_streambuf` on type `wchar_t`:

```
typedef basic_streambuf<wchar_t> wstreambuf;
```

Public Constructor

`basic_streambuf``();`

Constructs an object of class `basic_streambuf` and initializes all its pointer member objects to null pointers and the `getloc()` member function to return the value of `locale::locale()`.

Public Destructor

`virtual` **`~basic_streambuf`**`();`

Destroys an object of class `basic_streambuf`.

Public Member Functions

```
locale
getloc() const;
```

If `pubimbue()` has ever been called, returns the last value of `loc` supplied, otherwise, the default locale `locale::locale()` in effect at the time of construction.

```
streamsize
in_avail();
```

If a read position is available, returns the number of available character in the input sequence. Otherwise calls the protected function `showmanyc()`.

```
locale
pubimbue(const locale& loc);
```

Calls the protected function `imbue(loc)`.

```
pos_type
pubseekoff(off_type off, ios_base::seekdir way,
           ios_base::openmode which =
           ios_base::in | ios_base::out );
```

Calls the protected function `seekoff(off,way,which)`.

```
pos_type
pubseekpos(pos_type sp, ios_base::openmode which=
           ios_base::in | ios_base::out );
```

Calls the protected function `seekpos(sp,which)`.

`basic_streambuf<char_type,traits>*`
`pubsetbuf``(char_type* s,streamsize n);`
Calls the protected function `setbuf(s,n)` .

`int`
`pubsync``();`
Calls the protected function `sync()`.

`int_type`
`sbumpc``();`
If the input sequence read position is not available, calls the function `uflow()`, otherwise returns `*gptr()` and increments the next pointer for the input sequence.

`int_type`
`sgetc``();`
If the input sequence read position is not available, calls the protected function `underflow()`, otherwise returns `*gptr()` .

`streamsize`
`sgetn``(char_type* s, streamsize n);`
Calls the protected function `xsgetn(s,n)`.

`int_type`
`snextc``();`
Calls the function `sbumpc()` and if it returns `traits::eof()`, returns `traits::eof()`; otherwise calls the function `sgetc()` .

`int_type`
`sputbackc``(char_type c);`
If the input sequence putback position is not available, or if `traits::eq(c,gptr() [-1])` returns false, calls the protected function `pbackfail(c)`, otherwise decrements the next pointer for the input sequence and returns `*gptr()`.

`int_type`
`sputc``(char_type c);`
If the output sequence write position is not available, calls the protected function `overflow(traits::to_int_type( c ))`. Otherwise, stores `c` at the next pointer for the output sequence, increments the pointer, and returns `*pptr()` .

`streamsize`
`sputn``(const char_type* s, streamsize n);`
Calls the protected function `xsputn(s,n)`.

`int_type`
`sungetc``();`

If the input sequence putback position is not available, calls the protected function `pbackfail()`. Otherwise decrements the next pointer for the input sequence and returns `*gptr()`.

`ios_base::openmode`
`which_open_mode``();`

Returns the mode in which the stream buffer is opened. This function is not described in the C++ standard.

Protected Member Functions

`char_type*`
`eback``() const;`

Returns the beginning pointer for the input sequence.

`char_type*`
`egptr``() const;`

Returns the end pointer for the input sequence.

`char_type*`
`epptr``() const;`

Returns the end pointer for the output sequence.

`void`
`gbump``(int n);`

Advances the next pointer for the input sequence by `n`.

`char_type*`
`gptr``() const;`

Returns the next pointer for the input sequence.

`void`
`imbue``(const locale&);`

Changes any translations based on locale. The default behavior is to do nothing, this function has to be overloaded in the classes derived from `basic_streambuf`. The purpose of this function is to allow the derived class to be informed of changes in locale at the time they occur. The new imbued locale object is only used by the stream buffer, it does not affect the stream itself.

`int_type`
`overflow``(int_type c = traits::eof() );`

The member functions `sputc()` and `sputn()` call this function in case that not enough room can be found in the put buffer to accommodate the argument character sequence. The function returns `traits::eof()` if it fails to make more room available, or to empty the buffer by writing the characters to their output device.

```
int_type
pbackfail(int_type c = traits::eof() );
```

If `c` is equal to `traits::eof()`, `gptr()` is moved back one position otherwise `c` is prepended. The function returns `traits::eof()` to indicate failure.

```
char_type*
pbase() const;
```

Returns the beginning pointer for the output sequence.

```
void
pbump(int n);
```

Advances the next pointer for the output sequence by `n`.

```
char_type*
pptr() const;
```

Returns the next pointer for the output sequence.

```
pos_type
seekoff(off_type off, ios_base::seekdir way,
        ios_base::openmode which =
        ios_base::in | ios_base::out );
```

Alters the stream positions within one or more of the controlled sequences in a way that is defined separately for each class derived from `basic_streambuf`. The default behavior is to return an object of type `pos_type` that stores an invalid stream position.

```
pos_type
seekpos(pos_type sp, ios_base::openmode which=
        ios_base::in | ios_base::out );
```

Alters the stream positions within one or more of the controlled sequences in a way that is defined separately for each class derived from `basic_streambuf`. The default behavior is to return an object of class `pos_type` that stores an invalid stream position.

```
basic_streambuf*
setbuf(char_type* s, streamsize n);
```

Performs an operation that is defined separately for each class derived from `basic_streambuf`. The purpose of this function is to allow the user to provide his own buffer, or to resize the current buffer.

```
void
setg(char_type* gbeg, char_type* gnext, char_type* gend);
```

Sets up private member for the following to be true:

```
eback() == gbeg, gptr() == gnext and egptr() == gend
```

`void`
`setp``(char_type* pbeg, char_type* pend);`

Sets up private member for the following to be true:

`pbase() == pbeg, pptr() == pbeg and epptr() == pend`

`int`
`showmanyc``();`

Returns the number of characters available in the internal buffer, or -1.

`int`
`sync``();`

Synchronizes the controlled sequences with the internal buffer, in a way that is defined separately for each class derived from `basic_streambuf`. The default behavior is to do nothing. On failure the return value is -1.

`int_type`
`underflow``();`

The public members of `basic_streambuf` call this function only if `gptr()` is null or `gptr() >= egptr()`. This function returns the character pointed at by `gptr()` if `gptr()` is not null and if `gptr() < egptr()`. Otherwise the function try to read character into the buffer, and if it fails return `traits::eof()`.

`int_type`
`uflow``();`

Calls `underflow()` and if `underflow()` returns `traits::eof()`, returns `traits::eof()`. Otherwise, does `gbump(1)` and returns the value of `*gptr()`.

`streamsize`
`xsgetn``(char_type* s, streamsize n);`

Assigns up to `n` characters to successive elements of the array whose first element is designated by `s`. The characters are read from the input sequence. Assigning stops when either `n` characters have been assigned or a call to `sbumpc()` would return `traits::eof()`. The function returns the number of characters read.

`streamsize`
`xsputn``(const char_type* s, streamsize n);`

Writes up to `n` characters to the output sequence. The characters written are obtained from successive elements of the array whose first element is designated by `s`. Writing stops when either `n` characters have been written or a call to `sputc()` would return `traits::eof()`. The function returns the number of characters written.

See Also

char_traits(3C++), ***basic_filebuf***(3C++), ***basic_stringbuf***(3C++), ***strstreambuf***(3C++)

Working Paper for Draft Proposed International Standard for Information Systems--Programming Language C++, Section 27.5

Standards Conformance

ANSI X3J16/ISO WG21 Joint C++ Committee

basic_stringbuf → ***basic_streambuf***

Synopsis

```
#include <sstream>
template<class charT, class traits = char_traits<charT>,
         class Allocator = allocator<void> >
class basic_stringbuf
: public basic_streambuf<charT, traits>
```

Description

The class ***basic_stringbuf*** is derived from ***basic_streambuf***. Its purpose is to associate the input or output sequence with a sequence of arbitrary characters. The sequence can be initialized from, or made available as, an object of class ***basic_string***. Each object of type ***basic_stringbuf<charT, traits,Allocator>*** controls two character sequences:

- A character input sequence;
- A character output sequence.

Note: see ***basic_streambuf.***

The two sequences are related to each other, but are manipulated separately. This allows you to read and write characters at different positions in objects of type ***basic_stringbuf*** without any conflict (as opposed to the ***basic_filebuf*** objects, in which a joint file position is maintained).

Interface

```
template<class charT, class traits = char_traits<charT>,
         class allocator<void> >
class basic_stringbuf
: public basic_streambuf<charT, traits> {

  public:

   typedef basic_ios<charT, traits>     ios_type;

   typedef basic_string<charT, traits,
                        Allocator>      string_type;

   typedef charT                        char_type;
   typedef typename traits::int_type    int_type;
   typedef typename traits::pos_type    pos_type;
   typedef typename traits::off_type    off_type;


   explicit basic_stringbuf(ios_base::openmode which =
                           (ios_base::in | ios_base::out));

   explicit basic_stringbuf(const string_type& str,
                           ios_base::openmode which =
                           (ios_base::in | ios_base::out));
```

```
    virtual ~basic_stringbuf();

    string_type str() const;
    void str(const string_type& str_arg);

  protected:

    virtual int_type overflow(int_type c = traits::eof());

    virtual int_type pbackfail(int_type c = traits::eof());

    virtual int_type underflow();

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

    virtual pos_type seekoff(off_type off, ios_base::seekdir way,
                             ios_base::openmode which =
                             ios_base::in | ios_base::out);

    virtual pos_type seekpos(pos_type sp,
                             ios_base::openmode which =
                             ios_base::in | ios_base::out);

    virtual streamsize xsputn(const char_type* s, streamsize n);

 };
```

Types

char_type
The type char_type is a synonym for the template parameter charT.

ios_type
The type ios_type is an instantiation of class basic_ios on type charT.

off_type
The type off_type is a synonym of type traits::off_type.

pos_type
The type pos_type is a synonym of type traits::pos_type.

string_type
The type string_type is an instantiation of class basic_string on type charT.

stringbuf
The type stringbuf is an instantiation of class basic_stringbuf on type char:

```
typedef basic_stringbuf<char> stringbuf;
```

traits_type
The type traits_type is a synonym for the template parameter traits.

wstringbuf

The type wstringbuf is an instantiation of class basic_stringbuf on type wchar_t:

```
typedef basic_stringbuf<wchar_t> wstringbuf;
```

Constructors

```
explicit basic_stringbuf(ios_base::openmode which =
                ios_base::in | ios_base::out);
```

Constructs an object of class basic_stringbuf, initializing the base class with basic_streambuf(), and initializing the open mode with which.

```
explicit basic_stringbuf(const string_type& str,
                ios_base::openmode which =
                ios_base::in | ios_base::out);
```

Constructs an object of class basic_stringbuf, initializing the base class with basic_streambuf(), and initializing the open mode with which. The string object str is copied to the underlying buffer. If the opening mode is in, initialize the input sequence to point at the first character of the buffer. If the opening mode is out, it initializes the output sequence to point at the first character of the buffer. If the opening mode is out | app, it initializes the output sequence to point at the last character of the buffer.

Destructor

```
virtual ~basic_stringbuf();
```

Destroys an object of class basic_stringbuf.

Member Functions

```
int_type
overflow(int_type c = traits::eof() );
```

If the output sequence has a put position available, and c is not traits::eof(), then this functions writes c into it. If there is no position available, the function increases the size of the buffer by allocating more memory and then writes c at the new current put position. If the operation fails, the function returns traits::eof(). Otherwise it returns traits::not_eof(c).

```
int_type
pbackfail( int_type c = traits::eof() );
```

Puts back the character designated by c into the input sequence. If traits::eq_int_type(c,traits::eof()) returns true, the function moves the input sequence one position backward. If the operation fails, the function returns traits::eof(). Otherwise it returns traits::not_eof(c).

```
pos_type
seekoff(off_type off, ios_base::seekdir way,
        ios_base::openmode which =
        ios_base::in | ios_base::out);
```

If the open mode is in | out, this function alters the stream position of both the input and the output sequences. If the open mode is in, it alters the stream position of only the input sequence, and if it is out, it alters the stream position of the output sequence. The new position is calculated by combining the two parameters `off` (displacement) and `way` (reference point). If the current position of the sequence is invalid before repositioning, the operation fails and the return value is `pos_type(off_type(-1))`. Otherwise the function returns the current new position.

```
pos_type
seekpos(pos_type sp,ios_base::openmode which =
        ios_base::in | ios_base::out);
```
If the open mode is `in | out`, `seekpos()` alters the stream position of both the input and the output sequences. If the open mode is `in`, it alters the stream position of the input sequence only, and if the open mode is `out`, it alters the stream position of the output sequence only. If the current position of the sequence is invalid before repositioning, the operation fails and the return value is `pos_type(off_type(-1))`. Otherwise the function returns the current new position.

```
basic_streambuf<charT,traits>*
setbuf(char_type*s, streamsize n);
```
If the string buffer object is opened in output mode, proceed as follows: if `s` is not a null pointer and `n` is greater than the content of the current buffer, replace it (copy its contents) by the buffer of size `n` pointed at by `s`. In the case where `s` is a null pointer and `n` is greater than the content of the current buffer, resize it to size `n`. If the function fails, it returns a null pointer.

```
string_type
str() const;
```
Returns a string object of type `string_type` whose content is a copy of the underlying buffer contents.

```
void
str(const string_type& str_arg);
```
Clears the underlying buffer and copies the string object `str_arg` into it. If the opening mode is `in`, initializes the input sequence to point at the first character of the buffer. If the opening mode is `out`, the function initializes the output sequence to point at the first character of the buffer. If the opening mode is `out | app`, it initializes the output sequence to point at the last character of the buffer.

```
int_type
underflow();
```
If the input sequence has a read position available, the function returns the contents of this position. Otherwise it tries to expand the input sequence to match the output sequence and if possible returns the content of the new current position. The function returns `traits::eof()` to indicate failure.

```
streamsize
xsputn(const char_type* s, streamsize n);
```

Writes up to n characters to the output sequence. The characters written are obtained from successive elements of the array whose first element is designated by s. The function returns the number of characters written.

Examples

```
// stdlib/examples/manual/stringbuf.cpp
//
#include<iostream>
#include<sstream>
#include<string>

void main ( )
{
  using namespace std;

  // create a read/write string-stream object on tiny char
  // and attach it to an ostringstream object
  ostringstream out_1(ios_base::in | ios_base::out);

  // tie the istream object to the ostringstream object
  istream in_1(out_1.rdbuf());

  // output to out_1
  out_1 << "Here is the first output";

  // create a string object on tiny char
  string  string_ex("l'heure est grave !");

  // open a read only string-stream object on tiny char
  // and initialize it
  istringstream in_2(string_ex);

  // output in_1 to the standard output
  cout << in_1.str() << endl;

  // output in_2 to the standard output
  cout << in_2.rdbuf() << endl;

  // reposition in_2 at the beginning
  in_2.seekg(0);

  stringbuf::pos_type pos;

  // get the current put position
  // equivalent to
  // out_1.tellp();
  pos = out_1.rdbuf()->pubseekoff(0,ios_base::cur,
                                  ios_base::out);

  // append the content of stringbuffer
  // pointed at by in_2 to the one
  // pointed at by out_1
  out_1 << ' ' << in_2.rdbuf();

  // output in_1 to the standard output
  cout << in_1.str() << endl;
  // position the get sequence
  // equivalent to
  // in_1.seekg(pos);
  in_1.rdbuf()->pubseekpos(pos, ios_base::in);

  // output "l'heure est grave !"
  cout << in_1.rdbuf() << endl << endl;
}
```

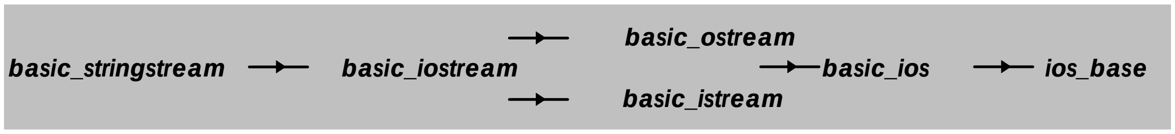
See Also

char_traits(3C++), ***ios_base***(3C++), ***basic_ios***(3C++), ***basic_streambuf***(3C++), ***basic_string***(3C++), ***basic_istringstream***(3C++), ***basic_ostringstream***(3C++), ***basic_stringstream***(3C++)

Working Paper for Draft Proposed International Standard for Information Systems--Programming Language C++, Section 27.7.1

Standards Conformance

ANSI X3J16/ISO WG21 Joint C++ Committee



Synopsis

```
#include <sstream>
template<class charT, class traits = char_traits<charT>,
         class Allocator = allocator<void> >
class basic_stringstream
: public basic_iostream<charT, traits>
```

Description

The template class ***basic_stringstream<charT,traits,Allocator>*** provides functionality to read and write to an array in memory. It supports writing and reading objects of class ***basic_string<charT,traits,Alocator>***. It uses a `basic_stringbuf` object to control the associated storage. It inherits from ***basic_iostream*** and therefore can use all the formatted and unformatted output and input functions.

Interface

```
template<class charT, class traits = char_traits<charT>,
         class Allocator = allocator<void> >
class basic_stringstream
: public basic_iostream<charT, traits> {

 public:

  typedef basic_stringbuf<charT, traits, Allocator>  sb_type;
  typedef basic_ios<charT, traits>                   ios_type;

  typedef basic_string<charT, traits, Allocator>     string_type;

  typedef traits                                traits_type;
  typedef charT                                 char_type;
  typedef typename traits::int_type             int_type;
  typedef typename traits::pos_type             pos_type;
  typedef typename traits::off_type             off_type;

  explicit basic_stringstream(ios_base::openmode which =
                              ios_base::out | ios_base::in);

  explicit basic_stringstream(const string_type& str,
                              ios_base::openmode which =
                              ios_base::out | ios_base::in);

  virtual ~basic_stringstream();

  basic_stringbuf<charT,traits,Allocator> *rdbuf() const;
  string_type str() const;

  void str(const string_type& str);

};
```

Types

`char_type`
The type `char_type` is a synonym for the template parameter `charT`.

`int_type`
The type `int_type` is a synonym of type `traits::in_type`.

`ios_type`
The type `ios_type` is an instantiation of class `basic_ios` on type `charT`.

`off_type`
The type `off_type` is a synonym of type `traits::off_type`.

`pos_type`
The type `pos_type` is a synonym of type `traits::pos_type`.

`sb_type`
The type `sb_type` is an instantiation of class `basic_stringbuf` on type `charT`.

`string_type`
The type `string_type` is an instantiation of class `basic_string` on type `charT`.

`stringstream`
The type `stringstream` is an instantiation of class `basic_stringstream` on type `char`:

```
typedef basic_stringstream<char> stringstream;
```

`traits_type`
The type `traits_type` is a synonym for the template parameter `traits`.

`wstringstream`
The type `wstringstream` is an instantiation of class `basic_stringstream` on type `wchar_t:`

```
typedef basic_stringstream<wchar_t> wstringstream;
```

Constructors

```
explicit basic_stringstream(ios_base::openmode which =
                   ios_base::in | ios_base::out);
```

Constructs an object of class `basic_stringstream`, initializing the base class `basic_iostream` with the associated string buffer. The string buffer is initialized by calling the `basic_stringbuf` constructor `basic_stringbuf<charT,traits,Allocator>(which).`

```
explicit basic_stringstream(const string_type& str,
                   ios_base::openmode which =
                   ios_base::in | ios_base::out);
```

Constructs an object of class `basic_stringstream`, initializing the base class `basic_iostream` with the associated string buffer. The string buffer is initialized by calling the `basic_stringbuf` constructor `basic_stringbuf<charT,traits,Allocator>(str,which)`.

Destructor

`virtual` **`~basic_stringstream`**`();`

Destroys an object of class `basic_stringstream`.

Member Functions

`basic_stringbuf<charT,traits,Allocator>*`
`rdbuf``() const;`

Returns a pointer to the `basic_stringbuf` associated with the stream.

`string_type`
`str``() const;`

Returns a string object of type `string_type` whose contents is a copy of the underlying buffer contents.

`void`
`str``(const string_type& str);`

Clears the string buffer and copies the string object `str` into it. If the opening mode is `in`, initializes the input sequence to point at the first character of the buffer. If the opening mode is `out`, initializes the output sequence to point at the first character of the buffer. If the opening mode is `out | app`, initializes the output sequence to point at the last character of the buffer.

Examples

```
//
// stdlib/examples/manual/stringstream.cpp
//
#include<iostream>
#include<sstream>

void main ( )
{
  using namespace std;

  // create a bi-directional wstringstream object
  wstringstream inout;

  // output characters
  inout << L"Das ist die rede von einem man" << endl;
  inout << L"C'est l'histoire d'un home" << endl;
  inout << L"This is the story of a man" << endl;

  wchar_t p[100];

  // extract the first line
  inout.getline(p,100);

  // output the first line to stdout
  wcout << endl << L"Deutch :" << endl;
  wcout << p;

  // extract the second line
  inout.getline(p,100);
  // output the second line to stdout
  wcout << endl << L"Francais :" << endl;
  wcout << p;

  // extract the third line
  inout.getline(p,100);

  // output the third line to stdout
  wcout << endl << L"English :" << endl;
  wcout << p;

  // output the all content of the
  //wstringstream object to stdout
  wcout << endl << endl << inout.str();
}
```

See Also

char_traits(3C++), ***ios_base***(3C++), ***basic_ios***(3C++), ***basic_stringbuf***(3C++), ***basic_string***(3C++), ***basic_istringstream***(3C++), ***basic_ostringstream***(3c++)

Working Paper for Draft Proposed International Standard for Information Systems--Programming Language C++, Section 27.7.3

Standards Conformance

ANSI X3J16/ISO WG21 Joint C++ Committee

cerr

Pre-defined stream

Synopsis

```
#include <iostream>
extern ostream cerr;
```

Description

`ostream` **`cerr`**`;`

The object `cerr` controls output to an unbuffered stream buffer associated with the object `stderr` declared in `<cstdio>`. By default, the standard C and C++ streams are synchronized, but you may improve performance by using the `ios_base` member function `synch_with_stdio` to desynchronize them.

Formatting

The formatting is done through member functions or manipulators. See `cout` or `basic_ostream` for details.

Examples

```
//
// cerr example
//
#include<iostream>
#include<fstream>

void main ( )
{
  using namespace std;

  // open the file "file_name.txt"
  // for reading
  ifstream in("file_name.txt");

  // output the all file to stdout
  if ( in )
    cout << in.rdbuf();
  else
    // if the ifstream object is in a bad state
    // output an error message to stderr
    cerr << "Error while opening the file" << endl;
}
```

See Also

basic_ostream(3C++), ***iostream***(3C++), ***basic_filebuf***(3C++), ***cout***(3C++), ***cin***(3C++), ***clog***(3C++), ***wcin***(3C++), ***wcout***(3C++), ***wcerr***(3C++), ***wclog***(3C++), ***iomanip***(3C++), ***ios_base***(3C++), ***basic_ios***(3C++)

Working Paper for Draft Proposed International Standard for Information Systems--Programming Language C++, Section 27.3.1

Standards Conformance

ANSI X3J16/ISO WG21 Joint C++ Committee

Summary

A traits class providing types and operations to the ***basic_string*** container and ***iostream*** classes.

Synopsis

```
#include <string>
template<class charT>
struct char_traits
```

Description

The template structure ***char_traits<charT>*** defines the types and functions necessary to implement the ***iostreams*** and ***string*** template classes. It is templatized on `charT`, which represents the character container type. Each specialized version of ***char_traits<charT>*** provides the default definitions corresponding to the specialized character container type.

Users have to provide specialization for ***char_traits*** if they use other character types than `char` and `wchar_t`.

Interface

```
template<class charT>
struct char_traits {

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

  static char_type        to_char_type(const int_type&);
  static int_type         to_int_type(const char_type&);
  static bool             eq(const char_type&,const char_type& );
  static bool             eq_int_type(const int_type&,const int_type&);

  static int_type         eof();
  static int_type         not_eof(const int_type&);

  static void             assign(char_type&,const char_type&);
  static bool             lt(const char_type&,const char_type&);
  static int              compare(const char_type*,const char_type*,size_t);
  static size_t           length(const char_type*);
  static const char_type* find(const char_type*,int n,const char_type&);

  static char_type*       move(char_type*,const char_type*,size_t);
  static char_type*       copy(char_type*,const char_type*, size_t);
  static char_type*       assign(char_type*,size_t,const char_type&);

};
```

Types

char_type

The type `char_type` represents the character container type. It must be convertible to `int_type`.

int_type

The type `int_type` is another character container type which can also hold an end-of-file value. It is used as the return type of some of the iostream class member functions. If `char_type` is either `char` or `wchar_t`, `int_type` is `int` or `wint_t`, respectively.

off_type

The type `off_type` represents offsets to positional information. It is used to represent:

- a signed displacement, measured in characters, from a specified position within a sequence.
- an absolute position within a sequence.

The value `off_type(-1)` can be used as an error indicator. Value of type `off_type` can be converted to type `pos_type,` but no validity of the resulting `pos_type` value is ensured.

If `char_type` is either `char` or `wchar_t`, `off_type` is `streamoff` or `wstreamoff`, respectively.

pos_type

The type `pos_type` describes an object that can store all the information necessary to restore an arbitrary sequence to a previous stream position and conversion state. The conversion `pos_type(off_type(-1))` constructs the invalid `pos_type` value to signal error.

If `char_type` is either `char` or `wchar_t`, `pos_type` is `streampos` or `wstreampos`, respectively.

state_type

The type `state_type` holds the conversion state, and is compatible with the function `locale::codecvt()`.

If `char_type` is either `char` or `wchar_t`, `state_type` is `mbstate_t`.

Types Default-Values

specialization type	on char	on wchar_t
`char_type`	`char`	`wchar_t`
`int_type`	`int`	`wint_t`
`off_type`	`streamoff`	`wstreamoff`
`pos_type`	`streampos`	`wstreampos`
`state_type`	`mbstate_t`	`mbstate_t`

Value Functions

```
void
assign(char_type& c1, const char_type& c2);
```

Assigns one character value to another. The value of `c2` is assigned to `c1`.

```
char_type*
assign(char_type* s,size_t n,const char_type& a);
```

Assigns one character value to `n` elements of a character array. The value of `a` is assigned to `n` elements of `s`.

```
char_type*
copy(char_type* s1, const char_type* s2, size_t n);
```

Copies `n` characters from the object pointed at by `s1` into the object pointed at by `s2`. The ranges of `(s1,s1+n)` and `(s2,s2+n)` may not overlap.

```
int_type
eof();
```

Returns an `int_type` value which represents the end-of-file. It is returned by several functions to indicate end-of-file state, or to indicate an invalid return value.

```
const char_type*
find(const char_type* s, int n, const char_type& a);
```

Looks for the value of `a` in `s`. Only `n` elements of `s` are examined. Returns a pointer to the matched element if one is found. Otherwise returns a pointer to the `n` element in `s`.

```
size_t
length(const char_type* s);
```

Returns the length of a null terminated character string pointed at by `s`.

```
char_type*
move(char_type* s1, const char_type* s2, size_t n);
```

Moves `n` characters from the object pointed at by `s1` into the object pointed at by `s2`. The ranges of `(s1,s1+n)` and `(s2,s2+n)` may overlap.

```
int_type
not_eof(const int_type& c);
```

Returns a value which is not equal to the end-of-file value.

Test Functions

```
int
compare(const char_type* s1,const char_type* s2,size_t n);
```

Compares n values from s1 with n values from s2. Returns 1 if s1 is greater than s2, -1 if s1 is less than s2, or 0 if they are equal.

```
bool
eq(const char_type& c1, const char_type& c2);
```

Returns true if `c1` and `c2` represent the same character.

```
bool
eq_int_type(const int_type& c1, const int_type& c2);
```

Returns true if `c1` and `c2` represents the same character.

```
bool
lt(const char_type& c1,const char_type& c2);
```

Returns true if c1 is less than c2.

Conversion Functions

```
char_type
to_char_type(const int_type& c);
```

Converts a valid character represented by a value of type `int_type` to the corresponding `char_type` value.

```
int_type
to_int_type(const char_type& c);
```

Converts a valid character represented by a value of type `char_type` to the corresponding `int_type` value.

See Also

iosfwd(3C++), ***fpos***(3C++)

Working Paper for Draft Proposed International Standard for Information Systems--Programming Language C++, Section 21.1.4, 21.1.5, 27.1.2.

Standards Conformance

ANSI X3J16/ISO WG21 Joint C++ Committee

cin

Pre-defined stream

Synopsis

```
#include <iostream>
extern istream cin;
```

Description

`istream` **`cin`**`;`

The object `cin` controls input from a stream buffer associated with the object `stdin` declared in `<cstdio>`. By default, the standard C and C++ streams are synchronized, but you can improve performance by using the `ios_base` member function `synch_with_stdio` to desynchronize them.

After the object `cin` is initialized, `cin.tie()` returns `&cout`, which implies that `cin` and `cout` are synchronized.

Examples

```
//
// cin example one
//
#include <iostream>

void main ( )
{
  using namespace std;

  int i;
  float f;
  char c;

  //read an integer, a float and a character from stdin
  cin >> i >> f >> c;

  // output i, f and c to stdout
  cout << i << endl << f << endl << c << endl;
}

//
// cin example two
//
#include <iostream>

void main ( )
{
  using namespace std;

  char p[50];

  // remove all the white spaces
  cin >> ws;

  // read characters from stdin until a newline
  // or 49 characters have been read
  cin.getline(p,50);
  // output the result to stdout
  cout << p;
}
```

When inputting `"   Grendel the monster" (newline)` in the previous test, the output will be `"Grendel the monster"`. The manipulator `ws` removes spaces.

See Also

basic_istream(3C++), ***iostream***(3C++), ***basic_filebuf***(3C++), ***cout***(3C++), ***cerr***(3C++), ***clog***(3C++), ***wcin***(3C++), ***wcout***(3C++), ***wcerr***(3C++), ***wclog***(3C++), ***ios_base***(3C++), ***basic_ios***(3C++)

Working Paper for Draft Proposed International Standard for Information Systems--Programming Language C++, Section 27.3.1

Standards Conformance

ANSI X3J16/ISO WG21 Joint C++ Committee

clog

Pre-defined stream

Synopsis

```
#include <iostream>
extern ostream clog;
```

Description

```
ostream clog;
```

The object `clog` controls output to a stream buffer associated with the object stderr declared in `<cstdio>`. The difference between `clog` and `cerr` is that `clog` is buffered but `cerr` is not. Therefore, commands like `clog << "ERROR !!";` and `fprintf(stderr,"ERROR !!");` are not synchronized.

Formatting

The formatting is done through member functions or manipulators. See ***cout*** or ***basic_ostream*** for details.

Examples

```
//
// clog example
//
#include<iostream>
#include<fstream>

void main ( )
{
  using namespace std;

  // open the file "file_name.txt"
  // for reading
  ifstream in("file_name.txt");

  // output the all file to stdout
  if ( in )
    cout << in.rdbuf();
  else
    // if the ifstream object is in a bad state
    // output an error message to stderr
    clog << "Error while opening the file" << endl;
}
```

Warnings

clog can be used to redirect some of the errors to another recipient. For example, you might want to redirect them to a file named `my_err`:

```
ofstream out("my_err");

if ( out )
  clog.rdbuf(out.rdbuf());

else
```

```
    cerr << "Error while opening the file" << endl;
```

Then when you are doing something like `clog << "error number x";` the error message is output to the file `my_err`. Obviously, you can use the same scheme to redirect ***clog*** to other devices.

See Also

basic_ostream(3C++), ***iostream***(3C++), ***basic_filebuf***(3C++), ***cout***(3C++), ***cin***(3C++), ***cerr***(3C++), ***wcin***(3C++), ***wcout***(3C++), ***wcerr***(3C++), ***wclog***(3C++), ***iomanip***(3C++), ***ios_base***(3C++), ***basic_ios***(3C++),

Working Paper for Draft Proposed International Standard for Information Systems--Programming Language C++, Section 27.3.1

Standards Conformance

ANSI X3J16/ISO WG21 Joint C++ Committee

code_cvt

codecvt. . . . . .→ *codecvt_base*

. . .→ *locale::facet*

Summary

Code conversion facet.

Synopsis

```
#include <locale>
class codecvt_base;

template <class internT, class externT, class stateT>
class codecvt;
```

Description

The ***codecvt<internT,externT,stateT>*** template provides code conversion facilities. Default implementations of ***codecvt<char,wchar_t,mbstate_t>*** and ***codecvt<wchar_t,char,mbstate_t>*** use `ctype<wchar_t>::widen` and `ctype<wchar_t>::narrow` respectively. The default implementation of ***codecvt<wchar_t,wchar_t,mbstate_t>*** simply uses `memcpy` (no particular conversion applied).

Interface

```
class codecvt_base {
public:
  enum result { ok, partial, error, noconv };
};

template <class internT, class externT, class stateT>
class codecvt : public locale::facet, public codecvt_base {
public:
  typedef internT  intern_type;
  typedef externT  extern_type;
  typedef stateT   state_type;

  explicit codecvt(size_t = 0)
  result out(stateT&, const internT*,
             const internT*, const internT*&,
             externT*, externT*, externT*&) const;
  result in(stateT&, const externT*,
             const externT*, const externT*&,
             internT*, internT*, internT*&) const;

  bool always_noconv() const throw();
  int length(const stateT&, const internT*, const internT*,
              size_t) const;

  int max_length() const throw();
  int encoding() const throw();
  static locale::id id;

protected:
    ~codecvt();  // virtual
    virtual result do_out(stateT&,
                          const internT*,
                          const internT*,
                          const internT*&,
                          externT*, externT*,
                          externT*&) const;
    virtual result do_in(stateT&,
                         const externT*,
                         const externT*,
                         const externT*&,
                         internT*, internT*,
                         internT*&) const;

    virtual bool do_always_noconv() const throw();
    virtual int do_length(const stateT&, const internT*,
                          const internT*,
                          size_t) const;

    virtual int do_max_length() const throw();
    virtual int do_encoding() const throw();
  };
```

Types

`intern_type`
Type of character to convert from.

`extern_type`
Type of character to convert to.

`state_type`
Type to keep track of state and determine the direction of the conversion.

Constructors and Destructors

`explicit` **`codecvt`**`(size_t refs = 0)`
Construct a ***codecvt*** facet. If the `refs` argument is 0 then destruction of the object is delegated to the locale, or locales, containing it. This allows the user to ignore lifetime management issues. On the other had, if `refs` is 1 then the object must be explicitly deleted; the locale will not do so. In this case, the object can be maintained across the lifetime of multiple locales.

`~codecvt``();  // virtual and protected`
Destroy the facet

Facet ID

`static locale::id` **`id`**`;`
Unique identifier for this type of facet.

Public Member Functions

The public members of the ***codecvt*** facet provide an interface to protected members. Each public member `xxx` has a corresponding virtual protected member `do_xxx`. All work is delegated to these protected members. For instance, the long version of the public `length` function simply calls its protected cousin `do_length`.

```
bool
always_noconv() const
  throw();

int
encoding() const
  throw();

result
in(stateT& state, const externT* from,
   const externT* from_end, const externT*& from_next,
   internT* to, internT* to_limit, internT*& to_next) const;

int
length(const stateT&, const internT* from,
       const internT* end,
       size_t max) const;

int
max_length() const
  throw();

result
out(stateT& state, const internT* from,
    const internT* from_end, const internT*& from_next,
    externT* to, externT* to_limit, externT*& to_next) const;
```

Each of these public member functions `xxx` simply calls the corresponding protected `do_xxx` function.

Protected Member Functions

```
virtual bool
do_always_noconv() const
throw();
```

Returns `true` if no conversion is required. This is the case if `do_in` and `do_out` return `noconv` for all valid arguments. The instantiation `codecvt<char,char,mbstate_t>` returns `true`, while all other default instantiations return `false`.

```
virtual int
do_encoding() const
  throw();
```

Returns one of the following

- `-1` if the encoding on the external character sequence is dependent on state.
- A constant number representing the number of external characters in a fixed width encoding.
- `0` if the encoding is uses a variable width.

```
virtual result
do_in(stateT& state,
      const externT* from,
      const externT* from_end,
      const externT*& from_next,
      internT* to, internT* to_limit,
      internT*& to_next) const;

virtual result
do_out(stateT& state,
       const internT* from,
       const internT* from_end,
       const internT*& from_next,
       externT* to, externT* to_limit,
       externT*& to_next) const;
```

Both functions take characters in the range of `[from,from_end)`, apply an appropriate conversion, and place the resulting characters in the buffer starting at `to`. Each function converts at most `from_end-from internT` characters, and stores no more than `to_limit-to externT` characters. Both `do_out` and `do_in` will stop if they find a character they cannot convert. In any case, `from_next` and `to_next` are always left pointing to the next character beyond the last one successfully converted.

`do_out` and `do_in` must be called under the following pre-conditions:

- `from <= from_end`
- `to <= to_end`
- `state` either initialized for the beginning of a sequence or equal to the result of the previous conversion on the sequence.

In the case where no conversion is required, `from_next` will be set to `from` and `to_next` set to `to`.

`do_out` and `do_in` return one the following:

Return Value	Meaning
`ok`	completed the conversion
`partial`	not all source characters converted
`error`	encountered a `from_type` character it could not convert
`noconv`	no conversion was needed

If either function returns `partial` and `(from == from_end)` then one of two conditions prevail:

- The destination sequence has not accepted all the converted characters, or
- Additional `internT` characters are needed before another `externT` character can be assembled.

```
virtual int
do_length(const stateT&, const internT* from,
          const internT* end,
          size_t max) const;
```

Returns the largest `number < max` of `internT` characters available in the range `[from,end)`.

`do_length` must be called under the following pre-conditions:

- `from <= from_end`
- `state` either initialized for the beginning of a sequence or equal to the result of the previous conversion on the sequence.

```
virtual int
do_max_length() const throw();
```

Returns the maximum value that `do_length` can return for any valid set of arguments.

```
virtual result
do_out(stateT& state,
       const internT* from,
       const internT* from_end,
       const internT*& from_next,
       externT* to, externT* to_limit,
       externT*& to_next) const;
```

See `do_in` above.

Example

```
//
// codecvt.cpp
//
#include <sstream>
#include "codecvte.h"

int main ()
{
  using namespace std;

  mbstate_t state;

  // A string of ISO characters and buffers to hold
  // conversions
  string ins("\xfc \xcc \xcd \x61 \xe1 \xd9 \xc6 \xf5");
  string ins2(ins.size(),'.');
  string outs(ins.size(),'.');

  // Print initial contents of buffers
  cout << "Before:\n" << ins << endl;
  cout << ins2 << endl;
  cout << outs << endl << endl;

  // Initialize buffers
  string::iterator in_it = ins.begin();
  string::iterator out_it = outs.begin();

  // Create a user defined codecvt fact
  // This facet converts from ISO Latin
  // Alphabet No. 1 (ISO 8859-1) to
  // U.S. ASCII code page 437
  // This facet replaces the default for
  // codecvt<char,char,mbstate_t>
  locale loc(locale(),new ex_codecvt);

  // Now get the facet from the locale
  const codecvt<char,char,mbstate_t>& cdcvt =
#ifndef _RWSTD_NO_TEMPLATE_ON_RETURN_TYPE
    use_facet<codecvt<char,char,mbstate_t> >(loc);
#else
    use_facet(loc,(codecvt<char,char,mbstate_t>*)0);
#endif

  // convert the buffer
  cdcvt.in(state,ins.begin(),ins.end(),in_it,
           outs.begin(),outs.end(),out_it);

  cout << "After in:\n" << ins << endl;
  cout << ins2 << endl;
  cout << outs << endl << endl;

  // Lastly, convert back to the original codeset
  in_it = ins.begin();
  out_it = outs.begin();
  cdcvt.out(state, outs.begin(),outs.end(),out_it,
            ins2.begin(),ins2.end(),in_it);

  cout << "After out:\n" << ins << endl;
  cout << ins2 << endl;
  cout << outs << endl;

  return 0;
}
```

See Also ***locale, facets, codecvt_byname***

codecvt_byname

$$\begin{tikzcd}
 & & \textit{codecvt\_base} \\
\textit{codecvt\_byname} \arrow[r] & \textit{codecvt} \ldots \arrow[ur, dashed] \arrow[dr, dashed] & \\
 & & \textit{locale::facet}
\end{tikzcd}$$

Summary

A facet that provides code set conversion classification facilities based on the named locales.

Synopsis

```
#include <locale>
template <class charT> class codecvt_byname;
```

Description

The ***codecvt_byname*** template provides the same functionality as the ***codecvt*** template, but specific to a particular named locale. For a description of the member functions of ***codecvt_byname,*** see the reference for ***codecvt***. Only the constructor is described here.

Interface

```
  template <class fromT, class toT, class stateT>
  class codecvt_byname : public codecvt<fromT, toT, stateT> {
  public:
    explicit codecvt_byname(const char*, size_t refs = 0);
  protected:
    ~codecvt_byname();  // virtual
    virtual result do_out(stateT&,
                          const internT*,
                          const internT*,
                          const internT*&,
                          externT*, externT*,
                          externT*&) const;
    virtual result do_in(stateT&,
                         const externT*,
                         const externT*,
                         const externT*&,
                         internT*, internT*,
                         internT*&) const;

    virtual bool do_always_noconv() const throw();
    virtual int do_length(const stateT&, const internT*,
                          const internT*,
    virtual int do_max_length() const throw();
    virtual int do_encoding() const throw();
  };
```

Constructor

`explicit` **`codecvt_byname`**`(const char* name, size_t refs = 0);`

Construct a ***codecvt_byname*** facet. The facet will provide codeset conversion relative to the named locale specified by the `name` argument. If the `refs` argument is 0, destruction of the object is delegated to the locale, or locales, containing it. This allows the user to ignore lifetime management issues. On the other had, if `refs` is 1, the object must be explicitly deleted: the locale will not do so. In this case, the object can be maintained across the lifetime of multiple locales.

See Also ***locale, facets, codecvt***

collate, collate_byname

collate_byname → ***collate*** → ***locale::facet***

Summary

String collation, comparison, and hashing facet.

Synopsis

```
#include <locale>
template <class charT> class collate;
template <class charT> class collate_byname;
```

Description

The ***collate*** and ***collate_byname*** facets provides string collation, comparison, and hashing facilities. ***collate*** provides these facilities for the "C" locale, while ***collate_byname*** provides the same thing for named locales.

Interface

```
template <class charT>
class collate : public locale::facet {
public:
  typedef charT               char_type;
  typedef basic_string<charT> string_type;
  explicit collate(size_t refs = 0);
  int compare(const charT*, const charT*,
              const charT*, const charT*) const;
  string_type transform(const charT*, const charT*) const;
  long hash(const charT*, const charT*) const;
  static locale::id id;
protected:
  ~collate();  // virtual
  virtual int do_compare(const charT*, const charT*,
                         const charT*, const charT*) const;
  virtual string_type do_transform(const charT*, const charT*)
const;
  virtual long do_hash (const charT*, const charT*) const;
};

template <class charT>
class collate_byname : public collate<charT> {
public:
  explicit collate_byname(const char*, size_t = 0);
protected:
  ~collate_byname();  // virtual
  virtual int do_compare(const charT*, const charT*,
                         const charT*, const charT*) const;
  virtual string_type do_transform(const charT*, const charT*)
const;
  virtual long do_hash(const charT*, const charT*) const;
};
```

Types

`char_type`
Type of character the facet is instantiated on.

string_type
Type of character string returned by member functions.

Constructors and Destructors

```
explicit collate(size_t refs = 0)
```
Construct a ***collate*** facet. If the `refs` argument is 0, destruction of the object is delegated to the locale, or locales, containing it. This allows the user to ignore lifetime management issues. On the other had, if `refs` is 1, the object must be explicitly deleted: the locale will not do so. In this case, the object can be maintained across the lifetime of multiple locales.

```
explicit collate_byname(const char* name, size_t refs = 0);
```
Construct a ***collate_byname*** facet. Use the named locale specified by the name argument. The `refs` argument serves the same purpose as it does for the ***collate*** constructor.

```
~collate();  // virtual and protected
~collate_byname();  // virtual and protected
```
Destroy the facet

Facet ID

```
static locale::id id;
```
Unique identifier for this type of facet.

Public Member Functions

The public members of the ***collate*** facet provide an interface to protected members. Each public member `xxx` has a corresponding virtual protected member `do_xxx`. All work is delegated to these protected members. For instance, the long version of the public `grouping` function simply calls its protected cousin `do_grouping`.

```
int
compare(const charT* low1, const charT* high1,
        const charT* low2, const charT* high2) const;
```

```
long
hash(const charT* low, const charT* high) const;
```

```
string_type
transform(const charT* low, const charT* high) const;
```
Each of these public member functions `xxx` simply call the corresponding protected `do_xxx` function.

Protected Member Functions

```
virtual int
do_compare(const charT* low1, const charT* high1,
           const charT* low2, const charT* high2) const;
```
Returns `1` if the character string represented by the range `[low1,high1)` is greater than the character string represented by the range `[low2,high2)`, `-1` if first string is less than the second, or `0` if the two are equal. ***collate*** uses a lexicographical comparison.

`virtual long`
`do_hash``( const charT* low, const charT* high)`

Generate a has value from a string defined by the range of characters `[low,high)`. Given two strings that compare equal (i.e. `do_compare` returns `0`), `do_hash` returns an integer value that is the same for both strings. For differing strings the probability that the return value will be equal is approximately `1.0/numeric_limits<unsigned long>::max().`

`virtual string_type`
`do_transform``(const charT* low, const charT* high) const;`

Returns a string that will yield the same result in a lexicographical comparison with another string returned from transform as does the `do_compare` function applied to the original strings. In other words, the result of applying a lexicographical comparison to two strings returned from `transform` will be the same as applying `do_compare` to the original strings passed to transform.

Example

```
//
// collate.cpp
//
#include <iostream>

int main ()
{
  using namespace std;
  locale loc;
  string s1("blue");
  string s2("blues");
  // Get a reference to the collate<char> facet
  const collate<char>& co =
#ifndef _RWSTD_NO_TEMPLATE_ON_RETURN_TYPE
      use_facet<collate<char> >(loc);
#else
      use_facet(loc,(collate<char>*)0);
#endif
  // Compare two strings
  cout << co.compare(s1.begin(),s1.end(),
                     s2.begin(),s2.end()-1) << endl;
  cout << co.compare(s1.begin(),s1.end(),
                     s2.begin(),s2.end()) << endl;
  // Retrieve hash values for two strings
  cout << co.hash(s1.begin(),s1.end()) << endl;
  cout << co.hash(s2.begin(),s2.end()) << endl;
  return 0;
}
```

See Also

locale, facets, ctype

cout

Pre-defined stream

Synopsis

```
#include <iostream>
extern ostream cout;
```

Description

`ostream` **`cout`**`;`

The object `cout` controls output to a stream buffer associated with the object `stdout` declared in `<cstdio>`. By default the standard C and C++ streams are synchronized, but performance improvement can be achieved by using the `ios_base` member function `synch_with_stdio` to desynchronize them.

After the object `cin` is initialized, `cin.tie()` returns `&cout`, which implies that `cin` and `cout` are synchronized.

Formatting

The formatting is done through member functions or manipulators.

Manipulators	Member functions
`showpos`	`setf(ios_base::showpos)`
`noshowpos`	`unsetf(ios_base::showpos)`
`showbase`	`setf(ios_base::showbase)`
`noshowbase`	`unsetf(ios_base::showbase)`
`uppercase`	`setf(ios_base::uppercase)`
`nouppercase`	`unsetf(ios_base::uppercase)`
`showpoint`	`setf(ios_base::showpoint)`
`noshowpoint`	`unsetf(ios_base::showpoint)`
`boolalpha`	`setf(ios_base::boolalpha)`
`noboolalpha`	`unsetf(ios_base::boolalpha)`
`unitbuf`	`setf(ios_base::unitbuf)`
`nounitbuf`	`unsetf(ios_base::unitbuf)`
`internal`	`setf(ios_base::internal, ios_base::adjustfield)`
`left`	`setf(ios_base::left, ios_base::adjustfield)`
`right`	`setf(ios_base::right, ios_base::adjustfield)`
`dec`	`setf(ios_base::dec, ios_base::basefield)`
`hex`	`setf(ios_base::hex, ios_base::basefield)`
`oct`	`setf(ios_base::oct, ios_base::basefield)`

`fixed`	`setf(ios_base::fixed, ios_base::floatfield)`
`scientific`	`setf(ios_base::scientific, ios_base::floatfield)`
`resetiosflags (ios_base::fmtflags flag)`	`setf(0,flag)`
`setiosflags (ios_base::fmtflags flag)`	`setf(flag)`
`setbase(int base)`	`see above`
`setfill(char_type c)`	`fill(c)`
`setprecision(int n)`	`precision(n)`
`setw(int n)`	`width(n)`
`endl`	
`ends`	
`flush`	`flush( )`

Description

`showpos`	Generates a + sign in non-negative generated numeric output.
`showbase`	Generates a prefix indicating the numeric base of generated integer output.
`uppercase`	Replaces certain lowercase letters with their uppercase equivalents in generated output.
`showpoint`	Generates a decimal-point character unconditionally in generated floating-point output.
`boolalpha`	Insert and extract bool type in alphabetic format.
`unitbuf`	Flushes output after each output operation.
`internal`	Adds fill characters at a designated internal point in certain generated output, or identical to right if no such point is designated.
`left`	Adds fill characters on the right (final positions) of certain generated output.
`right`	Adds fill characters on the left (initial positions) of certain generated output.
`dec`	Converts integer input or generates integer output in decimal base.
`hex`	Converts integer input or generates integer output in hexadecimal base.

`oct`	Converts integer input or generates integer output in octal base.
`fixed`	Generates floating-point output in fixed-point notation.
`scientific`	Generates floating-point output in scientific notation.
`resetiosflagss (ios_base::fmtflags flag)`	Resets the `fmtflags` field `flag.`
`setiosflags (ios_base::fmtflags flag)`	Sets up the flag `flag.`
`setbase(int base)`	Converts integer input or generates integer output in base `base`. The parameter base can be 8, 10 or 16.
`setfill(char_type c)`	Sets the character used to pad (fill) an output conversion to the specified field width.
`setprecision(int n)`	Set the precision (number of digits after the decimal point) to generate on certain output conversions.
`setw(int n)`	Sets the field with (number of characters) to generate on certain output conversions
`endl`	Inserts a newline character into the output sequence and flush the output buffer.
`ends`	Inserts a null character into the output sequence.
`flush`	Flush the output buffer.

Default Values

`precision()`	`6`
`width()`	`0`
`fill()`	`the space character`
`flags()`	`skipws | dec`
`getloc()`	`locale::locale()`

Examples

```
//
// cout example one
//
#include<iostream>
#include<iomanip>

void main ( )
{
  using namespace std;

  int i;
  float f;
  // read an integer and a float from stdin
  cin >> i >> f;

  // output the integer and goes at the line
  cout << i << endl;

  // output the float and goes at the line
  cout << f << endl;

  // output i in hexa
  cout << hex << i << endl;

  // output i in octal and then in decimal
  cout << oct << i << dec << i << endl;

  // output i preceded by its sign
  cout << showpos << i << endl;

  // output i in hexa
  cout << setbase(16) << i << endl;

  // output i in dec and pad to the left with character
  // @ until a width of 20
  // if you input 45 it outputs 45@@@@@@@@@@@@@@@@@@
  cout << setfill('@') << setw(20) << left << dec << i;
  cout << endl;

  // output the same result as the code just above
  // but uses member functions rather than manipulators
  cout.fill('@');
  cout.width(20);
  cout.setf(ios_base::left, ios_base::adjustfield);
  cout.setf(ios_base::dec, ios_base::basefield);
  cout << i << endl;

  // outputs f in scientific notation with
  // a precision of 10 digits
  cout << scientific << setprecision(10) << f << endl;

  // change the precision to 6 digits
  // equivalents to cout << setprecision(6);
  cout.precision(6);

  // output f and goes back to fixed notation
  cout << f << fixed << endl;

}

//
// cout example two
//
#include <iostream>

void main ( )
{
  using namespace std;
  char p[50];

  cin.getline(p,50);

  cout << p;
}

//
// cout example three
//
#include <iostream>
#include <fstream>

void main ( )
{
  using namespace std;

  // open the file "file_name.txt"
  // for reading
  ifstream in("file_name.txt");

  // output the all file to stdout
  if ( in )
    cout << in.rdbuf();
  else
    {
      cout << "Error while opening the file";
      cout << endl;
    }
}
```

Warnings

Keep in mind that the manipulator `endl` flushes the stream buffer. Therefore it is recommended to use '\n' if your only intent is to go at the line. It will greatly improve performance when C and C++ streams are not synchronized.

See Also

basic_ostream(3C++), ***iostream***(3C++), ***basic_filebuf***(3C++), ***cin***(3C++), ***cerr***(3C++), ***clog***(3C++), ***wcin***(3C++), ***wcout***(3C++), ***wcerr***(3C++), ***wclog***(3C++), ***iomanip***(3C++)

Working Paper for Draft Proposed International Standard for Information Systems--Programming Language C++, Section 27.3.1

Standards Conformance

ANSI X3J16/ISO WG21 Joint C++ Committee

$$\begin{tikzcd}
& \textit{ctype\_base} \\
\textit{ctype}\ldots \arrow[ur] \arrow[dr] & \\
& \textit{locale::facet}
\end{tikzcd}$$

Summary

A facet that provides character classification facilities.

Synopsis

```
#include <locale>
class ctype_base;
template <class charT> class ctype;
```

Specialization

```
class ctype<char>;
```

Description

ctype<charT> is the character classification facet. This facet provides facilities for classifying characters and performing simple conversions. ***ctype<charT>*** provides conversions for upper to lower and lower to upper case. The facet also provides conversions between `charT`, and `char`. ***ctype<charT>*** relies on `ctype_base` for a set of masks that identify the various classes of characters. These classes are:

space

print

cntrl

upper

lower

alpha

digit

punct

xdigit

alnum

graph

The masks are passed to member functions of ***ctype*** in order to obtain verify the classifications of a character or range of characters.

Interface

```
class ctype_base {
public:
  enum mask {
    space, print, cntrl, upper, lower,
    alpha, digit, punct, xdigit,
    alnum=alpha|digit, graph=alnum|punct
  };
};

template <class charT>
class ctype : public locale::facet, public ctype_base {
  public:
    typedef charT char_type;
    explicit ctype(size_t);
    bool         is(mask, charT) const;
    const charT* is(const charT*,
                    const charT*, mask*) const;
    const charT* scan_is(mask,
                         const charT*,
                         const charT*) const;
    const charT* scan_not(mask,
                          const charT*,
                          const charT*) const;
    charT        toupper(charT) const;
    const charT* toupper(charT*, const charT*) const;
    charT        tolower(charT) const;
    const charT* tolower(charT*, const charT*) const;
    charT        widen(char) const;
    const char*  widen(const char*,
                     const char*, charT*) const;
    char         narrow(charT, char) const;
    const charT* narrow(const charT*, const charT*,
                       char, char*) const;
    static locale::id id;

  protected:
     ~ctype();  // virtual
    virtual bool         do_is(mask, charT) const;
    virtual const charT* do_is(const charT*,
                               const charT*,
                               mask*) const;
    virtual const charT* do_scan_is(mask,
                                    const charT*,
                                    const charT*) const;
    virtual const charT* do_scan_not(mask,
                                     const charT*,
                                     const charT*) const;
    virtual charT        do_toupper(charT) const;
    virtual const charT* do_toupper(charT*,
                                    const charT*) const;
    virtual charT        do_tolower(charT) const;
    virtual const charT* do_tolower(charT*,
                                    const charT*) const;
    virtual charT        do_widen(char) const;
    virtual const char*  do_widen(const char*,
                                  const char*,
                                  charT*) const;
    virtual char         do_narrow(charT, char) const;
    virtual const charT* do_narrow(const charT*,
                                   const charT*,
                                   char, char*) const;
};
```

Type

char_type
Type of character the facet is instantiated on.

Constructor and Destructor

`explicit` **`ctype`**`(size_t refs = 0)`
Construct a ***ctype*** facet. If the `refs` argument is 0 then destruction of the object is delegated to the locale, or locales, containing it. This allows the user to ignore lifetime management issues. On the other had, if `refs` is 1 then the object must be explicitly deleted; the locale will not do so. In this case, the object can be maintained across the lifetime of multiple locales.

`~ctype``();  // virtual and protected`
Destroy the facet.

Public Member Functions

The public members of the ***ctype*** facet provide an interface to protected members. Each public member `xxx` has a corresponding virtual protected member `do_xxx`. All work is delegated to these protected members. For instance, the public `widen` function simply calls its protected cousin `do_widen`.

```
bool
is(mask m, charT c) const;
const charT*

is(const charT* low,
   const charT* high, mask* vec) const;
```
Returns `do_is(m,c)` or `do_is(low,high,vec).`

```
char
narrow(charT c, char dfault) const;
const charT*
narrow(const charT* low, const charT*, char dfault,
        char* to) const;
```
Returns `do_narrow(c,dfault)` or `do_narrow(low,high,dfault,to).`

```
const charT*
scan_is(mask m, const charT*, const charT* high) const;
```
Returns `do_scan_is(m,low,high).`

```
const charT*
scan_not(mask m, const charT* low, const charT* high) const;
```
Returns `do_scan_not(m,low,high).`

```
charT
tolower(charT c) const;
const charT*
tolower(charT* low, const charT* high) const;
```
Returns `do_tolower(c)` or `do_tolower(low,high).`

```
charT
toupper(charT) const;
const charT*
toupper(charT* low, const charT* high) const;
```

Returns `do_toupper(c)` or `do_toupper(low,high).`

```
charT
widen(char c) const;
const char*
widen(const char* low, const char* high, charT* to) const;
```

Returns `do_widen(c)` or `do_widen(low,high,to).`

Facet ID

```
static locale::id id;
```

Unique identifier for this type of facet.

Protected Member Functions

```
virtual bool
do_is(mask m, charT c) const;
```

Returns true if `c` matches the classification indicated by the mask `m`, where `m` is one of the values available from *ctype_base*. For instance, the following call returns true since `'a'` is an alphabetic character:

```
ctype<char>().is(ctype_base::alpha,'a');
```

See ***ctype_base*** for a description of the masks.

```
virtual const charT*
do_is(const charT* low, const charT* high,
      mask* vec) const;
```

Fills `vec` with every mask from *ctype_base* that applies to the range of characters indicated by `[low,high)`. See *ctype_base* for a description of the masks. For instance, after the following call `v` would contain `{alpha, lower, print,alnum ,graph`}:

```
char a[] = "abcde";
ctype_base::mask v[12];
ctype<char>().is(a,a+5,v);
```

This function returns `high`.

```
virtual char
do_narrow(charT, char dfault) const;
```

Returns the appropriate `char` representation for `c`, if such exists. Otherwise `do_narrow` returns dfault.

```
virtual const charT*
do_narrow(const charT* low, const charT* high,
          char dfault, char* dest) const;
```

Converts each character in the range `[low,high)` to its `char` representation, if such exists. If a `char` representation is not available then the character will be converted to `dfault`. Returns `high`.

virtual const charT*
do_scan_is(mask m, const charT* low, const charT* high) const;
Finds the first character in the range `[low,high)` that matches the classification indicated by the mask `m`.

virtual const charT*
do_scan_not(mask m, const charT* low, const charT* high) const;
Finds the first character in the range `[low,high)` that does not match the classification indicated by the mask `m`.

virtual charT
do_tolower(charT) const;
Returns the lower case representation of `c,` if such exists, otherwise returns `c`;

virtual const charT*
do_tolower(charT* low, const charT* high) const;
Converts each character in the range `[low,high)` to its lower case representation, if such exists. If a lower case representation does not exist then the character is not changed. Returns `high`.

virtual charT
do_toupper(charT c) const;
Returns the upper case representation of `c,` if such exists, otherwise returns `c`;

virtual const charT*
do_toupper(charT* low, const charT* high) const;
Converts each character in the range `[low,high)` to its upper case representation, if such exists. If an upper case representation does not exist then the character is not changed. Returns `high`.

virtual charT
do_widen(char c) const;
Returns the appropriate `charT` representation for `c`.

virtual const char*
do_widen(const char* low, const char* high,charT* dest) const;
Converts each character in the range `[low,high)` to its `charT` representation. Returns `high`.

Example

```
//
// ctype.cpp
//

#include <iostream>

int main ()
{
  using namespace std;

  locale loc;
  string s1("blues Power");

  // Get a reference to the ctype<char> facet
  const ctype<char>& ct =
#ifndef _RWSTD_NO_TEMPLATE_ON_RETURN_TYPE
      use_facet<ctype<char> >(loc);
#else
      use_facet(loc,(ctype<char>*)0);
#endif

  // Check the classification of the 'a' character
  cout << ct.is(ctype_base::alpha,'a') << endl;
  cout << ct.is(ctype_base::punct,'a') << endl;

  // Scan for the first upper case character
  cout << (char)*(ct.scan_is(ctype_base::upper,
                        s1.begin(),s1.end())) << endl;

  // Convert characters to upper case
  ct.toupper(s1.begin(),s1.end());
  cout << s1 << endl;

  return 0;
}
```

See Also ***locale, facets, collate, ctype<char>, ctype_byname***

ctype<char>

$$\begin{tikzcd} ctype<char> \arrow[r] & ctype\_base \\ ctype<char> \arrow[r] & locale::facet \end{tikzcd}$$

Summary

A specialization of the ***ctype*** facet.

Synopsis

```
#include <locale>
class ctype<char>;
```

Description

This specialization of the ***ctype<charT>*** template provides inline versions of ***ctype***'s member functions. The facet provides the same public interface, and uses the same set of masks, as the *ctype* template.

Interface

```
template <>
class ctype<char> : public locale::facet, public ctype_base {
  public:
    typedef char char_type;
    explicit ctype(const mask* = 0, bool = false,
                   size_t = 0);
    bool         is(mask, char) const;
    const charT* is(const char*,
                    const char*, mask*) const;
    const charT* scan_is(mask,
                         const char*,
                         const char*) const;
    const charT* scan_not(mask,
                          const char*,
                          const char*) const;
    charT        toupper(char) const;
    const charT* toupper(char*, const charT*) const;
    charT        tolower(char) const;
    const charT* tolower(char*, const char*) const;
    charT        widen(char) const;
    const char*  widen(const char*,
                     const char*, char*) const;
    char         narrow(char, char) const;
    const charT* narrow(const char*, const char*,
                        char, char*) const;
    static locale::id id;
    static const size_t table_size = 256;

  protected:
    const mask* table() const throw();
    static const mask* classic_table() throw();

~ctype();  // virtual
    virtual charT        do_toupper(charT) const;
    virtual const charT* do_toupper(charT*,
                                    const charT*) const;
    virtual charT        do_tolower(charT) const;
    virtual const charT* do_tolower(charT*,
                                             const charT*) const;
  };
```

Type

`char_type`
Type of character the facet is instantiated on.

Constructors and Destructors

```
explicit ctype(const mask* tbl = 0, bool del = false,
               size_t refs = 0)
```

Construct a ***ctype*** facet. The three parameters set up the following conditions:

- The `tbl` argument must be either 0 or an array of at least `table_size` elements. If `tbl` is non zero then the supplied table will be used for character classification.
- If `tbl` is non zero, and `del` is true then the `tbl` array will be deleted by the destructor, so the calling program need not concern itself with the lifetime of the facet.
- If the `refs` argument is 0 then destruction of the object itself is delegated to the locale, or locales, containing it. This allows the user to ignore lifetime management issues. On the other had, if `refs` is 1 then the object must be explicitly deleted; the locale will not do so. In this case, the object can be maintained across the lifetime of multiple locales.

```
~ctype();  // virtual and protected
```

Destroy the facet. If the constructor was called with a non-zero `tbl` argument and a true `del` argument, then the array supplied by the `tbl` argument will be deleted

Public Member Functions

The public members of the ***ctype<char>*** facet specialization do not all serve the same purpose as the functions in the template. In many cases these functions implement functionality, rather than just forwarding a call to a protected implementation function.

```
static const mask*
classic_table() throw();
```

Returns a pointer to a table_size character array that represents the classifications of characters in the "C" locale.

```
bool
is(mask m, charT c) const;
```

Determines if the character `c` has the classification indicated by the mask `m`. Returns `table()[(unsigned char)c]` & `m.`

```
const charT*
is(const charT* low,
   const charT* high, mask* vec) const;
```

Fills `vec` with every mask from ***ctype_base*** that applies to the range of characters indicated by `[low,high)`. See ***ctype_base*** for a description of the masks. For instance, after the following call `v` would contain `{alpha, lower, print,,alnum ,graph`}:

```
char a[] = “abcde”;
ctype_base::mask v[12];
ctype<char>().do_is(a,a+5,v);
```

This function returns `high`.

```
char
narrow(charT c, char dfault) const;
```

Returns `c`.

```
const charT*
narrow(const charT* low, const charT*, char dfault,
       char* to) const;
```

Performs `::memcpy(to,low,high-low)`. Returns `high.`

```
const charT*
scan_is(mask m, const charT*, const charT* high) const;
```

Finds the first character in the range `[low,high)` that matches the classification indicated by the mask `m`. The classification is matched by checking for `table()[(unsigned char) p] & m`, where `p` is in the range `[low,high)`. Returns the first `p` that matches, or `high` if none do.

```
const charT*
scan_not(mask m, const charT* low, const charT* high) const;
```

Finds the first character in the range `[low,high)` that does not match the classification indicated by the mask `m`. The classification is matched by checking for `!(table()[(unsigned char) p] & m)`, where `p` is in the range [low,high). Returns the first `p` that matches, or `high` if none do.

```
const mask*
table() const throw();
```

If the `tbl` argument that was passed to the constructor was non-zero, then this function returns that argument, otherwise it returns `classic_table()`.

```
charT
tolower(charT c) const;
const charT*
tolower(charT* low, const charT* high) const;
```

Returns `do_tolower(c)` or `do_tolower(low,high).`

```
charT
toupper(charT) const;
const charT*
toupper(charT* low, const charT* high) const;
```

Returns `do_toupper(c)` or `do_toupper(low,high).`

```
charT
widen(char c) const;
```

Returns `c.`

```
const char*
widen(const char* low, const char* high, charT* to) const;
```

Performs `::memcpy(to,low,high-low`. Returns `high.`

Facet ID

```
static locale::id id;
```

Unique identifier for this type of facet.

Protected Member Functions

```
virtual charT
do_tolower(charT) const;
```

Returns the lower case representation of `c,` if such exists, otherwise returns `c`;

```
virtual const charT*
do_tolower(charT* low, const charT* high) const;
```

Converts each character in the range `[low,high)` to its lower case representation, if such exists. If a lower case representation does not exist then the character is not changed. Returns `high`.

```
virtual charT
do_toupper(charT c) const;
```

Returns the upper case representation of `c,` if such exists, otherwise returns `c`;

```
virtual const charT*
do_toupper(charT* low, const charT* high) const;
```

Converts each character in the range `[low,high)` to its upper case representation, if such exists. If an upper case representation does not exist then the character is not changed. Returns `high`.

See Also

locale, facets, collate, ctype<char>, ctype_byname

ctype_byname

$$\begin{tikzcd}
 & & \textit{ctype\_base} \\
\textit{ctype\_byname} \arrow[r] & \textit{ctype} \arrow[ur] \arrow[dr] & \\
 & & \textit{locale::facet}
\end{tikzcd}$$

Summary

A facet that provides character classification facilities based on the named locales.

Synopsis

```
#include <locale>
template <class charT> class ctype_byname;
template <> class ctype_byname<char>;
```

Description

ctype_byname<charT> template and ***ctype_byname<char>*** specialization provide the same functionality as the ***ctype<charT>*** template, but specific to a particular named locale. For a description of the member functions of ***ctype_byname***, see the reference for ***ctype<charT>***. Only the constructor is described here.

Interface

```
template <class charT>
class ctype_byname : public ctype<charT> {
public:
  explicit ctype_byname(const char*, size_t = 0);
protected:
  ~ctype_byname();  // virtual
  virtual bool        do_is(mask, charT) const;
  virtual const charT* do_is(const charT*, const charT*,
                             mask*) const;
  virtual const char*  do_scan_is(mask,
                                  const charT*,
                                  const charT*) const;
  virtual const char*  do_scan_not(mask,
                                   const charT*,
                                   const charT*) const;
  virtual charT        do_toupper(charT) const;
  virtual const charT* do_toupper(charT*,
                                  const charT*) const;
  virtual charT        do_tolower(charT) const;
  virtual const charT* do_tolower(charT*,
                                  const charT*) const;
  virtual charT        do_widen(char) const;
  virtual const char*  do_widen(const char*, const char*,
                                charT*) const;
  virtual char         do_narrow(charT, char) const;
  virtual const charT* do_narrow(const charT*, const charT*,
                                 char, char*) const;
};

class ctype_byname<char> : public ctype<charT> {
public:
  explicit ctype_byname(const char*, size_t = 0);
protected:
  ~ctype_byname();  // virtual
  virtual char        do_toupper(char) const;
  virtual const char* do_toupper(char*, const char*) const;
  virtual char        do_tolower(char) const;
  virtual const char* do_tolower(char*, const char*) const;
};
```

Constructor

`explicit` **`ctype_byname`**`(const char* name, size_t refs = 0);`

Construct a ***ctype_byname*** facet. The facet will provide character classification relative to the named locale specified by the `name` argument. If the `refs` argument is 0 then destruction of the object is delegated to the locale, or locales, containing it. This allows the user to ignore lifetime management issues. On the other had, if `refs` is 1 then the object must be explicitly deleted; the locale will not do so. In this case, the object can be maintained across the lifetime of multiple locales.

See Also

locale, facets, collate, ctype<char>, ctype_byname

Summary

Family of classes used to encapsulate categories of locale functionality.

Description

The Standard C++ Localization library provides a locale interface that contains a collection of diverse facets. Each facet provides localization facilities for some specific area, such as character classification or numeric formatting. Each facet also falls into one or more broad categories. These categories are defined in the locale class, and the standard facets fit into these categories as follows.

Category	Facets
`collate`	`collate, collate_byname`
`ctype`	`ctype, codecvt, ctype_byname, codecvt_byname`
`monetary`	`moneypunct, moneypunct_byname, money_put, money_get`
`numeric`	`numpunct, numpunct_byname, num_put, num_get`
`time`	`time_put, time_put_byname, time_get, time_get_byname`
`messages`	`messages, messages_byname`

A facet must satisfy two properties. First, it must be derived from the base class `locale::facet`, either directly or indirectly (for example, `facet -> ctype<char> -> my_ctype`). Second, it must contain a member of type `locale::id`. This ensures that the locale class can manage its collection of facets properly.

See Also

locale, specific facet reference sections

Synopsis

```
#include <rw/iotraits>
template<class stateT = mbstate_t>
class fpos
```

Description

The template class ***fpos<stateT>*** is used by the ***iostream*** classes to maintain positioning information. It maintains three kinds of information: the absolute position, the conversion state and the validity of the stored position. Streams instantiated on tiny characters use ***streampos*** as their positioning type, whereas streams instantiated on wide characters use ***wstreampos***, but both are defined as ***fpos<mbstate_t>***.

Interface

```
template <class stateT = mbstate_t>
class fpos {

 public:

  typedef stateT  state_type;

  fpos(long off = 0);
  fpos(state_type);

  state_type   state(state_type);
  state_type   state () const;

};
```

Types

`state_type`
The type `state_type` holds the conversion state, and is compatible with the function `locale::codecvt()`. By default it is defined as `mbstate_t`.

Public Constructors

`fpos``(long off =0);`
Constructs an `fpos` object, initializing its position with `off` and its conversion state with the default `stateT` constructor. This function is not described in the C++ standard.

`fpos``(state_type st);`
Construct an `fpos` object, initializing its conversion state with `st`, its position with the start position, and its status to `good`.

Public Member Functions

`state_type`
`state``() const;`
Returns the conversion state stored in the fpos object.

```
state_type
state(state_type st);
```

Store st as the new conversion state in the fpos object and return its previous value.

Valid Operations

In the following,

- `P` refers to type `fpos<stateT>`
- `p` and `q` refer to an value of type `fpos<stateT>`
- `O` refers to the offset type (`streamoff`, `wstreamoff`, `long` …)
- `o` refers to a value of the offset type
- `i` refers to a value of type `int`

Valid operations:

`P p( i );`	Constructs from `int`.
`P p = i;`	Assigns from `int`.
`P( o )`	Converts from offset.
`O( p )`	Converts to offset.
`p == q`	Tests for equality.
`p != q`	Tests for inequality.
`q = p + o`	Adds offset.
`p += o`	Adds offset.
`q = p -o`	Subtracts offset.
`q -= o`	Subtracts offset.
`o = p - q`	Returns offset.

See Also

iosfwd(3C++), ***char_traits***(3C++)

Working Paper for Draft Proposed International Standard for Information Systems--Programming Language C++, Section 27.4.

Amendment 1 to the C Standard.

Standards Conformance

ANSI X3J16/ISO WG21 Joint C++ Committee

has_facet

Locale Function

Summary

A function template used to determine if a locale has a given facet.

Synopsis

```
#include <locale>
template <class Facet> bool has_facet(const locale&) throw();
```

Description

has_facet returns true if the requested facet is available in the locale, otherwise it returns false. You specify the facet type by explicitly providing the template parameter. (See the example below.)

Note that if your compiler cannot overload function templates on return type then you'll need to use an alternative ***has_facet*** template. The alternative template takes an additional argument that's a pointer to the type of facet you want to check on. The declaration looks like this:

```
template <class Facet>
const bool has_facet(const locale&, Facet*) throw();
```

The example below shows the use of both variations of ***has_facet***.

Example

```
//
// hasfacet.cpp
//

#include <iostream>

int main ()
{
  using namespace std;

  locale loc;

#ifndef _RWSTD_NO_TEMPLATE_ON_RETURN_TYPE
  cout << has_facet<ctype<char> >(loc) << endl;
#else
  cout << has_facet(loc,(ctype<char>*)0) << endl;
#endif

  return 0;
}
```

See Also

locale, facets, use_facet

ios_base

Base Class

Synopsis

```
#include <ios>
class ios_base;
```

Description

The class ***ios_base*** defines several member types:

- A class `failure` derived from `exception`.
- A class `Init.`
- Three bitmask types, `fmtflags`, `iostate` and `openmode.`
- Two enumerated types, `seekdir`, and `event.`

It maintains several kinds of data:

- Control information that influences how to interpret (format) input sequences and how to generate (format) output sequences.
- Locale object used within the stream classes.
- Additional information that is stored by the program for its private use.

Interface

```
 class ios_base {

  public:

    class failure : public exception {

       public:

           explicit failure(const string& msg);
           virtual ~failure() throw();
           virtual const char* what() const throw();
     };

    typedef int      iostate;

    enum io_state {
                   goodbit     = 0x00,
                   badbit      = 0x01,
                   eofbit      = 0x02,
                   failbit     = 0x04
                  };

    typedef int      openmode;

    enum open_mode {
```

```
                 app         = 0x01,
                 binary      = 0x02,
                 in          = 0x04,
                 out         = 0x08,
                 trunc       = 0x10,
                 ate         = 0x20
               };

typedef int      seekdir;

enum seek_dir {
                  beg         = 0x0,
                  cur         = 0x1,
                  end         = 0x2
              };


typedef int      fmtflags;

enum fmt_flags {
                 boolalpha   = 0x0001,
                 dec         = 0x0002,
                 fixed       = 0x0004,
                 hex         = 0x0008,
                 internal    = 0x0010,
                 left        = 0x0020,
                 oct         = 0x0040,
                 right       = 0x0080,
                 scientific  = 0x0100,
                 showbase    = 0x0200,
                 showpoint   = 0x0400,
                 showpos     = 0x0800,
                 skipws      = 0x1000,
                 unitbuf     = 0x2000,
                 uppercase   = 0x4000,
                 adjustfield = left | right | internal,
                 basefield   = dec | oct | hex,
                 floatfield  = scientific | fixed
               };

enum event     {
                 erase_event   = 0x0001,
                 imbue_event   = 0x0002,
                 copyfmt_event = 0x004
               };

typedef void (*event_callback) (event, ios_base&, int index);

void register_callback(event_callback fn, int index);

class Init;

fmtflags flags() const;
fmtflags flags(fmtflags fmtfl);
fmtflags setf(fmtflags fmtfl);
fmtflags setf(fmtflags fmtfl, fmtflags mask);

void unsetf(fmtflags mask);
```

```
   ios_base& copyfmt(const ios_base& rhs);

   streamsize precision() const;
   streamsize precision(streamsize prec);

   streamsize width() const;
   streamsize width(streamsize wide);

   locale imbue(const locale& loc);
   locale getloc() const

   static int xalloc();

   long&  iword(int index);
   void*& pword(int index);

   bool synch_with_stdio(bool sync = true);
   bool is_synch();

  protected:

   ios_base();
   ~ios_base();

  private:

   union ios_user_union {
                         long  lword;
                         void* pword;
                      };

   union ios_user_union *userwords_;
};

 ios_base& boolalpha(ios_base&);
 ios_base& noboolalpha(ios_base&);
 ios_base& showbase(ios_base&);
 ios_base& noshowbase(ios_base&);
 ios_base& showpoint(ios_base&);
 ios_base& noshowpoint(ios_base&);
 ios_base& showpos(ios_base&);
 ios_base& noshowpos(ios_base&);
 ios_base& skipws(ios_base&);
 ios_base& noskipws(ios_base&);
 ios_base& uppercase(ios_base&);
 ios_base& nouppercase(ios_base&);
 ios_base& internal(ios_base&);
 ios_base& left(ios_base&);
 ios_base& right(ios_base&);
 ios_base& dec(ios_base&);
 ios_base& hex(ios_base&);
 ios_base& oct(ios_base&);
 ios_base& fixed(ios_base&);
 ios_base& scientific(ios_base&);
 ios_base& unitbuf(ios_base&);
 ios_base& nounitbuf(ios_base&);
```

Types

fmtflags

The type fmtflags is a bitmask type. Setting its elements has the following effects:

showpos	Generates a + sign in non-negative generated numeric output.
showbase	Generates a prefix indicating the numeric base of generated integer output.
uppercase	Replaces certain lowercase letters with their uppercase equivalents in generated output.
showpoint	Generates a decimal-point character unconditionally in generated floating-point output.
boolalpha	Inserts and extracts bool type in alphabetic format.
unitbuf	Flushes output after each output operation.
internal	Adds fill characters at a designated internal point in certain generated output, or identical to right if no such point is designated.
left	Adds fill characters on the right (final positions) of certain generated output.
right	Adds fill characters on the left (initial positions) of certain generated output.
dec	Converts integer input or generates integer output in decimal base.
hex	Converts integer input or generates integer output in hexadecimal base.
oct	Converts integer input or generates integer output in octal base.
fixed	Generates floating-point output in fixed-point notation.
scientific	Generates floating-point output in scientific notation.
skipws	Skips leading white space before certain input operation.

iostate

The type iostate is a bitmask type. Setting its elements has the following effects:

badbit	Indicates a loss of integrity in an input or output sequence.

`eofbit`	Indicates that an input operation reached the end of an input sequence.
`failbit`	Indicates that an input operation failed to read the expected characters, or that an output operation failed to generate the desired characters.

openmode

The type `openmode` is a bitmask type. Setting its elements has the following effects:

`app`	Seeks to the end before writing.
`ate`	Opens and seek to end immediately after opening.
`binary`	Performs input and output in binary mode.
`in`	Opens for input.
`out`	Opens for output.
`trunc`	Truncates an existing stream when opening.

seekdir

The type `seekdir` is a bitmask type. Setting its elements has the following effects:

`beg`	Requests a seek relative to the beginning of the stream.
`cur`	Requests a seek relative to the current position within the sequence.
`end`	Requests a seek relative to the current end of the sequence.

event_callback

The type `event_callback` is the type of the callback function used as a parameter in the function `register_callback`. These functions allow you to use the `iword`, `pword` mechanism in an exception-safe environment.

Public Constructor

ios_base();

The `ios_base` members have an indeterminate value after construction.

Public Destructor

~ios_base();

Destroys an object of class `ios_base`. Calls each registered callback pair `(fn, index)` as `(*fn)(erase_event,*this, index)` at such a time that any `ios_base` member function called from within `fn` has well-defined results.

Public Member Functions

```
ios_base&
copyfmt(const ios_base& rhs);
```

Assigns to the member objects of `*this` the corresponding member objects of `rhs`. The content of the union pointed at by `pword` and `iword` is copied, not the pointers itself. Before copying any parts of `rhs`, calls each registered callback pair `(fn,index)` as `(*fn)(erase_even,*this, index)`. After all parts have been replaced, calls each callback pair that was copied from `rhs` as `(*fn)(copy_event,*this,index)`.

```
fmtflags
flags() const;
```

Returns the format control information for both input and output

```
fmtflags
flags(fmtflags fmtfl);
```

Saves the format control information, then sets it to `fmtfl` and returns the previously saved value.

```
locale
getloc() const;
```

Returns the imbued locale, to be used to perform locale-dependent input and output operations. The default locale, `locale::locale()`, is used if no other locale object has been imbued in the stream by a call to the `imbue` function.

```
locale
imbue(const locale& loc);
```

Saves the value returned by `getloc()` then assigns `loc` to a private variable and calls each registered callback pair `(fn, index)` as `(*fn)(imbue_event,*this, index)`. It then returns the previously saved value.

```
bool
is_sync();
```

Returns true if the C++ standard streams and the standard C streams are synchronized, otherwise returns false. This function is not part of the C++ standard.

```
long&
iword(int idx);
```

Returns `userwords_[idx].lword`. If `userwords_` is a null pointer, allocates a union of `long` and `void*` of unspecified size and stores a pointer to its first element in `userwords_`. The function then extends the union pointed at by `userwords_` as necessary to include the element `userwords_[idx]`. Each newly allocated element of the union is initialized to zero. The reference returned may become invalid after another call to the object's `iword` or `pword` member with a different index, after a call to its `copyfmt` member, or when the object is destroyed.

```
streamsize
precision() const;
```

Returns the precision (number of digits after the decimal point) to generate on certain output conversions.

```
streamsize
precision(streamsize prec);
```

Saves the precision then sets it to `prec` and returns the previously saved value.

```
void*&
pword(int idx);
```

Returns `userword_[idx].pword`. If `userwords_` is a null pointer, allocates a union of `long` and `void*` of unspecified size and stores a pointer to its first element in `userwords_`. The function then extends the union pointed at by `userwords_` as necessary to include the element `userwords_[idx]`. Each newly allocated element of the array is initialized to zero. The reference returned may become invalid after another call to the object's `pword` or `iword` member with different index, after a call to its `copyfmt` member, or when the object is destroyed.

```
void
register_callback(event_callback fn, int index);
```

Registers the pair (fn, index) such that during calls to `imbue()`,`copyfmt()`, or `~ios_base()`, the function `fn` is called with argument `index`. Functions registered are called when an event occurs, in opposite order of registration. Functions registered while a callback function is active are not called until the next event. Identical pairs are not merged; a function registered twice is called twice per event.

```
fmtflags
setf(fmtflags fmtfl);
```

Saves the format control information, then sets it to `fmtfl` and returns the previously saved value.

```
fmtflags
setf(fmtflags fmtfl, fmtflags mask);
```

Saves the format control information, then clears `mask` in `flags()`,sets `fmtfl & mask in flags()` and returns the previously saved value.

```
bool
sync_with_stdio(bool sync = true);
```

Called with a false argument, it allows the C++ standard streams to operate independently of the standard C streams, which will greatly improve performance when using the C++ standard streams. Called with a true argument it restores the default synchronization. The return value of the function is the status of the synchronization at the time of the call.

```
void
unsetf(fmtflags mask);
```

Clears `mask` in `flags()`.

```
streamsize
width() const;
```

Returns the field width (number of characters) to generate on certain output conversions.

```
streamsize
width(streamsize wide);
```

Saves the field width then sets it to `wide` and returns the previously saved value.

```
static int
xalloc();
```

Returns the next static index that can be used with `pword` and `iword`. This is useful if you want to share data between several stream objects.

The Class failure

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

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

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

```
const char*
what() const;
```

Returns the message `msg` with which the exception was created.

Class Init

The class ***Init*** describes an object whose construction ensures the construction of the eight objects declared in `<iostream>` that associate file stream buffers with the standard C streams.

Non-Member Functions

```
ios_base&
boolalpha(ios_base& str);
```

Calls `str.setf(ios_base::boolalpha)` and returns `str.`

```
ios_base&
dec(ios_base& str);
```

Calls `str.setf(ios_base::dec, ios_base::basefield)` and returns `str`.

```
ios_base&
fixed(ios_base& str);
```

Calls `str.setf(ios_base::fixed, ios_base::floatfield)` and returns `str`.

`ios_base&`
`hex``(ios_base& str);`
Calls `str.setf(ios_base::hex, ios_base::basefield)` and returns `str`.

`ios_base&`
`internal``(ios_base& str);`
Calls `str.setf(ios_base::internal, ios_base::adjustfield)` and returns `str`.

`ios_base&`
`left``(ios_base& str);`
Calls `str.setf(ios_base::left, ios_base::adjustfield)` and returns `str`.

`ios_base&`
`noboolalpha``(ios_base& str);`
Calls `str.unsetf(ios_base::boolalpha)` and returns `str.`

`ios_base&`
`noshowbase``(ios_base& str);`
Calls `str.unsetf(ios_base::showbase)` and returns `str`.

`ios_base&`
`noshowpoint``(ios_base& str);`
Calls `str.unsetf(ios_base::showpoint)` and returns `str`.

`ios_base&`
`noshowpos``(ios_base& str);`
Calls `str.unsetf(ios_base::showpos)` and returns `str`.

`ios_base&`
`noskipws``(ios_base& str);`
Calls `str.unsetf(ios_base::skipws)` and returns `str`.

`ios_base&`
`nounitbuf``(ios_base& str);`
Calls `str.unsetf(ios_base::unitbuf)` and returns `str`.

`ios_base&`
`nouppercase``(ios_base& str);`
Calls `str.unsetf(ios_base::uppercase)` and returns `str`.

`ios_base&`
`oct``(ios_base& str);`
Calls `str.setf(ios_base::oct, ios_base::basefield)` and returns `str`.

`ios_base&`
`**right**(ios_base& str);`
Calls `str.setf(ios_base::right, ios_base::adjustfield)` and returns `str`.

`ios_base&`
`**scientific**(ios_base& str);`
Calls `str.setf(ios_base::scientific, ios_base::floatfield)` and returns `str`.

`ios_base&`
`**showbase**(ios_base& str);`
Calls `str.setf(ios_base::showbase)` and returns `str`.

`ios_base&`
`**showpoint**(ios_base& str);`
Calls `str.setf(ios_base::showpoint)` and returns `str`.

`ios_base&`
`**showpos**(ios_base& str);`
Calls `str.setf(ios_base::showpos)` and returns `str`.

`ios_base&`
`**skipws**(ios_base& str);`
Calls `str.setf(ios_base::skipws)` and returns `str`.

`ios_base&`
`**unitbuf**(ios_base& str);`
Calls `str.setf(ios_base::unitbuf)` and returns `str`.

`ios_base&`
`**uppercase**(ios_base& str);`
Calls `str.setf(ios_base::uppercase)` and returns `str`.

See Also

basic_ios(3C++), ***basic_istream***(3C++), ***basic_ostream***(3C++), ***char_traits***(3C++)

Working Paper for Draft Proposed International Standard for Information Systems--Programming Language C++, Section 27.4.3

Standards Conformance

ANSI X3J16/ISO WG21 Joint C++ Committee

iosfwd

Description

The header iosfwd forward declares the input/output library template classes and specializes them for wide and tiny characters. It also defines the positional types used in class char_traits instantiated on tiny and wide characters.

Synopsis

```
 #include <iosfwd>

 // forward declare the traits class
template<class charT> struct char_traits;

// forward declare the positioning class
template<class stateT> class fpos;

// forward declare the state class
class mbstate_t;

// forward declare the allocator class
template<class T> class allocator;

// forward declare the iostreams template classes
template<class charT,class traits=char_traits<charT>>
    class basic_ios;
template<class charT,class traits=char_traits<charT>>
    class basic_streambuf;
template<class charT,class traits=char_traits<charT>>
    class basic_istream;
template<class charT,class traits=char_traits<charT>>
    class basic_ostream;
template<class charT,class traits=char_traits<charT>,
         class Allocator = allocator<void> >
    class basic_stringbuf;
template<class charT,class traits=char_traits<charT>,
         class Allocator = allocator<void> >
    class basic_istringstream;
template<class charT,class traits=char_traits<charT>,
         class Allocator = allocator<void> >
    class basic_ostringstream;
```

```
template<class charT,class traits=char_traits<charT>>
    class basic_filebuf;
template<class charT,class traits=char_traits<charT>>
    class basic_ifstream;
template<class charT,class traits=char_traits<charT>>
    class basic_ofstream;
template<class charT,class traits=char_traits<charT>>
    class ostreambuf_iterator;
template<class charT,class traits=char_traits<charT>>
    class istreambuf_iterator;
template<class charT,class traits=char_traits<charT>>
    class basic_iostream;
template<class charT,class traits=char_traits<charT>,
         class Allocator = allocator<void> >
    class basic_stringstream;
template<class charT,class traits=char_traits<charT>>
    class basic_fstream;

// specializations on tiny characters
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_iostream<char>         iostream;
typedef basic_stringstream<char>     stringstream;
typedef basic_fstream<char>          fstream;

// specializations on wide characters
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;
typedef basic_iostream<wchar_t>      wiostream;
typedef basic_stringstream<wchar_t>  wstringstream;
typedef basic_fstream<wchar_t>       wfstream;

// positional types used by char_traits
typedef fpos<mbstate_t> streampos;
typedef fpos<mbstate_t> wstreampos;

typedef long            streamoff;
typedef long            wstreamoff;
```

See Also

fpos(3C++), ***char_traits***(3C++), ***basic_ios***(3C++), ***basic_streambuf***(3C++),

basic_istream(3C++), ***basic_ostream***(3C++), ***basic_iostream***(3C++), ***basic_stringbuf***(3C++), ***basic_istringstream***(3C++), ***basic_ostringstream***(3C++), ***basic_stringstream***(3C++), ***basic_filebuf***(3C++), ***basic_ifstream***(3C++), ***basic_ofstream***(3C++), ***basic_fstream***(3C++), ***istreambuf_iterator***(3C++), ***ostreambuf_iterator***(3C++)

Working Paper for Draft Proposed International Standard for Information Systems--Programming Language C++, Section 27.2

Standards Conformance

ANSI X3J16/ISO WG21 Joint C++ Committee

Locale Convenience Function

Summary

Determines if a character is alphabetic or numeric.

Synopsis

```
#include <locale>
template <class charT>

bool isalnum (charT c, const locale& loc) const;
```

Description

The ***isalnum*** function returns true if the character passed as a parameter is either part of the alphabet specified by the locale parameter or a decimal digit, otherwise the function returns false. The check is made using the `ctype` facet from the locale parameter.

Example

```
//
// isalnum.cpp
//

#include <iostream>

int main ()
{
  using namespace std;

  locale loc;

  cout << isalnum('a',loc) << endl;
  cout << isalnum(',',loc) << endl;

  return 0;
}
```

See Also

other is... functions, locale, ctype

isalpha

Locale Convenience Function

Summary Determines if a character is alphabetic.

Synopsis

```
#include <locale>

template <class charT>
bool isalpha (charT c, const locale& loc) const;
```

Description The ***isalpha*** function returns true if the character passed as a parameter is part of the alphabet specified by the locale parameter, otherwise the function returns false.   The check is made using the `ctype` facet from the locale parameter.

See Also ***other is... functions, locale, ctype***

iscntrl

Locale Convenience Function

Summary Determines if a character is a control character.

Synopsis

```
#include <locale>

template <class charT>
bool iscntrl (charT c, const locale& loc) const;
```

Description The ***iscntrl*** function returns true if the character passed as a parameter is a control character, otherwise the function returns false. The check is made using the `ctype` facet from the locale parameter.

See Also ***other is... functions, locale, ctype***

isdigit

Locale Convenience Function

Summary Determines if a character is a decimal digit.

Synopsis

```
#include <locale>

template <class charT>
bool isdigit (charT c, const locale& loc) const;
```

Description The ***isdigit*** function returns true if the character passed as a parameter is a decimal digit, otherwise the function returns false. The check is made using the `ctype` facet from the locale parameter.

See Also ***other is... functions, locale, ctype***

Locale Convenience Function

Summary

Determines if a character is a graphic character.

Synopsis

```
#include <locale>

template <class charT>
bool isgraph (charT c, const locale& loc) const;
```

Description

The ***isgraph*** function returns true if the character passed as a parameter is a graphic character, otherwise the function returns false. The check is made using the `ctype` facet from the locale parameter.

See Also

other is... functions, locale, ctype

Locale Convenience Function

Summary

Determines whether a character is lower case.

Synopsis

```
#include <locale>

template <class charT>
bool islower (charT c, const locale& loc) const;
```

Description

The ***islower*** function returns true if the character passed as a parameter is lower case, otherwise the function returns false. The check is made using the `ctype` facet from the locale parameter.

See Also

other is... functions, locale, ctype

isprint

Locale Convenience Function

Summary Determines if a character is printable.

Synopsis

```
#include <locale>

template <class charT>
bool isprint (charT c, const locale& loc) const;
```

Description The ***isprint*** function returns true if the character passed as a parameter is printable, otherwise the function returns false. The check is made using the `ctype` facet from the locale parameter.

See Also ***other is... functions, locale, ctype***

ispunct

Locale Convenience Function

Summary Determines if a character is punctuation.

Synopsis

```
#include <locale>

template <class charT>
bool ispunct (charT c, const locale& loc) const;
```

Description The ***ispunct*** function returns true if the character passed as a parameter is punctuation, otherwise the function returns false. The check is made using the `ctype` facet from the locale parameter.

See Also ***other is... functions, locale, ctype***

Locale Convenience Function

Summary

Determines if a character is a space.

Synopsis

```
#include <locale>

template <class charT>
bool isspace (charT c, const locale& loc) const;
```

Description

The ***isspace*** function returns true if the character passed as a parameter is a space character, otherwise the function returns false. The check is made using the `ctype` facet from the locale parameter.

See Also

other is... functions, locale, ctype

istreambuf_iterator

istreambuf_iterator → ***input_iterator***

Synopsis

```
#include <streambuf>
template<class charT, class traits = char_traits<charT> >
class istreambuf_iterator
: public input_iterator
```

Description

The template class ***istreambuf_iterator*** reads successive characters from the stream buffer for which it was constructed. `operator*` provides access to the current input character, if any, and `operator++` advances to the next input character. If the end of stream is reached, the iterator becomes equal to the end of stream iterator value, which is constructed by the default constructor, `istreambuf_iterator()`. An ***istreambuf_iterator*** object can be used only for one-pass-algorithms.

Interface

```
template<class charT, class traits = char_traits<charT> >
class istreambuf_iterator
: public input_iterator {

 public:

  typedef charT                            char_type;
  typedef typename traits::int_type        int_type;
  typedef traits                           traits_type;
  typedef basic_streambuf<charT, traits> streambuf_type;
  typedef basic_istream<charT, traits>   istream_type;

  class proxy;

  istreambuf_iterator() throw();
  istreambuf_iterator(istream_type& s)  throw();
  istreambuf_iterator(streambuf_type *s) throw();
  istreambuf_iterator(const proxy& p) throw();

  char_type operator*();
  istreambuf_iterator<charT, traits>& operator++();
  proxy operator++(int);
  bool equal(istreambuf_iterator<charT, traits>& b);

};

template<class charT, class traits>
bool operator==(istreambuf_iterator<charT, traits>& a,
                istreambuf_iterator<charT, traits>& b);
```

Types

`char_type`

The type `char_type` is a synonym for the template parameter `charT`.

int_type
The type `int_type` is a synonym of type `traits::in_type`.

istream_type
The type `istream_type` is an instantiation of class `basic_istream` on types `charT` and `traits`:

```
typedef basic_istream<charT, traits>   istream_type;
```

streambuf_type
The type `streambuf_type` is an instantiation of class `basic_streambuf` on types `charT` and `traits`:

```
typedef basic_streambuf<charT, traits> streambuf_type;
```

traits_type
The type `traits_type` is a synonym for the template parameter `traits`.

Nested Class Proxy

Class ***istreambuf_iterator<charT,traits>::proxy*** provides a temporary placeholder as the return value of the post-increment operator. It keeps the character pointed to by the previous value of the iterator for some possible future access.

Constructors

```
istreambuf_iterator()
 throw();
```
Constructs the end of stream iterator.

```
istreambuf_iterator(istream_type& s)
 throw();
```
Constructs an `istreambuf_iterator` that inputs characters using the `basic_streambuf` object pointed to by `s.rdbuf()`. If `s.rdbuf()` is a null pointer, the `istreambuf_iterator` is the end-of-stream iterator.

```
istreambuf_iterator(streambuf_type *s)
 throw();
```
Constructs an `istreambuf_iterator` that inputs characters using the `basic_streambuf` object pointed at by `s`. If `s` is a null pointer, the `istreambuf_iterator` is the end-of-stream iterator.

```
istreambuf_iterator(const proxy& p)
 throw();
```
Constructs an `istreambuf_iterator` that uses the `basic_streambuf` object embedded in the proxy object.

Member Operators

```
char_type
operator*();
```
Returns the character pointed at by the input sequence of the attached stream buffer. If no character is available, the iterator becomes equal to the end-of-stream iterator.

```
istreambuf_iterator<charT, traits>&
operator++();
```

Increments the input sequence of the attached stream buffer to point to the next character. If the current character is the last one, the iterator becomes equal to the end-of-stream iterator.

```
proxy
operator++(int);
```

Increments the input sequence of the attached stream buffer to point to the next character. If the current character is the last one, the iterator becomes equal to the end-of-stream iterator. The proxy object returned contains the character pointed at before carrying out the post-increment operator.

Public Member Function

```
bool
equal(istreambuf_iterator<charT, traits>& b);
```

Returns true if and only if both iterators are at end of stream, or neither is at end of stream, regardless of what stream buffer object they are using.

Non Member Functions

```
template<class charT, class traits>
bool
operator==(istreambuf_iterator<charT, traits>& a,
                 istreambuf_iterator<charT, traits>& b);
```

Returns `a.equal(b)`.

Examples

```
//
// stdlib/examples/manual/istreambuf_iterator.cpp
//
#include<iostream>
#include<fstream>

void main ( )
{
  using namespace std;

  // open the file is_iter.out for reading and writing
  ofstream out("is_iter.out", ios_base::out | ios_base::in );

  // output the example sentence into the file
  out << "Ceci est un simple example pour demontrer le" << endl;
  out << "fonctionement de istreambuf_iterator";

  // seek to the beginning of the file
  out.seekp(0);

  // construct an istreambuf_iterator pointing to
  // the ofstream object underlying stream buffer
  istreambuf_iterator<char> iter(out.rdbuf());

  // construct an end of stream iterator
  istreambuf_iterator<char> end_of_stream_iterator;

  cout << endl;

  // output the content of the file
  while( !iter.equal(end_of_stream_iterator) )

  // use both operator++ and operator*
  cout << *iter++;

  cout << endl;

}
```

See Also

basic_streambuf(3C++), ***basic_istream***(3C++), ***ostreambuf_iterator***(3C++)

Working Paper for Draft Proposed International Standard for Information Systems--Programming Language C++, Section 24.4.3

Standards Conformance

ANSI X3J16/ISO WG21 Joint C++ Committee

istrstream ⟶ ***basic_istream*** ⟶ ***basic_ios*** ⟶ ***ios_base***

Synopsis

```
#include <strstream>
class istrstream
: public basic_istream<char>
```

Description

The class ***istrstream*** provides functionality to read characters from an array in memory. It uses a private ***strstreambuf*** object to control the associated array object. It inherits from ***basic_istream<char>*** and therefore can use all the formatted and unformatted input functions.

Interface

```
class istrstream : public basic_istream<char> {

 public:

  typedef char_traits<char>             traits;

  typedef char                         char_type;
  typedef typename traits::int_type    int_type;
  typedef typename traits::pos_type    pos_type;
  typedef typename traits::off_type    off_type;

  explicit istrstream(const char *s);
  istrstream(const char *s, streamsize n);
  explicit istrstream(char *s);
  istrstream(char *s, streamsize n);

  virtual ~istrstream();

  strstreambuf *rdbuf() const;

  char *str();

};
```

Types

`char_type`
The type `char_type` is a synonym of type `char`.

`int_type`
The type `int_type` is a synonym of type `traits::in_type`.

`off_type`
The type `off_type` is a synonym of type `traits::off_type`.

`pos_type`
The type `pos_type` is a synonym of type `traits::pos_type`.

`traits`

The type `traits` is a synonym of type `char_traits<char>`.

Constructors

```
explicit istrstream(const char* s);
explicit istrstream(char* s);
```

Constructs an object of class `istrstream`, initializing the base class `basic_istream<char>` with the associated `strstreambuf` object. The `strstreambuf` object is initialized by calling `strstreambuf(s,0)`, where `s` shall designate the first element of an NTBS.

```
explicit istrstream(const char* s, streamsize n);
explicit istrstream(char* s, streamsize n);
```

Constructs an object of class `istrstream`, initializing the base class `basic_istream<char>` with the associated `strstreambuf` object. The `strstreambuf` object is initialized by calling `strstreambuf(s,n)`, where s shall designate the first element of an array whose length is `n` elements, and `n` shall be greater than zero.

Destructors

```
virtual ~istrstream();
```

Destroys an object of class `istrstream`.

Member Functions

```
char*
str();
```

Returns a pointer to the underlying array object which may be null.

```
strstreambuf*
rdbuf() const;
```

Returns a pointer to the private `strstreambuf` object associated with the stream.

Examples

```
//
// stdlib/examples/manual/istrstream.cpp
//
#include<iostream>
#include<strstream>

void main ( )
{
  using namespace std;

  const char* p="C'est pas l'homme qui prend la mer, ";
  char* s="c'est la mer qui prend l'homme";

  // create an istrstream object and initialize
  // the underlying strstreambuf with p
  istrstream in_first(p);

  // create an istrstream object and initialize
  // the underlying strstreambuf with s
  istrstream in_next(s);

  // create an ostrstream object
  ostrstream out;
  // output the content of in_first and
  // in_next to out
  out << in_first.rdbuf() << in_next.str();

  // output the content of out to stdout
  cout << endl << out.rdbuf() << endl;


  // output the content of in_first to stdout
  cout << endl << in_first.str();

  // output the content of in_next to stdout
  cout << endl << in_next.rdbuf() << endl;

}
```

See Also

char_traits(3C++), ***ios_base***(3C++), ***basic_ios***(3C++), ***strstreambuf***(3C++), ***ostrstream***(3C++), ***strstream***(3c++)

Working Paper for Draft Proposed International Standard for Information Systems--Programming Language C++, Annex D Compatibility features Section D.6.2

Standards Conformance

ANSI X3J16/ISO WG21 Joint C++ Committee

isupper

Locale Convenience Function

Summary

Determines whether a character is upper case.

Synopsis

```
#include <locale>

template <class charT>
bool isupper (charT c, const locale& loc) const;
```

Description

The ***isupper*** function returns true if the character passed as a parameter is upper case, otherwise the function returns false. The check is made using the `ctype` facet from the locale parameter.

See Also

other is... functions, locale, ctype

Locale Convenience Function

Summary Determines whether a character is a hexadecimal digit.

Synopsis

```
#include <locale>

template <class charT>
bool isxdigit (charT c, const locale& loc) const;
```

Description The ***isxdigit*** function returns true if the character passed as a parameter is a hexadecimal digit, otherwise the function returns false. The check is made using the ctype facet from the locale parameter.

See Also ***other is... functions, locale, ctype***

locale

Localization Container

Summary

Localization class containing a polymorphic set of facets.

Synopsis

```
#include<locale>
class locale;
```

Description

locale provides a localization interface and a set of indexed facets, each of which covers one particular localization issue. The default locale object is constructed on the “C” locale. Locales can also be constructed on named locales.

A calling program can determine whether a particular facet is contained in a locale by using the `has_facet` function, and the program can obtain a reference to that facet with the `use_facet` function. These are not member functions, but instead take a locale object as an argument.

locale has several important characteristics.

First, successive calls to member functions will always return the same result. This allows a calling program to safely cache the results of a call.

Any standard facet not implemented by a locale will be obtained from the global locale the first time that `use_facet` is called (for that facet).

Only a locale constructed from a name (i.e., “POSIX”), from parts of two named locales, or from a stream, has a name. All other locales are unnamed. Only named locales may be compared for equality. An unnamed locale is equal only to itself.

Interface

```
class locale {
public:
  // types:
  class facet;
  class id;
  typedef int category;
  static const category   none, collate, ctype, monetary,
                          numeric, time, messages,
                          all = collate | ctype | monetary |
                                numeric | time  | messages;
  // construct/copy/destroy:
  locale() throw()
  locale(const locale&) throw()
  explicit locale(const char*);
  locale(const locale&, const char*, category);
  template <class Facet> locale(const locale&, Facet*);
  template <class Facet> locale(const locale&,
                                const locale&);
  locale(const locale&, const locale&, category);
  ~locale() throw();  // non-virtual
  const locale& operator=(const locale&) throw();
  // locale operations:
  basic_string<char>                  name() const;
  bool operator==(const locale&) const;
  bool operator!=(const locale&) const;
  template <class charT,Traits>
    bool operator()(const basic_string<charT,Traits>&,
                    const basic_string<charT,Traits>&) const;
  // global locale objects:
  static       locale  global(const locale&);
  static const locale& classic();
};

class locale::facet {
protected:
  explicit facet(size_t refs = 0);
  virtual ~facet();
private:
  facet(const facet&);          // not defined
  void operator=(const facet&); // not defined
};

class locale::id {
public:
  id();
private:
  void operator=(const id&); // not defined
  id(const id&);             // not defined
};
```

Types

category

Standard facets fall into eight broad categories. These are: `none`, `collate`, `ctype`, `monetary`, `numeric`, `time`, `messages`, and `all`. `all` is a combination of all the other categories except `none`. Bitwise operations may be applied to combine or screen these categories. For instance `all` is defined as:

```
(collate | ctype | monetary | numeric | time | messages)
```

locale member functions that take a category argument must be provided with one of the above values, or one of the constants from the old C locale (e.g., `LC_CTYPE`).

facet

Base class for facets. This class exists primarily to provide reference counting services to derived classes. All facets must derive from it, either directly or indirectly indirectly (e.g., `facet -> ctype<char> -> my_ctype`).

If the `refs` argument to the constructor is 0 then destruction of the object is delegated to the locale, or locales, containing it. This allows the user to ignore lifetime management issues. On the other had, if `refs` is 1, then the object must be explicitly deleted; the locale will not do so. In this case, the object can be maintained across the lifetime of multiple locales.

Copy construction and assignment of this class are disallowed.

`id`

Type used to index facets in the ***locale*** container. Every facet must contain a member of this type.

Copy construction and assignment of this type are disallowed.

Constructors and Destructors

```
locale()
  throw()
```

Constructs a default ***locale*** object. This locale will be the same as the last argument passed to `locale::global()`, or, if that function has not been called, the locale will be the same as the classic "C" locale.

```
locale(const locale& other)
  throw()
```

Constructs a copy of the locale argument `other`.

```
explicit locale(const char* std_name);
```

Constructs a ***locale*** object on the named locale indicated by `std_name`. Throws a `runtime_error` exception of if `std_name` is not a valid locale name.

```
locale(const locale& other, const char* std_name,
       category cat);
```

Construct a ***locale*** object that is a copy of `other`, except for the facets that are in the category specified by `cat`. These facets will be obtained from the named locale identified by `std_name`. Throws a `runtime_error` exception of if `std_name` is not a valid locale name.

Note that the resulting locale will have the same name (possibly none) as `other`.

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

Constructs a ***locale*** object that is a copy of `other`, except for the facet of type `Facet`. Unless f is null, it `f` is used to supply the missing facet, otherwise the facet will come from `other` as well.

Note that the resulting locale will not have a name.

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

Constructs a ***locale*** object that is a copy of `other`, except for the facet of type `Facet`. This missing facet is obtained from the second locale argument, `one`. Throws a `runtime_error` exception if one does not contain a facet of type `Facet`.

Note that the resulting locale will not have a name.

`locale``(const locale& other, const locale& one, category cat);`

Constructs a ***locale*** object that is a copy of `other`, except for facets that are in the category specified by category argument `cat`. These missing facets are obtained from the other locale argument, `one`.

Note that the resulting locale will only have a name only if both `other` and `one` have names. The name will be the same as `other's` name.

`~locale``();`

Destroy the locale.

Public Member Operators

`const locale&`
`operator=``(const locale& other) throw();`

Replaces `*this` with a copy of `other`. Returns `*this`.

`bool`
`operator==``(const locale& other) const;`

Returns `true` if both `other` and `*this` are the same object, if one is a copy of another, or if both have the same name. Otherwise returns `false`.

`bool`
`operator!=``(const locale& other) const;`

Returns `!(*this == other)`

```
template <class charT,Traits>
bool
operator()(const basic_string<charT,Traits>& s1,
           const basic_string<charT,Traits>& s2) const;
```

This operator is provided to allow a ***locale*** object to be used as a comparison object for comparing two strings. Returns the result of comparing the two strings using the `compare` member function of the `collate<charT>` facet contained in this. Specifically, this function returns the following:

```
use_facet< collate<charT> >(*this).compare(s1.data(),
  s1.data()+s1.size(), s2.data(),
  s2.data()+s2.size()) < 0;
```

This allows a `locale` to be used with standard algorithms, such as `sort`, to provide localized comparison of strings.

Public Member Functions

`basic_string<char>`
`name()`` const;`

Returns the name of `this`, if `this` has one; otherwise returns the string `"*"`.

Static Public Member Functions

```
static locale
global(const locale& loc);
```

Sets the global locale to `loc`. This causes future uses of the default constructor for `locale` to return a copy of `loc`. If `loc` has a name, this function has the further effect of calling `std::setlocale(LC_ALL,loc.name().c_str());`. Returns the previous value of `locale()`.

```
static const locale&
classic();
```

Returns a locale with the same semantics as the classic "C" locale.

Example

```
//
// locale.cpp
//

 #include <string>
 #include <vector>
 #include <iostream>
 #include "codecvte.h"

 int main ()
 {
  using namespace std;

  locale loc;  // Default locale

  // Construct new locale using default locale plus
  // user defined codecvt facet
  // This facet converts from ISO Latin
  // Alphabet No. 1 (ISO 8859-1) to
  // U.S. ASCII code page 437
  // This facet replaces the default for
  // codecvt<char,char,mbstate_t>
  locale my_loc(loc,new ex_codecvt);

  // imbue modified locale onto cout
  locale old = cout.imbue(my_loc);
  cout << "A \x93 jolly time was had by all" << endl;

  cout.imbue(old);
  cout << "A jolly time was had by all" << endl;

  // Create a vector of strings
  vector<string,allocator<void> > v;
  v.insert(v.begin(),"antelope");
  v.insert(v.begin(),"bison");
  v.insert(v.begin(),"elk");

  copy(v.begin(),v.end(),
       ostream_iterator<string,char,
                char_traits<char> >(cout," "));
  cout << endl;

  // Sort the strings using the locale as a comparitor
  sort(v.begin(),v.end(),loc);
  copy(v.begin(),v.end(),
       ostream_iterator<string,char,
               char_traits<char> >(cout," "));

  cout << endl;
  return 0;
}
```

See Also ***facets, has_facet, use_facet, specific facet reference sections***

messages, messages_byname

$$\begin{tikzcd}
 & & \textit{messages\_base} \\
\textit{messages\_bynam} \arrow[r] & \textit{messages} \arrow[ur] \arrow[dr] & \\
 & & \textit{locale::facet}
\end{tikzcd}$$

Summary

Messaging facets.

Synopsis

```
#include<locale>
class messages_base;

template <class charT> class messages;
```

Description

messages provides access to a localized messaging facility. The ***messages*** facet provides facilities based on the “C” locale, while the ***messages_byname*** facet provides the same facilities for named locales.

The ***messages_base*** class provides a catalog type for use by the derived ***messages*** and ***messages_byname*** classes.

Note that the default messages facet uses `catopen, catclose`, etc., to implement the message database. If your platform does not support these then you’ll need to imbue your own messages facet by implementing whatever database is available.

Interface

```
class messages_base {
public:
  typedef int catalog;
};

template <class charT>
class messages : public locale::facet, public messages_base {
public:
  typedef charT char_type;
  typedef basic_string<charT> string_type;
  explicit messages(size_t = 0);
  catalog open(const basic_string<char>&, const locale&) const;
  string_type  get(catalog, int, int,
                   const string_type&) const;
  void    close(catalog) const;
  static locale::id id;
protected:
  ~messages();  // virtual
  virtual catalog do_open(const basic_string<char>&,
                          const locale&) const;
  virtual string_type  do_get(catalog, int, int,
                              const string_type&) const;
  virtual void    do_close(catalog) const;
};

class messages_byname : public messages<charT> {
public:
  explicit messages_byname(const char*, size_t = 0);
protected:
  ~messages_byname();  // virtual
  virtual catalog do_open(const basic_string<char>&,
                          const locale&) const;
virtual string_type  do_get(catalog, int, int,
                            const string_type&) const;
  virtual void    do_close(catalog) const;
};
```

Types

`char_type`

Type of character the facet is instantiated on.

`string_type`

Type of character string returned by member functions.

Constructors and Destructors

`explicit` **`messages`**`(size_t refs = 0)`

Construct a ***messages*** facet. If the `refs` argument is 0 then destruction of the object is delegated to the locale, or locales, containing it. This allows the user to ignore lifetime management issues. On the other hand, if `refs` is 1 then the object must be explicitly deleted: the locale will not do so. In this case, the object can be maintained across the lifetime of multiple locales.

`explicit` **`messages_byname`**`(const char* name, size_t refs = 0);`

Construct a ***messages_byname*** facet. Use the named locale specified by the name argument. The `refs` argument serves the same purpose as it does for the ***messages*** constructor.

`~messages``();  // virtual and protected`

Destroy the facet

Facet ID

`static locale::id` **`id`**`;`

Unique identifier for this type of facet.

Public Member Functions

The public members of the ***messages*** facet provide an interface to protected members. Each public member `xxx` has a corresponding virtual protected member `do_xxx`. All work is delegated to these protected members. For instance, the long version of the public `open` function simply calls its protected cousin `do_open`.

```
void
close(catalog c) const;
```

```
string_type
get(catalog c, int set, int msgid,
    const string_type& dfault) const;
```

```
catalog
open(const basic_string<char>& fn, const locale&) const;
```

Each of these public member functions `xxx` simply call the corresponding protected `do_xxx` function.

Protected Member Functions

```
virtual void
do_close(catalog cat) const;
```

Closes the catalog. The `cat` argument must have been obtained by a call to `open()`.

```
virtual string_type
do_get(catalog cat, int set, int msgid,
       const string_type& dfault) const;
```

Retrieves a specific message. Returns the message identified by `cat`, `set`, `msgid`, and `dfault`. `cat` must have been obtained by a previous call to `open()` and this function must not be called with a cat that has had `close()` called on yet after the last call to `open()`. In other words, the catalog must have been opened and not yet closed.

```
virtual catalog
do_open(const basic_string<char>& name, const locale&) const;
```

Opens a message catalog. Returns a catalog identifier that can be passed to the `get()` function in order to access specific messages. Returns `-1` if the catalog name specified in the `name` argument is invalid. The `loc` argument will be used for codeset conversion if necessary.

Example

```
//
// messages.cpp
//
#include <string>
#include <iostream>

int main ()
{
  using namespace std;

  locale loc;

  // Get a reference to the messages<char> facet
  const messages<char>& mess =
#ifndef _RWSTD_NO_TEMPLATE_ON_RETURN_TYPE
    use_facet<messages<char> >(loc);
#else
    use_facet(loc,(messages<char>*)0);
#endif

  // Open a catalog and try to grab
  // both some valid messages, and an invalid message
  string def("Message Not Found");
  messages<char>::catalog cat =
          mess.open("./rwstdmessages.cat",loc);
  if (cat != -1)
  {
    string msg0 = mess.get(cat,1,1,def);
    string msg1 = mess.get(cat,1,2,def);
    string msg2 = mess.get(cat,1,6,def); // invalid msg #
    string msg3 = mess.get(cat,2,1,def);

    mess.close(cat);
    cout << msg0 << endl << msg1 << endl
         << msg2 << endl << msg3 << endl;
  }
  else
    cout << "Unable to open message catalog" << endl;

  return 0;
}
```

See Also ***locale, facets***

money_get → *locale::facet*

Summary

Monetary formatting facet for input.

Synopsis

```
#include <locale>
template <class charT, bool Intl = false,
          class InputIterator = istreambuf_iterator<charT> >
class money_get;
```

Description

The ***money_get*** facet interprets formatted monetary string values. The `Intl` template parameter is used to specify locale or international formatting. If `Intl` is true then international formatting is used, otherwise locale formatting is used.

Interface

```
template <class charT, bool Intl = false,
          class InputIterator = istreambuf_iterator<charT> >
class money_get : public locale::facet {
public:
  typedef charT              char_type;
  typedef InputIterator      iter_type;
  typedef basic_string<charT> string_type;
  explicit money_get(size_t = 0);
  iter_type get(iter_type, iter_type, ios_base&,
                ios_base::iostate&, long double&) const;
  iter_type get(iter_type, iter_type, ios_base&,
                ios_base::iostate&, string_type&) const;
  static const bool intl = Intl;
  static locale::id id;
protected:
  ~money_get();  // virtual
  virtual iter_type do_get(iter_type, iter_type, ios_base&,
                           ios_base::iostate&,
                           long double&) const;
  virtual iter_type do_get(iter_type, iter_type, ios_base&,
                           ios_base::iostate&,
                           string_type&) const;
};
```

Types

`char_type`

Type of character upon which the facet is instantiated.

`iter_type`

Type of iterator used to scan the character buffer.

`string_type`

Type of character string passed to member functions.

Constructors and Destructors

`explicit` **`money_get`**`(size_t refs = 0)`

Construct a ***money_get*** facet. If the `refs` argument is 0 then destruction of the object is delegated to the locale, or locales, containing it. This allows the user to ignore lifetime management issues. On the other had, if `refs` is 1 then the object must be explicitly deleted; the locale will not do so. In this case, the object can be maintained across the lifetime of multiple locales.

```
~money_get();  // virtual and protected
```
Destroy the facet

Static Members

```
static locale::id id;
```
Unique identifier for this type of facet.

```
static const bool intl = Intl;
```
`true` if international formatting is used, `false` otherwise.

Public Member Functions

The public members of the ***money_get*** facet provide an interface to protected members. Each public member `get` has a corresponding virtual protected member `do_get`.

```
iter_type
get(iter_type s, iter_type end, ios_base& f,
    ios_base::iostate& err, long double& units) const;
iter_type
get(iter_type s, iter_type end, ios_base& f,
    ios_base::iostate& err, string_type& digits) const;
```
Each of these two overloads of the public member function `get` calls the corresponding protected `do_get` function.

Protected Member Functions

```
virtual iter_type
do_get(iter_type s, iter_type end,
       ios_base& f,
       ios_base::iostate& err,
       long double& units) const;
virtual iter_type
do_get(iter_type s, iter_type end,
       ios_base& f,
       ios_base::iostate& err,
       string_type& digits) const;
```
Reads in a localized character representation of a monetary value and generates a generic representation, either as a sequence of digits or as a `long double` value. In either case `do_get` uses the smallest possible unit of currency.

Both overloads of `do_get` read characters from the range `[s,end)` until one of three things occurs:

- A monetary value is assembled
- An error occurs
- No more characters are available.

The functions use `f.flags()` and the `moneypunct<charT>` facet from `f.getloc()` for formatting information to use in interpreting the sequence of characters. `do_get` then places a pure sequence of digits representing the monetary value in the smallest possible unit of currency into the string argument `digits`, or it calculates a `long double` value based on those digits and returns that value in `units`.

The following specifics apply to formatting:

- Digit group separators are optional. If no grouping is specified then in thousands separator characters are treated as delimiters.
- If `space` or `none` are part of the format pattern in `moneypunct` then optional whitespace is consumed, except at the end. See the `moneypunct` reference section for a description of money specific formatting flags.
- If `iosbase::showbase` is set in `f.flags()` then the currency symbol is optional, and if it appears after all other elements then it will not be consumed. Otherwise the currency symbol is required, and will be consumed where ever it occurs.
- `digits` will be preceded by a `‘-’` , or `units` will be negated, if the monetary value is negative.
- See the `moneypunct` reference section for a description of money specific `formatting` flags.

The `err` argument will be set to `iosbase::failbit` if an error occurs during parsing.

Returns an iterator pointing one past the last character that is part of a valid monetary sequence.

Example

```
//
// moneyget.cpp
//

#include <string>
#include <sstream>

int main ()
{
  using namespace std;
  typedef istreambuf_iterator<char,char_traits<char> > iter_type;

  locale loc;
  string buffer("$100.02");
  string dest;
  long double ldest;
  ios_base::iostate state;
  iter_type end;

  // Get a money_get facet
  const money_get<char,false,iter_type>& mgf =
#ifndef _RWSTD_NO_TEMPLATE_ON_RETURN_TYPE
    use_facet<money_get<char,false,iter_type> >(loc);
#else
    use_facet(loc,(money_get<char,false,iter_type>*)0);
#endif

  {
    // Build an istringstream from the buffer and construct
    // a beginning iterator on it.
    istringstream ins(buffer);
    iter_type begin(ins);

    // Get a string representation of the monetary value
    mgf.get(begin,end,ins,state,dest);
  }
  {
    // Build another istringstream from the buffer, etc.
    // so we have an iterator pointing to the beginning
    istringstream ins(buffer);
    iter_type begin(ins);

    // Get a long double representation
    // of the monetary value
    mgf.get(begin,end,ins,state,ldest);
  }
  cout << buffer << " --> " << dest
       << " --> " << ldest << endl;

  return 0;
}
```

See Also ***locale, facets, money_put, moneypunct***

money_put

money_put → ***locale::facet***

Summary

Monetary formatting facet for output.

Synopsis

```
#include <locale>
template <class charT, bool Intl = false,
          class OutputIterator = ostreambuf_iterator<charT> >
class money_put;
```

Description

The ***money_put*** facet takes a `long double` value, or generic sequence of digits and writes out a formatted representation of the monetary value.

Interface

```
  template <class charT, bool Intl = false,
            class OutputIterator = ostreambuf_iterator<charT> >
  class money_put : public locale::facet {
  public:
    typedef charT              char_type;
    typedef OutputIterator     iter_type;
    typedef basic_string<charT> string_type;
    explicit money_put(size_t = 0);
    iter_type put(iter_type, ios_base&, char_type,
                  long double) const;
    iter_type put(iter_type, ios_base&, char_type,
                  const string_type&) const;
    static const bool intl = Intl;
    static locale::id id;
  protected:
    ~money_put();  // virtual
    virtual iter_type  do_put(iter_type, ios_base&, char_type,
                              long double) const;
    virtual iter_type do_put(iter_type, ios_base&, char_type,
                             const string_type&) const;
  };
```

Types

`char_type`

Type of the character upon which the facet is instantiated.

`iter_type`

Type of iterator used to scan the character buffer.

`string_type`

Type of character string passed to member functions.

Constructors and Destructors

`explicit` **`money_put`**`(size_t refs = 0)`

Construct a ***money_put*** facet. If the `refs` argument is 0 then destruction of the object is delegated to the locale, or locales, containing it. This allows the user to ignore lifetime management issues. On the other had, if `refs` is 1 then the object must be explicitly deleted; the locale will not do so. In this case, the object can be maintained across the lifetime of multiple locales.

```
~money_put();  // virtual and protected
```
Destroy the facet

```
static locale::id id;
```
Unique identifier for this type of facet.

Static Members

```
static const bool intl = Intl;
```
`true` if international representation, `false` otherwise.

Public Member Functions

The public members of the ***money_put*** facet provide an interface to protected members. Each public member `put` has a corresponding virtual protected member `do_put`.

```
iter_type
put(iter_type s, ios_base& f, char_type fill,
    long double units) const;
iter_type
put(iter_type s, ios_base& f, char_type fill,
    const string_type& digits) const;
```
Each of these two overloads of the public member function `put` simply calls the corresponding protected `do_put` function.

Protected Member Functions

```
virtual iter_type
do_put(iter_type s, ios_base& f, char_type fill,
       long double units) const;
```
Writes out a character string representation of the monetary value contained in `units`. Since `units` represents the monetary value in the smallest possible unit of currency any fractional portions of the value are ignored. `f.flags()` and the `moneypunct<charT>` facet from `f.getloc()` provide the formatting information.

The `fill` argument is used for any padding.

Returns an iterator pointing one past the last character written.

```
virtual iter_type
do_put(iter_type s, ios_base& f, char_type fill,
       const string_type& digits) const;
```
Writes out a character string representation of the monetary contained in `digits`. `digits` represents the monetary value as a sequence of digits in the smallest possible unit of currency. `do_put` only looks at an optional `-` character and any immediately contiguous digits. `f.flags()` and the `moneypunct<charT>` facet from `f.getloc()` provide the formatting information.

The `fill` argument is used for any padding.

Returns an iterator pointing one past the last character written.

Example

```
//
// moneyput.cpp
```

```
//

#include <string>
#include <iostream>

int main ()
{
  using namespace std;

  typedef ostreambuf_iterator<char,char_traits<char> > iter_type;

  locale loc;
  string buffer("10002");
  long double ldval = 10002;

  // Construct a ostreambuf_iterator on cout
  iter_type begin(cout);

  // Get a money put facet
  const money_put<char,false,iter_type>& mp =
#ifndef _RWSTD_NO_TEMPLATE_ON_RETURN_TYPE
  use_facet<money_put<char,false,iter_type> >(loc);
#else
  use_facet(loc,(money_put<char,false,iter_type>*)0);
#endif

  // Put out the string representation of the monetary value
  cout << buffer << " --> ";
  mp.put(begin,cout,' ',buffer);

  // Put out the long double representation
  // of the monetary value
  cout << endl << ldval << " --> ";
  mp.put(begin,cout,' ',ldval);

  cout <<  endl;

  return 0;
}
```

See Also ***locale, facets, money_get, moneypunct***

moneypunct, moneypunct_byname

$$\begin{tikzcd}
 & & \textit{moneypunct\_base} \\
\textit{moneypunct\_byname} \arrow[r] & \textit{moneypunct} \arrow[ur] \arrow[dr] & \\
 & & \textit{locale::facet}
\end{tikzcd}$$

Summary

Monetary punctuation facets.

Synopsis

```
#include <locale>
class money_base;

template <class charT, bool International = false>
class moneypunct;
```

Description

The ***moneypunct*** facets provide formatting specifications and punctuation character for monetary values. The ***moneypunct*** facet provides punctuation based on the "C" locale, while the ***moneypunct_byname*** facet provides the same facilities for named locales.

The facet is used by ***money_put*** for outputting formatted representations of monetary values and by ***money_get*** for reading these strings back in.

money_base provides a structure, `pattern`, that specifies the order of syntactic elements in a monetary value and enumeration values representing those elements. The `pattern` struct provides a simple array of characters, `field`. Each index in `field` is taken up by an enumeration value indicating the location of a syntactic element. The enumeration values are described below:

Format Flag	Meaning
`none`	No grouping seperator
`space`	Use space for grouping seperator
`symbol`	Currency symbol
`sign`	Sign of monetary value
`value`	The monetary value itself

The `do_pos_format` an `do_neg_format` member functions of ***moneypunct*** both return the `pattern` type. See the description of these functions for further elaboration.

Interface

```
class money_base {
public:
  enum part { none, space, symbol, sign, value };
  struct pattern { char field[4]; };
};
```

```
template <class charT, bool International = false>
class moneypunct : public locale::facet, public money_base {
public:
  typedef charT char_type;
  typedef basic_string<charT> string_type;
  explicit moneypunct(size_t = 0);
  charT        decimal_point() const;
  charT        thousands_sep() const;
  string       grouping()      const;
  string_type  curr_symbol()   const;
  string_type  positive_sign() const;
  string_type  negative_sign() const;
  int          frac_digits()   const;
  pattern      pos_format()    const;
  pattern      neg_format()    const;
  static locale::id id;
  static const bool intl = International;
protected:
  ~moneypunct();  // virtual
  virtual charT        do_decimal_point() const;
  virtual charT        do_thousands_sep() const;
  virtual string       do_grouping()      const;
  virtual string_type  do_curr_symbol()   const;
  virtual string_type  do_positive_sign() const;
  virtual string_type  do_negative_sign() const;
  virtual int          do_frac_digits()   const;
  virtual pattern      do_pos_format()    const;
  virtual pattern      do_neg_format()    const;
};

template <class charT, bool Intl = false>
class moneypunct_byname : public moneypunct<charT, Intl> {
public:
  explicit moneypunct_byname(const char*, size_t = 0);
protected:
  ~moneypunct_byname();  // virtual
  virtual charT        do_decimal_point() const;
  virtual charT        do_thousands_sep() const;
  virtual string       do_grouping()      const;
  virtual string_type  do_curr_symbol()   const;
  virtual string_type  do_positive_sign() const;
  virtual string_type  do_negative_sign() const;
  virtual int          do_frac_digits()   const;
  virtual pattern      do_pos_format()    const;
  virtual pattern      do_neg_format()    const;
};
```

Types

char_type

Type of character the facet is instantiated on.

string_type

Type of character string returned by member functions.

Constructors and Destructors

explicit **moneypunct**(size_t refs = 0)

Constructs a ***moneypunct*** facet. If the `refs` argument is 0 then destruction of the object is delegated to the locale, or locales, containing it.

This allows the user to ignore lifetime management issues. On the other had, if `refs` is 1 then the object must be explicitly deleted; the locale will not do so. In this case, the object can be maintained across the lifetime of multiple locales.

`explicit` **`moneypunct_byname`**`(const char* name, size_t refs = 0);`
Constructs a ***moneypunct_byname*** facet. Uses the named locale specified by the `name` argument. The `refs` argument serves the same purpose as it does for the ***moneypunct*** constructor.

`~moneypunct``();  // virtual and protected`
Destroy the facet

Static Members

`static locale::id` **`id`**`;`
Unique identifier for this type of facet.

`static const bool intl =` **`Intl`**`;`
`true` if international representation, `false` otherwise.

Public Member Functions

The public members of the ***moneypunct*** and ***moneypunct_byname*** facets provide an interface to protected members. Each public member `xxx` has a corresponding virtual protected member `do_xxx`. All work is delegated to these protected members. For instance, the long version of the public `decimal_point` function simply calls its protected cousin `do_decimal_point`.

```
string_type  curr_symbol()   const;
charT        decimal_point() const;
int          frac_digits()   const;
string       grouping()      const;
pattern      neg_format()    const;
string_type  negative_sign() const;
pattern      pos_format()    const;
string_type  positive_sign() const;
charT        thousands_sep() const;
```
Each of these public member functions `xxx` simply calls the corresponding protected `do_xxx` function.

Protected Member Functions

```
virtual string_type
do_curr_symbol()   const;
```
Returns a string to use as the currency symbol.

```
virtual charT
do_decimal_point() const;
```
Returns the radix separator to use if fractional digits are allowed (see `do_frac_digits`).

```
virtual int
do_frac_digits()   const;
```

Returns the number of digits in the fractional part of the monetary representation.

```
virtual string
do_grouping()      const;
```

Returns a string in which each character is used as an integer value to represent the number of digits in a particular grouping, starting with the rightmost group. A group is simply the digits between adjacent thousands' separators. Each group at a position larger than the size of the string gets the same value as the last element in the string. If a value is less than or equal to zero, or equal to `CHAR_MAX`, then the size of that group is unlimited. ***moneypunct*** returns an empty string, indicating no grouping.

```
virtual string_type
do_negative_sign() const;
```

A string to use as the negative sign. The first character of this string will be placed in the position indicated by the format pattern (see `do_neg_format`); the rest of the characters, if any, will be placed after all other parts of the monetary value.

```
virtual pattern
do_neg_format()    const;
virtual pattern
do_pos_format()    const;
```

Returns a pattern object specifying the location of the various syntactic elements in a monetary representation. The enumeration values `symbol`, `sign`, and `value` will appear exactly once in this pattern, with the remaining location taken by either `none` or `space`. `none` will never occupy the first position in the pattern and `space` will never occupy the first or the last position. Beyond these restrictions, elements may appear in any order. ***moneypunct*** returns {symbol, sign, none, value}.

```
virtual string_type
do_positive_sign() const;
```

A string to use as the positive sign. The first character of this string will be placed in the position indicated by the format pattern (see `do_pos_format`); the rest of the characters, if any, will be placed after all other parts of the monetary value.

```
virtual charT
do_thousands_sep() const;
```

Returns the grouping separator if grouping is allowed (see `do_grouping`).

Example

```
//
// moneypun.cpp
```

```
//

#include <string>

#include <iostream>

int main ()
{
  using namespace std;

  locale loc;

  // Get a moneypunct facet
  const moneypunct<char,false>& mp =
#ifndef _RWSTD_NO_TEMPLATE_ON_RETURN_TYPE
  use_facet<moneypunct<char,false> >(loc);
#else
  use_facet(loc,(moneypunct<char,false>*)0);
#endif

  cout << "Decimal point        = "
       << mp.decimal_point() << endl;
  cout << "Thousands separator  = "
       << mp.thousands_sep() << endl;
  cout << "Currency symbol      = "
       << mp.curr_symbol() << endl;
  cout << "Negative Sign        = "
       << mp.negative_sign() << endl;
  cout << "Digits after decimal = "
       << mp.frac_digits() << endl;

  return 0;
}
```

See Also ***locale, facets, money_put, money_get***

num_get → ***locale::facet***

Summary Numeric formatting facet for input.

Synopsis

```
#include <locale>
template <class charT, class InputIterator > class num_get;
```

Description The ***num_get*** provides facilities for formatted input of numbers. ***basic_istream*** and all other input-oriented streams use this facet to implement formatted numeric input.

Interface

```
template <class charT, class InputIterator =
istreambuf_iterator<charT> >
class num_get : public locale::facet {
public:
  typedef charT            char_type;
  typedef InputIterator    iter_type;
  explicit num_get(size_t refs = 0);
  iter_type get(iter_type, iter_type, ios_base&,
                ios_base::iostate&, bool&)          const;
  iter_type get(iter_type, iter_type, ios_base& ,
                ios_base::iostate&, long&)          const;
  iter_type get(iter_type, iter_type, ios_base&,
                ios_base::iostate&, unsigned short&) const;
  iter_type get(iter_type, iter_type, ios_base&,
                ios_base::iostate&, unsigned int&)  const;
  iter_type get(iter_type, iter_type, ios_base&,
                ios_base::iostate&, unsigned long&) const;
  iter_type get(iter_type, iter_type, ios_base&,
                ios_base::iostate&, float&)         const;
  iter_type get(iter_type, iter_type, ios_base&,
                ios_base::iostate&, double&)        const;
  iter_type get(iter_type, iter_type, ios_base&,
                ios_base::iostate&, long double&)   const;
  static locale::id id;

protected:
  ~num_get();  // virtual
  virtual iter_type do_get(iter_type, iter_type, ios_base&,
                           ios_base::iostate&, bool&) const;
  virtual iter_type do_get(iter_type, iter_type, ios_base&,
                           ios_base::iostate&, long&) const;
  virtual iter_type do_get(iter_type, iter_type, ios_base&,
                           ios_base::iostate&,
                           unsigned short&) const;
  virtual iter_type do_get(iter_type, iter_type, ios_base&,
                           ios_base::iostate&,
                           unsigned int&) const;
  virtual iter_type do_get(iter_type, iter_type, ios_base&,
                           ios_base::iostate&,
                           unsigned long&) const;
  virtual iter_type do_get(iter_type, iter_type, ios_base&,
                                ios_base::iostate&, float&) const;
    virtual iter_type do_get(iter_type, iter_type, ios_base&,
                                ios_base::iostate&, double&) const;
    virtual iter_type do_get(iter_type, iter_type, ios_base&,
                                ios_base::iostate&,
                                long double&) const;
  };
```

Types

`char_type`

Type of character the facet is instantiated on.

`iter_type`

Type of iterator used to scan the character buffer.

Constructor and Destructor

`explicit` **`num_get`**`(size_t refs = 0)`

Construct a ***num_get*** facet. If the `refs` argument is 0 then destruction of the object is delegated to the locale, or locales, containing it. This allows the user to ignore lifetime management issues. On the other had, if `refs` is 1 then the object must be explicitly deleted; the locale will not do so. In this case, the object can be maintained across the lifetime of multiple locales.

`~`**`num_get`**`();  // virtual and protected`

Destroy the facet

Facet ID

`static locale::id` **`id`**`;`

Unique identifier for this type of facet.

Public Member Functions

The public members of the ***num_get*** facet provide an interface to protected members. Each public member `xxx` has a corresponding virtual protected member `do_xxx`. All work is delegated to these protected members. For instance, the long version of the public `get` function simply calls its protected cousin `do_get`.

```
iter_type
get(iter_type in, iter_type end, ios_base& io,
    ios_base::iostate& err, bool& v)          const;
iter_type
get(iter_type in, iter_type end, ios_base& io,
    ios_base::iostate& err, long& v)          const;
iter_type
get(iter_type in, iter_type end, ios_base& io,
    ios_base::iostate& err, unsigned short& v) const;
iter_type
get(iter_type in, iter_type end, ios_base& io,
    ios_base::iostate& err, unsigned int& v)  const;
iter_type
get(iter_type in, iter_type end, ios_base& io,
    ios_base::iostate& err, unsigned long& v) const;
```

```
iter_type
get(iter_type in, iter_type end, ios_base& io,
    ios_base::iostate& err, float& v)         const;
iter_type
get(iter_type in, iter_type end, ios_base& io,
    ios_base::iostate& err, double& v)        const;
iter_type
get(iter_type in, iter_type end, ios_base& io,
    ios_base::iostate& err, long double& v)   const;
```

Each of the eight overloads of the `get` function simply call the corresponding `do_get` function.

Protected Member Functions

```
virtual iter_type
do_get(iter_type in, iter_type end, ios_base& io,
       ios_base::iostate& err, bool& v) const;
virtual iter_type
do_get(iter_type in, iter_type end, ios_base& io,
       ios_base::iostate& err, long& v) const;
virtual iter_type
do_get(iter_type in, iter_type end, ios_base& io,
       ios_base::iostate& err,
       unsigned short& v) const;
virtual iter_type
do_get(iter_type in, iter_type end, ios_base& io,
       ios_base::iostate& err,
       unsigned int& v) const;
virtual iter_type
do_get(iter_type in, iter_type end, ios_base& io,
       ios_base::iostate& err,
       unsigned long& v) const;
virtual iter_type
do_get(iter_type in, iter_type end, ios_base& io,
       ios_base::iostate& err, float& v) const;
virtual iter_type
do_get(iter_type in, iter_type end, ios_base& io,
       ios_base::iostate& err, double& v) const;
virtual iter_type
do_get(iter_type in, iter_type end, ios_base& io,
       ios_base::iostate& long double& v) const;
```

The eight overloads of the `do_get` member function all take a sequence of characters `[int,end)`, and extract a numeric value. The numeric value is returned in `v`. The `io` argument is used to obtain formatting information and the `err` argument is used to set error conditions in a calling stream.

Example

```
//
// numget.cpp
//

#include <sstream>

int main ()
{
  using namespace std;

 typedef istreambuf_iterator<char,char_traits<char> > iter_type;
```

```
  locale loc;
  ios_base::iostate state;
  bool bval = false;
  long lval = 0L;
  long double ldval = 0.0;
  iter_type end;

  // Get a num_get facet
  const num_get<char,iter_type>& tg =
#ifndef _RWSTD_NO_TEMPLATE_ON_RETURN_TYPE
  use_facet<num_get<char,iter_type> >(loc);
#else
  use_facet(loc,(num_get<char,iter_type>*)0);
#endif

  {
    // Build an istringstream from the buffer and construct
    // beginning and ending iterators on it.
    istringstream ins("true");
    iter_type begin(ins);

    // Get a bool value
    tg.get(begin,end,ins,state,bval);
  }
  cout << bval << endl;
  {
    // Get a long value
    istringstream ins("2422235");
    iter_type begin(ins);
    tg.get(begin,end,ins,state,lval);
  }
  cout << lval << endl;
  {
    // Get a long double value
    istringstream ins("32324342.98908");
    iter_type begin(ins);
    tg.get(begin,end,ins,state,ldval);
  }
  cout << ldval << endl;
  return 0;
}
```

See Also ***locale, facets, num_put, numpunct, ctype***

numpunct, numpunct_byname

numpunct → ***locale::facet***

Summary

Numeric punctuation facet.

Synopsis

```
#include <locale>
template <class charT>  class numpunct;
template <class charT>  class numpunct_byname;
```

Description

The ***numpunct<charT>*** facet specifies numeric punctuation. This template provides punctuation based on the "C" locale, while the ***numpunct_byname*** facet provides the same facilities for named locales.

Both ***num_put*** and ***num_get*** make use of this facet.

Interface

```
template <class charT>
class numpunct : public locale::facet {
public:
  typedef charT               char_type;
  typedef basic_string<charT> string_type;
  explicit numpunct(size_t refs = 0);
  char_type    decimal_point()   const;
  char_type    thousands_sep()   const;
  string       grouping()        const;
  string_type  truename()        const;
  string_type  falsename()       const;
  static locale::id id;
protected:
  ~numpunct();  // virtual
  virtual char_type    do_decimal_point() const;
  virtual char_type    do_thousands_sep() const;
  virtual string       do_grouping()      const;
  virtual string_type  do_truename()      const;  // for bool
  virtual string_type  do_falsename()     const;  // for bool
};

template <class charT>
class numpunct_byname : public numpunct<charT> {
public:
  explicit numpunct_byname(const char*, size_t refs = 0);
protected:
  ~numpunct_byname();  // virtual
  virtual char_type    do_decimal_point() const;
  virtual char_type    do_thousands_sep() const;
  virtual string       do_grouping()      const;
  virtual string_type  do_truename()      const;  // for bool
  virtual string_type  do_falsename()     const;  // for bool
};
```

Types

`char_type`

Type of character upon which the facet is instantiated.

string_type
Type of character string returned by member functions.

Constructors and Destructors

`explicit` **`numpunct`**`(size_t refs = 0)`
Construct a ***numpunct*** facet. If the `refs` argument is 0 then destruction of the object is delegated to the locale, or locales, containing it. This allows the user to ignore lifetime management issues. On the other had, if `refs` is 1 then the object must be explicitly deleted; the locale will not do so. In this case, the object can be maintained across the lifetime of multiple locales.

`explicit` **`numpunct_byname`**`(const char* name, size_t refs = 0);`
Construct a ***numpunct_byname*** facet. Use the named locale specified by the name argument. The `refs` argument serves the same purpose as it does for the ***numpunct*** constructor.

```
~numpunct();  // virtual and protected
~numpunct_byname();  // virtual and protected
```
Destroy the facet

Facet ID

`static locale::id` **`id`**`;`
Unique identifier for this type of facet.

Public Member Functions

The public members of the ***numpunct*** facet provide an interface to protected members. Each public member `xxx` has a corresponding virtual protected member `do_xxx`. All work is delegated to these protected members. For instance, the long version of the public `grouping` function simply calls its protected cousin `do_grouping`.

```
char_type    decimal_point()   const;
string_type  falsename()       const;
string       grouping()        const;
char_type    thousands_sep()   const;
string_type  truename()        const;
```

Each of these public member functions `xxx` simply call the corresponding protected `do_xxx` function.

Protected Member Functions

```
virtual char_type
do_decimal_point() const;
```
Returns the decimal radix separator . ***numpunct*** returns '.'.

```
virtual string_type
do_falsename()     const;  // for bool
virtual string_type
do_truename()      const;  // for bool
```
Returns a string representing the name of the boolean values true and false respectively . ***numpunct*** returns "true" and "false".

```
virtual string
do_grouping()     const;
```

Returns a string in which each character is used as an integer value to represent the number of digits in a particular grouping, starting with the rightmost group. A group is simply the digits between adjacent thousands separators. Each group at a position larger than the size of the string gets the same value as the last element in the string. If a value is less than or equal to zero, or equal to `CHAR_MAX` then the size of that group is unlimited. ***numpunct*** returns an empty string, indicating no grouping.

```
virtual char_type
do_thousands_sep() const;
```

Returns the decimal digit group separator. ***numpunct*** returns ‘,’.

Example

```
//
// numpunct.cpp
//

#include <iostream>

int main ()
{
  using namespace std;
  locale loc;

  // Get a numpunct facet
  const numpunct<char>& np =
#ifndef _RWSTD_NO_TEMPLATE_ON_RETURN_TYPE
  use_facet<numpunct<char> >(loc);
#else
  use_facet(loc,(numpunct<char>*)0);
#endif

  cout << "Decimal point      = "
       << np.decimal_point() << endl;
  cout << "Thousands separator = "
       << np.thousands_sep() << endl;
  cout << "True name           = "
       << np.truename() << endl;
  cout << "False name          = "
       << np.falsename() << endl;

  return 0;
}
```

See Also ***locale, facets, num_put, num_get, ctype***

num_put → ***locale::facet***

Summary Numeric formatting facet for output.

Synopsis

```
#include <locale>
template <class charT, class OutputIterator> class num_put;
```

Description The ***num_put<charT,OutputIterator>*** template provides facilities for formatted output of numbers. ***basic_ostream*** and all other input oriented streams use this facet to implement formatted numeric output.

Interface

```
template <class charT, class OutputIterator =
ostreambuf_iterator<charT> >
class num_put : public locale::facet {
public:
  typedef charT            char_type;
  typedef OutputIterator   iter_type;
  explicit num_put(size_t = 0);

  iter_type put(iter_type, ios_base&, char_type, bool) const;
  iter_type put(iter_type, ios_base&, char_type, long) const;
  iter_type put(iter_type, ios_base&, char_type,
              unsigned long) const;
  iter_type put(iter_type, ios_base&, char_type,
              double) const;
  iter_type put(iter_type, ios_base&, char_type,
              long double) const;
  static locale::id id;

protected:
  ~num_put();  // virtual
  virtual iter_type do_put(iter_type, ios_base&, char_type,
                         bool) const;
  virtual iter_type do_put(iter_type, ios_base&, char_type,
                         long) const;
  virtual iter_type do_put(iter_type, ios_base&, char_type,
                         unsigned long) const;
  virtual iter_type do_put(iter_type, ios_base&, char_type,
                         double) const;
  virtual iter_type do_put(iter_type, ios_base&, char_type,
                         long double) const;
};
```

Types

`char_type`
Type of character upon which the facet is instantiated.

`iter_type`
Type of iterator used to scan the character buffer.

Constructors and Destructors

```
explicit num_put(size_t refs = 0)
```

Construct a ***num_put*** facet. If the `refs` argument is 0 then destruction of the object is delegated to the locale, or locales, containing it. This allows the user to ignore lifetime management issues. On the other had, if `refs` is 1 then the object must be explicitly deleted; the locale will not do so. In this case, the object can be maintained across the lifetime of multiple locales.

```
~num_put();  // virtual and protected
```

Destroy the facet

Facet ID

```
static locale::id id;
```

Unique identifier for this type of facet.

Public Member Functions

The public members of the ***num_put*** facet provide an interface to protected members. Each public member `xxx` has a corresponding virtual protected member `do_xxx`. All work is delegated to these protected members. For instance, the long version of the public `put` function simply calls its protected cousin `do_put`.

```
iter_type
put(iter_type s, ios_base& io, char_type fill, bool v) const;
iter_type
put(iter_type s, ios_base& io, char_type fill, long v) const;
iter_type
put(iter_type s, ios_base& io, char_type fill,
    unsigned long v) const;
iter_type
put(iter_type s, ios_base& io, char_type fill, double v) const;
iter_type
put(iter_type s, ios_base& io, char_type fill, long double v) const;
```

Each of the five overloads of the `put` function simply call the corresponding `do_put` function.

Protected Member Functions

```
virtual iter_type
do_put(iter_type s, ios_base& io,
       char_type fill, bool v) const;
virtual iter_type
do_put(iter_type s, ios_base& io,
       char_type fill, long v) const;
virtual iter_type
do_put(iter_type s, ios_base& io,
       char_type fill,unsigned long) const;
virtual iter_type
do_put(iter_type s, ios_base& io,
       char_type fill, double v) const;
virtual iter_type
do_put(iter_type s, ios_base& io,
       char_type fill,long double v) const;
```

The five overloads of the `do_put` member function all take a numeric value and output a formatted character string representing that value. The character string is output through the `s` argument to the function. The `io` argument is used to obtain formatting specifications, and the `fill` argument determines the character to use in padding.

Example

```
//
// numput.cpp
//

#include <iostream>

int main ()
{
  using namespace std;

  typedef ostreambuf_iterator<char,char_traits<char> > iter_type;

  locale loc;
  bool bval = true;
  long lval = 422432L;
  unsigned long ulval = 12328889UL;
  double dval = 10933.8934;
  long double ldval = 100028933.8934;

  // Construct a ostreambuf_iterator on cout
  iter_type begin(cout);

  // Get a num_put facet reference
  const num_put<char,iter_type>& np =
#ifndef _RWSTD_NO_TEMPLATE_ON_RETURN_TYPE
  use_facet<num_put<char,iter_type> >(loc);
#else
  use_facet(loc,(num_put<char,iter_type>*)0);
#endif

  // Put out a bool
  cout << bval << " --> ";
  np.put(begin,cout,' ',bval);

  // Put out a long
  cout << endl << lval << " --> ";
  np.put(begin,cout,' ',lval);

  // Put out an unsigned long
  cout << endl << ulval << " --> ";
  np.put(begin,cout,' ',ulval);

  // Put out a double
  cout << endl << dval << " --> ";
  np.put(begin,cout,' ',dval);

  // Put out a long double
  cout << endl << ldval << " --> ";
  np.put(begin,cout,' ',ldval);

  cout <<  endl;

  return 0;
}
```

See Also ***locale, facets, numget, numpunct, ctype***

ostreambuf_iterator

ostreambuf_iterator → ***output_iterator***

Synopsis

```
#include <streambuf>
template<class charT, class traits = char_traits<charT> >
class ostreambuf_iterator
: public output_iterator
```

Description

The template class ***ostreambuf_iterator*** writes successive characters onto the stream buffer object from which it was constructed. The `operator=` is used to write the characters and in case of failure the member function `failed()` returns true.

Interface

```
template<class charT, class traits = char_traits<charT> >
class ostreambuf_iterator
: public output_iterator {

 public:

  typedef charT                          char_type;
  typedef traits                         traits_type;
  typedef basic_streambuf<charT, traits> streambuf_type;
  typedef basic_ostream<charT, traits>   ostream_type;

  ostreambuf_iterator(ostream_type& s) throw();

  ostreambuf_iterator(streambuf_type *s) throw();

  ostreambuf_iterator& operator*();
  ostreambuf_iterator& operator++();
  ostreambuf_iterator operator++(int);

  ostreambuf_iterator& operator=(charT c);

  bool failed( ) const throw();

};
```

Types

`char_type`

The type `char_type` is a synonym for the template parameter `charT`.

`ostream_type`

The type `ostream_type` is an instantiation of class `basic_ostream` on types `charT` and `traits`:

```
typedef basic_ostream<charT, traits>   ostream_type;
```

`streambuf_type`

The type `streambuf_type` is an instantiation of class `basic_streambuf` on types `charT` and `traits`:

```
typedef basic_streambuf<charT, traits> streambuf_type;
```

traits_type
The type traits_type is a synonym for the template parameter traits.

Constructors

ostreambuf_iterator(ostream_type& s) throw();
Constructs an ostreambuf_iterator that uses the basic_streambuf object pointed at by s.rdbuf()to output characters. If s.rdbuf() is a null pointer, calls to the member function failed() return true.

ostreambuf_iterator(streambuf_type *s) throw();
Constructs an ostreambuf_iterator that uses the basic_streambuf object pointed at by s to output characters. If s is a null pointer, calls to the member function failed() return true.

Member Operators

ostreambuf_iterator&
operator=(charT c);
Inserts the character c into the output sequence of the attached stream buffer. If the operation fails, calls to the member function failed() return true.

ostreambuf_iterator&
operator++();
Returns *this.

ostreambuf_iterator
operator++(int);
Returns *this.

ostreambuf_iterator
operator*();
Returns *this.

Public Member Function

bool
failed() const
throw();
Returns true if the iterator failed inserting a characters or false otherwise.

Examples

```
//
// stdlib/examples/manual/ostreambuf_iterator.cpp
//
#include<iostream>
#include<fstream>

void main ( )
{
  using namespace std;

  // create a filebuf object
  filebuf  buf;

  // open the file iter_out and link it to the filebuf object
  buf.open("iter_out", ios_base::in | ios_base::out );

  // create an ostreambuf_iterator and link it to
  // the filebuf object
  ostreambuf_iterator<char> out_iter(&buf);

  // output into the file using the ostreambuf_iterator
  for(char i=64; i<128; i++ )
   out_iter = i;

  // seek to the beginning of the file
  buf.pubseekpos(0);

  // create an istreambuf_iterator and link it to
  // the filebuf object
  istreambuf_iterator<char> in_iter(&buf);

  // construct an end of stream iterator
  istreambuf_iterator<char> end_of_stream_iterator;

  cout << endl;

  // output the content of the file
  while( !in_iter.equal(end_of_stream_iterator) )

  // use both operator++ and operator*
  cout << *in_iter++;

  cout << endl;

}
```

See Also

basic_streambuf(3C++), ***basic_ostream***(3C++), ***istreambuf_iterator***(3C++)

Working Paper for Draft Proposed International Standard for Information Systems--Programming Language C++, Section 24.4.4

Standards Conformance

ANSI X3J16/ISO WG21 Joint C++ Committee

ostrstream → ***basic_ostream*** → ***basic_ios*** → ***ios_base***

Synopsis

```
#include <strstream>
class ostrstream
: public basic_ostream<char>
```

Description

The class ***ostrstream*** provides functionality to write to an array in memory. It uses a private ***strstreambuf*** object to control the associated array object. It inherits from ***basic_ostream<char>*** and therefore can use all the formatted and unformatted output functions.

Interface

```
class ostrstream
: public basic_ostream<char> {

 public:

  typedef char_traits<char>             traits;

  typedef char                         char_type;
  typedef typename traits::int_type    int_type;
  typedef typename traits::pos_type    pos_type;
  typedef typename traits::off_type    off_type;

  ostrstream();
  ostrstream(char *s, int n,
             ios_base::openmode = ios_base::out);

  virtual ~ostrstream();

  strstreambuf *rdbuf() const;

  void freeze(int freezefl = 1);

  char *str();

  int pcount() const;

};
```

Types

`char_type`
The type `char_type` is a synonym of type `char`.

`int_type`
The type `int_type` is a synonym of type `traits::in_type`.

`off_type`
The type `off_type` is a synonym of type `traits::off_type`.

`pos_type`

The type `pos_type` is a synonym of type `traits::pos_type`.

`traits`

The type `traits` is a synonym of type `char_traits<char>`.

Constructors

`ostrstream();`

Constructs an object of class `ostrstream`, initializing the base class `basic_ostream<char>` with the associated `strstreambuf` object. The `strstreambuf` object is initialized by calling its default constructor `strstreambuf()`.

```
ostrstream(char* s,int n, ios_base::openmode
           mode = ios_base::out);
```

Constructs an object of class `ostrstream`, initializing the base class `basic_ostream<char>` with the associated `strstreambuf` object. The `strstreambuf` object is initialized by calling one of two constructors:

- if `mode & app == 0` calls `strstreambuf(s,n,s)`
- Otherwise calls `strstreambuf(s,n,s + ::strlen(s))`

Destructor

`virtual ~ostrstream();`

Destroys an object of class `ostrstream`.

Member Functions

```
void
freeze(bool freezefl = 1);
```

If the mode is dynamic, alters the freeze status of the dynamic array object as follows:

- If `freezefl` is false, the function sets the freeze status to frozen.
- Otherwise, it clears the freeze status.

```
int
pcount() const;
```

Returns the size of the output sequence.

```
strstreambuf*
rdbuf() const;
```

Returns a pointer to the private `strstreambuf` object associated with the stream.

```
char*
str();
```

Returns a pointer to the underlying array object which may be null.

Examples

See ***strstream, istrstream*** and ***strstreambuf*** examples.

See Also

char_traits(3C++), ***ios_base***(3C++), ***basic_ios***(3C++), ***strstreambuf***(3C++), ***istrstream***(3C++), ***strstream***(3C++)

Working Paper for Draft Proposed International Standard for Information Systems--Programming Language C++, Annex D Compatibility features Section D.6.3

Standards Conformance

ANSI X3J16/ISO WG21 Joint C++ Committee

Synopsis

```
#include <iomanip>
template<class T> class smanip;
template<class T, class traits> class smanip_fill;
```

Description

The template classes ***smanip*** and ***smanip_fill*** are helper classes used to implement parameterized manipulators. The class ***smanip*** is used as the return type for manipulators that do not need to carry information about the character type of the stream they are applied to. This is the case for `resetiosflags`, `setiosflags`, `setbase`, `setprecision` and `setw.` The class ***smanip_fill*** is used as the return type for manipulators that do need to carry information about the character type of the stream they are applied to. This is the case for `setfill.`

interface

```
template<class T>
class smanip {

 public:

  smanip(ios_base& (*pf) (ios_base&, T), T manarg);

};

template<class T, class traits>
class smanip_fill {

 public:

  smanip_fill(basic_ios<T, traits>& (*pf) (basic_ios<T, traits>&,
              T), T manarg);

};

// parameterized manipulators

smanip<ios_base::fmtflags> resetiosflag(ios_base::fmtflags mask);
smanip<ios_base::fmtflags> setiosflag(ios_base::fmtflags mask);
smanip<int>                setbase(int base);
smanip<int>                setprecision(int n);
smanip<int>                setw(int n);

template <class charT>
smanip_fill<charT, char_traits<charT> > setfill(charT c);

// overloaded extractors

template <class charT, class traits, class T>
basic_istream<charT,traits>&
operator>>(basic_istream<charT,traits>& is, const smanip<T>& a);
```

```
template <class charT, class traits>
basic_istream<charT,traits>&
operator>>(basic_istream<charT,traits>& is,
const smanip_fill<charT,char_traits<charT> >& a);

// overloaded insertors

template <class charT, class traits, class T>
basic_ostream<charT,traits>&
operator<<(basic_ostream<charT,traits>& is, const smanip<T>& a);

template <class charT, class traits>
basic_ostream<charT,traits>&
operator>>(basic_ostream<charT,traits>& is,
const smanip_fill<charT,char_traits<charT> >& a);
```

Class smanip Constructor

`smanip``(ios_base& (*pf) (ios_base&, T), T manarg);`

Constructs an object of class `smanip` that stores a function pointer `pf,` that will be called with argument `manarg,` in order to perform the manipulator task. The call to `pf` is performed in the insertor or extractor overloaded on type `smanip`.

Class smanip_fill Constructor

```
smanip_fill(basic_ios<T, traits>& (*pf) (basic_ios<T, traits>&,
            T), T manarg);
```

Constructs an object of class `smanip_fill` that stores a function pointer `pf,` that will be called with argument `manarg,` in order to perform the manipulator task. The call to `pf` is performed in the insertor or extractor overloaded on type `smanip_fill`.

Manipulators

```
smanip<ios_base::fmtflags>
resetiosflag(ios_base::fmtflags mask);
```

Resets the `ios_base::fmtflags` designated by `mask` in the stream to which it is applied.

```
smanip<int>
setbase(int base);
```

Sets the base for the output or input of integer values in the stream to which it is applied. The valid values for `mask` are 8, 10, 16.

```
template <class charT>
smanip_fill<charT, char_traits<charT> >
setfill(charT c);
```

Sets the fill character in the stream it is applied to.

```
smanip<ios_base::fmtflags>
setiosflag(ios_base::fmtflags mask);
```

Sets the `ios_base::fmtflags` designated by `mask` in the stream it is applied to.

```
smanip<int>
setprecision(int n);
```

Sets the precision for the output of floating point values in the stream to which it is applied.

```
smanip<int>
setw(int n);
```

Set the field width in the stream to which it is applied.

Extractors

```
template <class charT, class traits, class T>
basic_istream<charT,traits>&
operator>>(basic_istream<charT,traits>& is, const smanip<T>& a);
```

Applies the function stored in the parameter of type `smanip<T>,` on the stream `is`.

```
template <class charT, class traits>
basic_istream<charT,traits>&
operator>>(basic_istream<charT,traits>& is,
           const smanip_fill<charT,char_traits<charT> >& a);
```

Applies the function stored in the parameter of type `smanip_fill<charT, char_traits<charT> >,` on the stream `is`.

Insertors

```
template <class charT, class traits, class T>
basic_ostream<charT,traits>&
operator<<(basic_ostream<charT,traits>& os, const smanip<T>& a);
```

Applies the function stored in the parameter of type `smanip<T>,` on the stream `os`.

```
template <class charT, class traits>
basic_ostream<charT,traits>&
operator<<(basic_ostream<charT,traits>& os,
           const smanip_fill<charT,char_traits<charT> >& a);
```

Applies the function stored in the parameter of type `smanip_fill<charT, char_traits<charT> >,` on the stream `os`.

See Also

ios_base(3C++), ***basic_ios***(3C++), ***basic_istream***(3C++), ***basic_ostream***(3C++)

Working Paper for Draft Proposed International Standard for Information Systems--Programming Language C++, Section 27.6.3

Standards Conformance

ANSI X3J16/ISO WG21 Joint C++ Committee

strstream

$$\begin{tikzcd}
 & & \textit{basic\_ostream} \arrow[dr] & & \\
\textit{strstream} \arrow[r] & \textit{basic\_iostream} \arrow[ur] \arrow[dr] & & \textit{basic\_ios} \arrow[r] & \textit{ios\_base} \\
 & & \textit{basic\_istream} \arrow[ur] & &
\end{tikzcd}$$

Synopsis

```
#include <strstream>
class strstream
: public basic_iostream<char>
```

Description

The class ***strstream*** provides functionality to read and write to an array in memory. It uses a private ***strstreambuf*** object to control the associated array. It inherits from ***basic_iostream<char>*** and therefore can use all the formatted and unformatted output and input functions.

Interface

```
class strstream
: public basic_iostream<char> {

 public:

  typedef char_traits<char>          traits;

  typedef char                      char_type;
  typedef typename traits::int_type  int_type;
  typedef typename traits::pos_type  pos_type;
  typedef typename traits::off_type  off_type;

  strstream();
  strstream(char *s, int n,
           ios_base::openmode = ios_base::out | ios_base::in);

  void freeze(int freezefl = 1);

  int pcount() const;

  virtual ~strstream();

  strstreambuf *rdbuf() const;

  char *str();

};
```

Types

char_type

The type `char_type` is a synonym of type `char`.

int_type

The type `int_type` is a synonym of type `traits::in_type`.

`off_type`
The type `off_type` is a synonym of type `traits::off_type`.

`pos_type`
The type `pos_type` is a synonym of type `traits::pos_type`.

`traits`
The type `traits` is a synonym of type `char_traits<char>`.

Constructors

```
strstream();
```
Constructs an object of class `strstream`, initializing the base class `basic_iostream<char>` with the associated `strstreambuf` object. The `strstreambuf` object is initialized by calling its default constructor `strstreambuf()`.

```
strstream(char* s,int n, ios_base::openmode
          mode = ios_base::out | ios_base::in);
```
Constructs an object of class `strstream`, initializing the base class `basic_iostream<char>` with the associated `strstreambuf` object. The `strstreambuf` object is initialized by calling one of two constructors:

- if `mode & app == 0` calls `strstreambuf(s,n,s)`
- Otherwise calls `strstreambuf(s,n,s + ::strlen(s))`

Destructors

```
virtual ~strstream();
```
Destroys an object of class `strstream`.

Member Functions

```
void
freeze(int freezefl = 1);
```
If the mode is dynamic, alters the freeze status of the dynamic array object as follows:

- If `freezefl` is false, the function sets the freeze status to `frozen`.
- Otherwise, it clears the freeze status.

```
int
pcount() const;
```
Returns the size of the output sequence.

```
strstreambuf*
rdbuf() const;
```
Returns a pointer to the `strstreambuf` object associated with the stream.

```
char*
str();
```
Returns a pointer to the underlying array object which may be null.

Examples

```
//
// stdlib/examples/manual/strstream.cpp
//
#include<strstream>
```

```
void main ( )
{
  using namespace std;

  // create a bi-directional strstream object
  strstream inout;

  // output characters
  inout << "Das ist die rede von einem man" << endl;
  inout << "C'est l'histoire d'un home" << endl;
  inout << "This is the story of a man" << endl;

  char p[100];

  // extract the first line
  inout.getline(p,100);

  // output the first line to stdout
  cout << endl << "Deutch :" << endl;
  cout << p;

  // extract the second line
  inout.getline(p,100);

  // output the second line to stdout
  cout << endl << "Francais :" << endl;
  cout << p;

  // extract the third line
  inout.getline(p,100);

  // output the third line to stdout
  cout << endl << "English :" << endl;
  cout << p;

  // output the all content of the
  // strstream object to stdout
  cout << endl << endl << inout.str();

}
```

See Also

char_traits(3C++), ***ios_base***(3C++), ***basic_ios***(3C++), ***strstreambuf***(3C++), ***istrstream***(3C++), ***ostrstream***(3c++)

Working Paper for Draft Proposed International Standard for Information Systems--Programming Language C++, Annex D Compatibility features Section D.6.4

Standards Conformance

ANSI X3J16/ISO WG21 Joint C++ Committee

strstreambuf → ***basic_streambuf***

Synopsis

```
#include <strstream>
class strstreambuf
: public basic_streambuf<char>
```

Description

The class ***strstreambuf*** is derived from ***basic_streambuf*** specialized on type `char` to associate possibly the input sequence and possibly the output sequence with a tiny character array, whose elements store arbitrary values.

Each object of type ***strstreambuf*** controls two character sequences:

- A character input sequence;
- A character output sequence.

Note: see ***basic_streambuf.***

The two sequences are related to each other, but are manipulated separately. This means that you can read and write characters at different positions in objects of type ***strstreambuf*** without any conflict (in opposition to the ***basic_filebuf*** objects).

The underlying array has several attributes:

- **allocated**, set when a dynamic array has been allocated, and hence should be freed by the destructor of the ***strstreambuf*** object.
- **constant**, set when the array has const elements, so the output sequence cannot be written.
- **dynamic**, set when the array object is allocated (or reallocated) as necessary to hold a character sequence that can change in length.
- **frozen**, set when the program has requested that the array will not be altered, reallocated, or freed.

Interface

```
class strstreambuf
: public basic_streambuf<char> {

 public:

  typedef char_traits<char>          traits;
  typedef basic_ios<char, traits>    ios_type;

  typedef char                       char_type;
  typedef typename traits::int_type  int_type;
  typedef typename traits::pos_type  pos_type;
  typedef typename traits::off_type  off_type;

  explicit strstreambuf(streamsize alsize = 0);
  strstreambuf(void *(*palloc)(size_t), void (*pfree)(void *));
  strstreambuf(char *gnext, streamsize n, char *pbeg = 0);

  strstreambuf(unsigned char *gnext, streamsize n,
               unsigned char *pbeg = 0);
  strstreambuf(signed char *gnext, streamsize n,
               signed char *pbeg = 0);

  strstreambuf(const char *gnext, streamsize n);
  strstreambuf(const unsigned char *gnext, streamsize n);
  strstreambuf(const signed char *gnext, streamsize n);

  virtual ~strstreambuf();

  void freeze(bool f = 1);

  char *str();
  int pcount() const;

 protected:

  virtual int_type overflow(int_type c = traits::eof());

  virtual int_type pbackfail(int_type c = traits::eof());

  virtual int_type underflow();

  virtual pos_type seekoff(off_type, ios_type::seekdir way,
                           ios_type::openmode which =
                           ios_type::in | ios_type::out);

  virtual pos_type seekpos(pos_type sp, ios_type::openmode which =
                           ios_type::in | ios_type::out);

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

  virtual streamsize xsputn(const char_type* s, streamsize n);

};
```

Types

char_type
The type char_type is a synonym of type char.

int_type
The type int_type is a synonym of type traits::in_type.

ios_type
The type ios_type is an instantiation of class basic_ios on type char.

off_type
The type off_type is a synonym of type traits::off_type.

pos_type

The type `pos_type` is a synonym of type `traits::pos_type`.

traits

The type `traits` is a synonym of type `char_traits<char>`.

Constructors

```
explicit strstreambuf(streamsize alsize = 0);
```

Constructs an object of class `strstreambuf`, initializing the base class with `streambuf()`. After initialization the `strstreambuf` object is in dynamic mode and its array object has a size of `alsize`.

```
strstreambuf(void* (*palloc)(size_t),
             void (*pfree)(void*));
```

Constructs an object of class `strstreambuf`, initializing the base class with `streambuf()`. After initialization the `strstreambuf` object is in dynamic mode. The function used to allocate memory is pointed at by `void* (*palloc)(size_t)` and the one used to free memory is pointed at by `void (*pfree)(void*)`.

```
strstreambuf(char* gnext, streamsize n,
             char* pbeg = 0);
strstreambuf(signed char* gnext, streamsize n,
             signed char* pbeg = 0);
strstreambuf(unsigned char* gnext, streamsize n,
             unsigned char* pbeg = 0);
```

Constructs an object of class `strstreambuf`, initializing the base class with `streambuf()`. The argument `gnext` point to the first element of an array object whose number of elements is:

```
n, if n > 0
::strlen(gnext), if n == 0
INT_MAX, if n < 0
```

If `pbeg` is a null pointer set only the input sequence to `gnext`, otherwise set also the output sequence to `pbeg`.

```
strstreambuf(const char* gnext, streamsize n);
strstreambuf(const signed char* gnext, streamsize n);
strstreambuf(const unsigned char* gnext, streamsize n);
```

Constructs an object of class `strstreambuf`, initializing the base class with `streambuf()`. The argument `gnext` point to the first element of an array object whose number of elements is:

```
n, if n > 0
::strlen(gnext), if n == 0
INT_MAX, if n < 0
```

Set the input sequence to `gnext` and the mode to `constant`.

Destructors

```
virtual ~strstreambuf();
```

Destroys an object of class `strstreambuf`. The function frees the dynamically allocated array object only if allocated is set and frozen is not set.

Member Functions

```
void
freeze(bool freezefl = 1);
```

If the mode is dynamic, alters the `freeze` status of the dynamic array as follows:

- If `freezefl` is false, the function sets the `freeze` status to `frozen`.
- Otherwise, it clears the `freeze` status.

```
int_type
overflow( int_type c = traits::eof() );
```

If the output sequence has a put position available, and `c` is not `traits::eof()`, then write `c` into it. If there is no position available, the function grow the size of the array object by allocating more memory and then write `c` at the new current put position. If dynamic is not set or if frozen is set the operation fails. The function returns `traits::not_eof(c)`, except if it fails, in which case it returns `traits::eof()`.

```
int_type
pbackfail( int_type c = traits::eof() );
```

Puts back the character designated by `c` into the input sequence. If `traits::eq_int_type(c,traits::eof())` returns true, move the input sequence one position backward. If the operation fails, the function returns `traits::eof()`. Otherwise it returns `traits::not_eof(c)`.

```
int
pcount() const;
```

Returns the size of the output sequence.

```
pos_type
seekoff(off_type off, ios_base::seekdir way,
        ios_base::openmode which =
        ios_base::in | ios_base::out);
```

If the open mode is `in | out`, alters the stream position of both the input and the output sequence. If the open mode is `in`, alters the stream position of only the input sequence, and if it is `out`, alters the stream position of only the output sequence. The new position is calculated by combining the two parameters `off` (displacement) and `way` (reference point). If the current position of the sequence is invalid before repositioning, the operation fails and the return value is `pos_type(off_type(-1))`. Otherwise the function returns the current new position.

```
pos_type
seekpos(pos_type sp,ios_base::openmode which =
        ios_base::in | ios_base::out);
```

If the open mode is `in | out`, alters the stream position of both the input and the output sequence. If the open mode is `in`, alters the stream position of only the input sequence, and if it is `out`, alters the stream position of only the output sequence. If the current position of the sequence is invalid before repositioning, the operation fails and the return value is `pos_type(off_type(-1))`. Otherwise the function returns the current new position.

```
strstreambuf*
setbuf(char* s, streamsize n);
```

If `dynamic` is set and `freeze` is not, proceed as follows:
If `s` is not a null pointer and `n` is greater than the number of characters already in the current array, replaces it (copy its contents) by the array of size `n` pointed at by `s`.

```
char*
str();
```

Calls `freeze()`, then returns the beginning pointer for the input sequence.

```
int_type
underflow();
```

If the input sequence has a read position available, returns the content of this position. Otherwise tries to expand the input sequence to match the output sequence and if possible returns the content of the new current position. The function returns `traits::eof()` to indicate failure.

In the case where `s` is a null pointer and `n` is greater than the number of characters already in the current array, resize it to size `n`.
If the function fails, it returns a null pointer.

```
streamsize
xsputn(const char_type* s, streamsize n);
```

Writes up to `n` characters to the output sequence. The characters written are obtained from successive elements of the array whose first element is designated by `s`. The function returns the number of characters written.

Examples

```
//
// stdlib/examples/manual/strstreambuf.cpp
//
#include<iostream>
#include<strstream>
#include<iomanip>

void main ( )
{
  using namespace std;
```

```
 // create a read/write strstream object
 // and attach it to an ostrstream object
 ostrstream out;

 // tie the istream object to the ostrstream object
 istream in(out.rdbuf());

 // output to out
 out << "anticonstitutionellement is a big word !!!";

 // create a NTBS
 char *p ="Le rat des villes et le rat des champs";

 // output the NTBS
 out << p << endl;

 // resize the buffer
 if ( out.rdbuf()->pubsetbuf(0,5000) )
  cout << endl << "Success in allocating the buffer" << endl;

 // output the all buffer to stdout
 cout << in.rdbuf( );

 // output the decimal conversion of 100 in hex
 // with right padding and a width field of 200
 out << dec << setfill('!') << setw(200) << 0x100 << endl;

 // output the content of the input sequence to stdout
 cout << in.rdbuf( ) << endl;

 // number of elements in the output sequence
 cout << out.rdbuf()->pcount() << endl;

 // resize the buffer to a minimum size
 if ( out.rdbuf()->pubsetbuf(0,out.rdbuf()->pcount()) )
  cout << endl << "Success in resizing the buffer" << endl;

 // output the content of the all array object
 cout << out.rdbuf()->str() << endl;

}
```

See Also

char_traits(3C++), ***ios_base***(3C++), ***basic_ios***(3C++), ***basic_streambuf***(3C++), ***istrstream***(3c++), ***ostrstream***(3C++), ***strstream***(3c++)

Working Paper for Draft Proposed International Standard for Information Systems--Programming Language C++, Annex D Compatibility features Section D.5

Standards Conformance

ANSI X3J16/ISO WG21 Joint C++ Committee

time_get

→ *time_get_base*
time_get
→ *locale::facet*

Summary Time formatting facet for input.

Synopsis

```
#include <locale>
class time_base;
template <class charT, class InputIterator =
          istreambuf_iterator<charT> >
class time_get;
```

Description

The ***time_get*** facet extracts time and date components from a character string and stores the resulting values in a `struct tm` argument. The facet parses the string using a specific format as a guide. If the string does not fit the format, then the facet will indicate an error by setting the `err` argument in the public member functions to `iosbase::failbit`. See member function descriptions for details.

The time_base class provides a set of values for specifying the order in which the three parts of a date appear. The `dateorder` function returns one of these five possible values:

Order	Meaning
`noorder`	Date format has no set ordering
`dmy`	Date order is day, month, year
`mdy`	Date order is month, day, year
`ymd`	Date order is year, month, day
`ydm`	Date order is year, day, month

Interface

```
class time_base {
public:
  enum dateorder { no_order, dmy, mdy, ymd, ydm };
};

template <class charT, class InputIterator =
          istreambuf_iterator<charT> >
class time_get : public locale::facet, public time_base {
public:
  typedef charT           char_type;
  typedef InputIterator   iter_type;
  explicit time_get(size_t = 0);
  dateorder date_order()  const;
  iter_type get_time(iter_type, iter_type, ios_base&,
                     ios_base::iostate&, tm*)  const;
  iter_type get_date(iter_type, iter_type, ios_base&,
                     ios_base::iostate&, tm*)  const;
  iter_type get_weekday(iter_type, iter_type, ios_base&,
                        ios_base::iostate&, tm*) const;
  iter_type get_monthname(iter_type, iter_type, ios_base&,
                          ios_base::iostate&, tm*) const;
  iter_type get_year(iter_type, iter_type, ios_base&,
                     ios_base::iostate&, tm*) const;
  static locale::id id;
protected:
  ~time_get();  // virtual
  virtual dateorder do_date_order()  const;
  virtual iter_type do_get_time(iter_type, iter_type, os_base&,
                                ios_base::iostate&, tm*) const;
  virtual iter_type do_get_date(iter_type, iter_type, ios_base&,
                                ios_base::iostate&, tm*) const;
  virtual iter_type do_get_weekday(iter_type, iter_type, os_base&,
                                   ios_base::iostate&, tm*) const;
  virtual iter_type do_get_monthname(iter_type, ios_base&,
                                     ios_base::iostate&, tm*)
const;
  virtual iter_type do_get_year(iter_type, iter_type, ios_base&,
                                ios_base::iostate&, tm*) const;
};
```

Types

`char_type`

Type of character the facet is instantiated on.

`iter_type`

Type of iterator used to scan the character buffer.

Constructors and Destructors

`explicit` **`time_get`**`(size_t refs = 0)`

Construct a ***time_get*** facet. If the `refs` argument is 0 then destruction of the object is delegated to the locale, or locales, containing it. This allows the user to ignore lifetime management issues. On the other had, if `refs` is 1 then the object must be explicitly deleted; the locale will not do so. In this case, the object can be maintained across the lifetime of multiple locales.

`~time_get``();  // virtual and protected`

Destroy the facet

Facet ID

`static locale::id` **`id`**`;`

Unique identifier for this type of facet.

Public Member Functions

The public members of the ***time_get*** facet provide an interface to protected members. Each public member `xxx` has a corresponding virtual protected member `do_xxx`. All work is delegated to these protected members. For instance, the long version of the public `get_time` function simply calls its protected cousin `do_get_time`.

```
dateorder
date_order()  const;
iter_type
get_date(iter_type s, iter_type end, ios_base& f,
         ios_base::iostate& err, tm* t)  const;
iter_type
get_monthname(iter_type s, iter_type end, ios_base& f,
              ios_base::iostate& err, tm* t) const;
iter_type
get_time(iter_type s, iter_type end, ios_base& f,
         ios_base::iostate& err, tm* t)  const;
iter_type
get_weekday(iter_type s, iter_type end, ios_base& f,
            ios_base::iostate& err, tm* t) const;
iter_type
get_year(iter_type s, iter_type end, ios_base& f,
         ios_base::iostate& err, tm* t) const;
```

Each of these public functions simply calls a corresponding protected virtual `do_` function.

Protected Member Functions

```
virtual dateorder
do_date_order()  const;
```

Return the a value indicating the relative ordering of the three basic parts of a date. Possible return values are:

- `noorder`, indicating that the date format has no ordering, an ordering cannot be determined, or the date format contains variable components other than Month, Day and Year. `noorder` is never returned by the default time_put, but may be used by derived classes.
- one of `dmy, mdy, ymd, ydm`, indicating the relative ordering of Day, Month and Year.

```
virtual iter_type
do_get_date(iter_type s, iter_type end, ios_base&,
            ios_base::iostate& err, tm* t) const;
```

Fills out the `t` argument with date values parsed from the character buffer specified by the range `(s,end]`. If the buffer does not contain a valid date representation then the err argument will be set to iosbase::failbit.

Returns an iterator pointing just beyond the last character that can be determined to be part of a valid date.

```
virtual iter_type
do_get_monthname(iter_type s, ios_base&,
                 ios_base::iostate& err, tm* t) const;
```

Fills out the tm_mon member of the `t` argument with a month name parsed from the character buffer specified by the range `(s,end]`. As with do_get_weekday, this name may be an abbreviation, but the function will attempt to read a full name if valid characters are found after an abbreviation. For example, if a full name is “December”, and an abbreviation is “Dec”, then the string “Dece” will cause an error. If an error occurs then the `err` argument will be set to `iosbase::failbit`.

Returns an iterator pointing just beyond the last character that can be determined to be part of a valid name.

`virtual iter_type`
`do_get_time``(iter_type s, iter_type end, os_base&,`
`             ios_base::iostate& err, tm* t) const;`

Fills out the `t` argument with time values parsed from the character buffer specified by the range `(s,end]`. The buffer must contain a valid time representation. Returns an iterator pointing just beyond the last character that can be determined to be part of a valid time.

`virtual iter_type`
`do_get_weekday``(iter_type s, iter_type end, os_base&,`
`                ios_base::iostate& err, tm* t) const;`

Fills out the `tm_wday` member of the `t` argument with a weekday name parsed from the character buffer specified by the range `(s,end]`. This name may be an abbreviation, but the function will attempt to read a full name if valid characters are found after an abbreviation. For instance, if a full name is “Monday”, and an abbreviation is “Mon”, then the string “Mond” will cause an error. If an error occurs then the `err` argument will be set to `iosbase::failbit`.

Returns an iterator pointing just beyond the last character that can be determined to be part of a valid name.

`virtual iter_type`
`do_get_year``(iter_type s, iter_type end, ios_base&,`
`             ios_base::iostate& err, tm* t) const;`

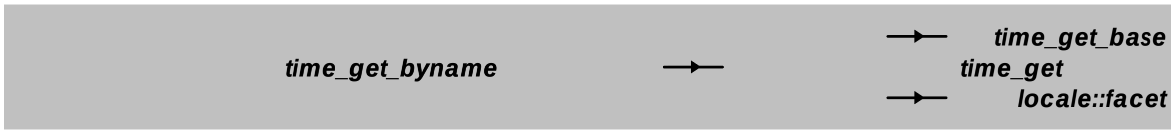
Fills in the `tm_year` member of the `t` argument with a year parsed from the character buffer specified by the range `(s,end]`. If an error occurs then the `err` argument will be set to `iosbase::failbit`.

Returns an iterator pointing just beyond the last character that can be determined to be part of a valid year.

Example

```
//
// timeget.cpp
//
#include <locale>
#include <sstream>
#include <time.h>

  using namespace std;

// Print out a tm struct
ostream& operator<< (ostream& os, const struct tm& t)
{
  os << "Daylight Savings = " << t.tm_isdst << endl;
  os << "Day of year      = " << t.tm_yday << endl;
  os << "Day of week      = " << t.tm_wday << endl;
  os << "Year             = " << t.tm_year << endl;
  os << "Month            = " << t.tm_mon << endl;
  os << "Day of month     = " << t.tm_mday << endl;
  os << "Hour             = " << t.tm_hour << endl;
  os << "Minute           = " << t.tm_min << endl;
  os << "Second           = " << t.tm_sec << endl;
  return os;
}

int main ()
{
  typedef istreambuf_iterator<char,char_traits<char> > iter_type;

  locale loc;
  time_t tm = time(NULL);
  struct tm* tmb = localtime(&tm);
  struct tm timeb;
  memcpy(&timeb,tmb,sizeof(struct tm));
  ios_base::iostate state;
  iter_type end;

  // Get a time_get facet
  const time_get<char,iter_type>& tg =
#ifndef _RWSTD_NO_TEMPLATE_ON_RETURN_TYPE
  use_facet<time_get<char,iter_type> >(loc);
#else
  use_facet(loc,(time_get<char,iter_type>*)0);
#endif

  cout << timeb << endl;
  {
    // Build an istringstream from the buffer and construct
    // beginning and ending iterators on it.
    istringstream ins("12:46:32");
    iter_type begin(ins);

    // Get the time
    tg.get_time(begin,end,ins,state,&timeb);
  }
  cout << timeb << endl;
  {
    // Get the date
    istringstream ins("Dec 6 1996");
    iter_type begin(ins);
    tg.get_date(begin,end,ins,state,&timeb);
  }
  cout << timeb << endl;
  {
    // Get the weekday
    istringstream ins("Tuesday");
    iter_type begin(ins);
    tg.get_weekday(begin,end,ins,state,&timeb);
  }
  cout << timeb << endl;
  return 0;
}
```

See Also ***locale, facets, time_put***



Summary

A facet that provides formatted time input facilities based on the named locales.

Synopsis

```
#include <locale>
template <class charT, class InputIterator =
          istreambuf_iterator<charT> >
class time_get_byname;
```

Description

The ***time_get_byname*** template provides the same functionality as the ***time_get*** template, but specific to a particular named locale. For a description of the member functions of ***time_get_byname***, see the reference for ***time_get***. Only the constructor is described here.

Interface

```
  template <class charT, class InputIterator =
            istreambuf_iterator<charT> >
  class time_get_byname : public time_get<charT, InputIterator> {
  public:
    explicit time_get_byname(const char*, size_t = 0);
  protected:
    ~time_get_byname();  // virtual
    virtual dateorder do_date_order()  const;
    virtual iter_type do_get_time(iter_type, iter_type, ios_base&,
                                ios_base::iostate&, tm*) const;
    virtual iter_type do_get_date(iter_type, iter_type, ios_base&,
                                ios_base::iostate&, tm*) const;
    virtual iter_type do_get_weekday(iter_type, iter_type,
  ios_base&,
                                 ios_base::iostate& err, tm*)
  const;
    virtual iter_type do_get_monthname(iter_type, iter_type,
  ios_base&,
                                     ios_base::iostate&, tm*)
  const;
    virtual iter_type do_get_year(iter_type, iter_type, ios_base&,
                                ios_base::iostate&, tm*) const;
  };
```

Constructor

`explicit` **`time_get_byname`**`(const char* name, size_t refs = 0);`

Construct a ***time_get_byname*** facet. The facet will provide time formatting facilities relative to the named locale specified by the `name` argument. If the `refs` argument is 0 then destruction of the object is delegated to the locale, or locales, containing it. This allows the user to ignore lifetime management issues. On the other had, if `refs` is 1 then the object must be explicitly deleted; the locale will not do so. In this case, the object can be maintained across the lifetime of multiple locales.

See Also ***locale, facets, time_get, time_put_byname***

time_put

→ *time_put_base*
time_put
→ *locale::facet*

Summary Time formatting facet for output.

Synopsis

```
#include <locale>
template <class charT, class OutputIterator =
          ostreambuf_iterator<charT> >
class time_put;
```

Description The ***time_put*** facet provides facilities for formatted output of date/time values. The member function of ***time_put*** take a date/time in the form of a struct `tm` and translate this into character string representation.

Interface

```
template <class charT, class OutputIterator =
          ostreambuf_iterator<charT> >
class time_put : public locale::facet {
public:
  typedef charT            char_type;
  typedef OutputIterator   iter_type;
  explicit time_put(size_t = 0);
  iter_type put(iter_type, ios_base&,
                char_type, const tm*,
                const charT*, const charT*) const;
  iter_type put(iter_type, ios_base&, char_type,
                const tm*, char, char = 0) const;
  static locale::id id;
protected:
  ~time_put();  // virtual
  virtual iter_type do_put(iter_type, ios_base&,
                           char_type, const tm*,
                           char, char) const;
};
```

Types

`char_type`
Type of character the facet is instantiated on.

`iter_type`
Type of iterator used to scan the character buffer.

Constructors and Destructors

`explicit` **`time_put`**`(size_t refs = 0)`
Construct a ***time_put*** facet. If the `refs` argument is 0 then destruction of the object is delegated to the locale, or locales, containing it. This allows the user to ignore lifetime management issues. On the other had, if `refs` is 1 then the object must be explicitly deleted; the locale will not do so. In this case, the object can be maintained across the lifetime of multiple locales.

```
~time_put();  // virtual and protected
```

Destroy the facet

Facet ID

```
static locale::id id;
```

Unique identifier for this type of facet.

Public Member Functions

```
iter_type
put(iter_type s, ios_base& f,
    char_type fill, const tm* tmb,
    const charT* pattern, const charT* pat_end) const;
```

Creates a character string representing the Date/Time contained in `tmb`. The format of the string is determined by a sequence of format modifiers contained in the range `[pattern,pat_end)`. These modifiers are from the same set as those use by the `strftime` function and apply in exactly the same way. The resulting string is written out to the buffer pointed to by the iterator `s`. See the Table 1 below for a description of `strftime` formatting characters.

The `fill` argument is used for any padding.

Returns an iterator pointing one past the last character written.

```
iter_type
put(iter_type s, ios_base& f, char_type fill,
    const tm* tmb, char format, char modifier = 0) const;
```

Calls the protected virtual `do_put` function.

Protected Member Functions

```
virtual iter_type
do_put(iter_type s, ios_base&,
       char_type fill, const tm* t,
       char format, char modifier) const;
```

Writes out a character string representation of the Date/Time contained in `t`. The string is formatted according the specifier `format` and modifier `modifier`. These values are interpreted in exactly the same way that the `strftime` function interprets its format and modifier flags. See the Table 1 below for a description of `strftime` formatting characters.

The `fill` argument is used for any padding.

Returns an iterator pointing one past the last character written.

Table 1. Formatting characters used by strftime().

For those formats that do not use all members of the struct tm, only those members that are actually used are noted [in brackets].

Format character	Meaning	Example
`a`	Abbreviated weekday name [from `tm::tm_wday`]	`Sun`
`A`	Full weekday name [from `tm::tm_wday`]	`Sunday`
`b`	Abbreviated month name	`Feb`
`B`	Full month name	`February`
`c`	Date and time [may use all members]	`Feb 29 14:34:56 1984`
`d`	Day of the month	`29`
`H`	Hour of the 24-hour day	`14`
`I`	Hour of the 12-hour day	`02`
`j`	Day of the year, from 001 [from `tm::tm_yday`]	`60`
`m`	Month of the year, from 01	`02`
`M`	Minutes after the hour	`34`
`p`	AM/PM indicator, if any	`AM`
`S`	Seconds after the minute	`56`
`U`	Sunday week of the year, from 00 [from `tm::tm_yday` and `tm::tm_wday`]	
`w`	Day of the week, with 0 for Sunday	`0`
`W`	Monday week of the year, from 00 [from `tm::tm_yday` and `tm::tm_wday`]	
`x`	Date [uses `tm::tm_yday` in some locales]	`Feb 29 1984`
`X`	Time	`14:34:56`
`y`	Year of the century, from 00 (deprecated)	`84`
`Y`	Year	`1984`

Format character	Meaning	Example
`Z`	Time zone name [from `tm::tm_isdst`]	`PST` or `PDT`

Example

```
//
// timeput.cpp
//
#include <iostream>

int main ()
{
  using namespace std;

  typedef ostreambuf_iterator<char,char_traits<char> > iter_type;

  locale loc;
  time_t tm = time(NULL);
  struct tm* tmb = localtime(&tm);
  struct tm timeb;
  memcpy(&timeb,tmb,sizeof(struct tm));
  char pat[] = "%c";

  // Get a time_put facet
  const time_put<char,iter_type>& tp =
#ifndef _RWSTD_NO_TEMPLATE_ON_RETURN_TYPE
  use_facet<time_put<char,iter_type> >(loc);
#else
  use_facet(loc,(time_put<char,iter_type>*)0);
#endif

  // Construct a ostreambuf_iterator on cout
  iter_type begin(cout);

  cout << " --> ";
  tp.put(begin,cout,' ',&timeb,pat,pat+2);

  cout << endl << " --> ";
  tp.put(begin,cout,' ',&timeb,'c',' ');

  cout <<  endl;

  return 0;
}
```

See Also

locale, facets, time_get

time_put_byname

$$\begin{tikzcd}
 & & \textit{time\_put\_base} \\
\textit{time\_put\_byname} \arrow[r] & \textit{time\_put} \arrow[ur] \arrow[dr] & \\
 & & \textit{locale::facet}
\end{tikzcd}$$

Summary

A facet that provides formatted time output facilities based on the named locales.

Synopsis

```
#include <locale>
template <class charT, class OuputIterator =
          ostreambuf_iterator<charT> >
class time_put_byname;
```

Description

The ***time_put_byname*** template provides the same functionality as the ***time_put*** template, but specific to a particular named locale. For a description of the member functions of ***time_put_byname,*** see the reference for ***time_put***. Only the constructor is described here.

Interface

```
template <class charT, class OutputIterator =
ostreambuf_iterator<charT> >
class time_put_byname : public time_put<charT, OutputIterator>
{
public:
  explicit time_put_byname(const char*, size_t refs = 0);
protected:
  ~time_put_byname();  // virtual
  virtual iter_type do_put(iter_type s, ios_base&,
                           char_type, const tm* t,
                           char format, char modifier) const;
};
```

Constructor

`explicit time_put_byname(const char* name, size_t refs = 0);`

Construct a ***time_put_byname*** facet. The facet will provide time formatting facilities relative to the named locale specified by the `name` argument. If the `refs` argument is 0 then destruction of the object is delegated to the locale, or locales, containing it. This allows the user to ignore lifetime management issues. On the other had, if `refs` is 1 then the object must be explicitly deleted; the locale will not do so. In this case, the object can be maintained across the lifetime of multiple locales.

See Also

locale, facets, time_put, time_get_byname

tolower

Locale Convenience Function

Summary

Converts a character to lower case.

Synopsis

```
#include <locale>

template <class charT>
charT tolower (charT c, const locale& loc) const;
```

Description

The ***tolower*** returns the parameter `c` converted to lower case. The conversion is made using the ***ctype*** facet from the locale parameter.

Example

```
//
// toupper.cpp
//
#include <iostream>

int main ()
{
  using namespace std;

  locale loc;

  cout << 'a' << toupper('c') << tolower('F') << endl;

  return 0;
}
```

See Also

toupper, locale, ctype

toupper

Locale Convenience Function

Summary

Converts a character to upper case.

Synopsis

```
#include <locale>

template <class charT>
charT toupper (charT c, const locale& loc) const;
```

Description

The ***toupper*** returns the parameter `c` converted to upper case. The conversion is made using the ***ctype*** facet from the locale parameter.

Example

```
//
// toupper.cpp
//
#include <iostream>

int main ()
{
  using namespace std;

  locale loc;

  cout << 'a' << toupper('c') << tolower('F') << endl;

  return 0;
}
```

See Also

tolower, locale, ctype

use_facet

Locale Function

Summary

Template function used to obtain a facet.

Synopsis

```
#include <locale>
template <class Facet> const Facet& use_facet(const locale&);
```

Description

use_facet returns a reference to the corresponding facet contained in the locale argument. You specify the facet type be explicitly providing the template parameter. (See the example below.) If that facet is not present then ***use_facet*** throws ***bad_cast.*** Otherwise, the reference will remain valid for as long as any copy of the locale exists.

Note that if your compiler cannot overload function templates on return type then you'll need to use an alternate ***use_facet*** template. The alternate template takes an additional argument that's a pointer to the type of facet you want to extract from the locale. The declaration looks like this:

```
template <class Facet>
const Facet& use_facet(const locale&, Facet*);
```

The example below shows the use of both variations of ***use_facet***.

Example

```
//
// usefacet.cpp
//
#include <iostream>

int main ()
{
  using namespace std;

  locale loc;

  // Get a ctype facet
  const ctype<char>& ct =
#ifndef _RWSTD_NO_TEMPLATE_ON_RETURN_TYPE
     use_facet<ctype<char> >(loc);
#else
     use_facet(loc,(ctype<char>*)0);
#endif

  cout << 'a' << ct.toupper('c') << endl;

  return 0;
}
```

See Also

locale, facets, has_facet

wcerr

Pre-defined stream

Synopsis

```
#include <iostream>
extern wostream wcerr;
```

Description

`wostream` **`wcerr`**`;`

The object `wcerr` controls output to an unbuffered stream buffer associated with the object `stderr` declared in `<cstdio>`. By default the standard C and C++ streams are synchronized, but you can improve performance by using the `ios_base` member function `synch_with_stdio` to desynchronize them.

`wcerr` uses the locale `codecvt` facet to convert the wide characters it receives to the tiny characters it outputs to `stderr`.

Formatting

The formatting is done through member functions or manipulators. See `cout`, `wcout` or `basic_ostream` for details.

Examples

```
//
// wcerr example
//
#include<iostream>
#include<fstream>

void main ( )
{
  using namespace std;

  // open the file "file_name.txt"
  // for reading
  wifstream in("file_name.txt");

  // output the all file to stdout
  if ( in )
    wcout << in.rdbuf();
  else
    // if the wifstream object is in a bad state
    // output an error message to stderr
    wcerr << L"Error while opening the file" << endl;
}
```

See Also

basic_ostream(3C++), ***iostream***(3C++), ***basic_filebuf***(3C++), ***cout***(3C++), ***cin***(3C++), ***cerr***(3C++), ***clog***(3C++), ***wcin***(3C++), ***wcout***(3C++), ***wclog***(3C++), ***iomanip***(3C++), ***ios_base***(3C++), ***basic_ios***(3C++)

Working Paper for Draft Proposed International Standard for Information Systems--Programming Language C++, Section 27.3.1

Standards Conformance

ANSI X3J16/ISO WG21 Joint C++ Committee

Pre-defined stream

Synopsis

```
#include <iostream>
extern wistream wcin;
```

Description

wistream **wcin**;

The object wcin controls input from a stream buffer associated with the object stdin declared in <cstdio>. By default the standard C and C++ streams are synchronized, but performance improvement can be achieved by using the ios_base member function synch_with_stdio to desynchronize them.

After the object wcin is initialized, wcin.tie() returns &wcout, which implies that wcin and wcout are synchronized. wcin uses the locale codecvt facet to convert the tiny characters extracted from stdin to the wide characters stored in the wcin buffer.

Examples

```
//
// wcin example one
//
#include <iostream>

void main ( )
{
  using namespace std;

  int i;
  float f;
  wchar_t c;

  //read an integer, a float and a wide character from stdin
  wcin >> i >> f >> c;

  // output i, f and c to stdout
  wcout << i << endl << f << endl << c << endl;
}

//
// wcin example two
//
#include <iostream>

void main ( )
{
  using namespace std;

  wchar_t p[50];

  // remove all the white spaces
  wcin >> ws;

  // read characters from stdin until a newline
  // or 49 characters have been read
  wcin.getline(p,50);

  // output the result to stdout
  wcout << p;
}
```

When inputting `"   Grendel the monster" (newline)` in the previous test, the output will be `"Grendel the monster"`. The manipulator `ws` removes spaces.

See Also

basic_istream(3C++), ***iostream***(3C++), ***basic_filebuf***(3C++), ***cin***(3C++), ***cout***(3C++), ***cerr***(3C++), ***clog***(3C++), ***wcout***(3C++), ***wcerr***(3C++), ***wclog***(3C++), ***ios_base***(3C++), ***basic_ios***(3C++)

Working Paper for Draft Proposed International Standard for Information Systems--Programming Language C++, Section 27.3.2

Standards Conformance

ANSI X3J16/ISO WG21 Joint C++ Committee

wclog

Pre-defined stream

Synopsis

```
#include <iostream>
extern wostream wclog;
```

Description

`wostream` **`wclog`**`;`

The object `wclog` controls output to a stream buffer associated with the object `stderr` declared in `<cstdio>`. The difference between `wclog` and `wcerr` is that `wclog` is buffered, but `wcerr` isn't . Therefore, commands like `wclog << L"ERROR !!";` and `fprintf(stderr,"ERROR !!");` are not synchronized. `wclog` uses the locale `codecvt` facet to convert the wide characters it receives to the tiny characters it outputs to `stderr`.

Formatting

The formatting is done through member functions or manipulators. See ***cout***, ***wcout*** or ***basic_ostream*** for details.

Examples

```
//
// wclog example
//
#include<iostream>
#include<fstream>

void main ( )
{
  using namespace std;

  // open the file "file_name.txt"
  // for reading
  wifstream in("file_name.txt");

  // output the all file to stdout
  if ( in )
    wcout << in.rdbuf();
  else
    // if the wifstream object is in a bad state
    // output an error message to stderr
    wclog << L"Error while opening the file" << endl;
}
```

Warnings

`wclog` can be used to redirect some of the errors to another recipient. For example, you might want to redirect them to a file named `my_err`:

```
wofstream out("my_err");

if ( out )
  wclog.rdbuf(out.rdbuf());
else
  cerr << "Error while opening the file" << endl;
```

Then when you are doing something like `wclog << L"error number x";` the error message is output to the file `my_err`. Obviously, you can use the same scheme to redirect `wclog` to other devices.

See Also

basic_ostream(3C++), ***iostream***(3C++), ***basic_filebuf***(3C++), ***cout***(3C++), ***cin***(3C++), ***cerr***(3C++), ***clog***(3C++), ***wcin***(3C++), ***wcout***(3C++), ***wcerr***(3C++), ***iomanip***(3C++), ***ios_base***(3C++), ***basic_ios***(3C++)

Working Paper for Draft Proposed International Standard for Information Systems--Programming Language C++, Section 27.3.1

Standards Conformance

ANSI X3J16/ISO WG21 Joint C++ Committee

wcout

Pre-defined stream

Synopsis

```
#include <iostream>
extern wostream wcout;
```

Description

`wostream` **`wcout`**;

The object `wcout` controls output to a stream buffer associated with the object `stdout` declared in `<cstdio>`. By default the standard C and C++ streams are synchronized, but performance can be improved by using the `ios_base` member function `synch_with_stdio` to desynchronize them.

After the object `wcin` is initialized, `wcin.tie()` returns `&wcout`, which implies that `wcin` and `wcout` are synchronized. `wcout` uses the locale `codecvt` facet to convert the wide characters it receives to the tiny characters it outputs to `stdout`.

Formatting

The formatting is done through member functions or manipulators.

Manipulators	Member Functions
`showpos`	`setf(ios_base::showpos)`
`noshowpos`	`unsetf(ios_base::showpos)`
`showbase`	`setf(ios_base::showbase)`
`noshowbase`	`unsetf(ios_base::showbase)`
`uppercase`	`setf(ios_base::uppercase)`
`nouppercase`	`unsetf(ios_base::uppercase)`
`showpoint`	`setf(ios_base::showpoint)`
`noshowpoint`	`unsetf(ios_base::showpoint)`
`boolalpha`	`setf(ios_base::boolalpha)`
`noboolalpha`	`unsetf(ios_base::boolalpha)`
`unitbuf`	`setf(ios_base::unitbuf)`
`nounitbuf`	`unsetf(ios_base::unitbuf)`
`internal`	`setf(ios_base::internal, ios_base::adjustfield)`
`left`	`setf(ios_base::left, ios_base::adjustfield)`
`right`	`setf(ios_base::right, ios_base::adjustfield)`
`dec`	`setf(ios_base::dec, ios_base::basefield)`
`hex`	`setf(ios_base::hex, ios_base::basefield)`
`oct`	`setf(ios_base::oct, ios_base::basefield)`

`fixed`	`setf(ios_base::fixed, ios_base::floatfield)`
`scientific`	`setf(ios_base::scientific, ios_base::floatfield)`
`resetiosflags (ios_base::fmtflags flag)`	`setf(0,flag)`
`setiosflags (ios_base::fmtflags flag)`	`setf(flag)`
`setbase(int base)`	`see above`
`setfill(char_type c)`	`fill(c)`
`setprecision(int n)`	`precision(n)`
`setw(int n)`	`width(n)`
`endl`	
`ends`	
`flush`	`flush( )`

Description

`showpos`	Generates a + sign in non-negative generated numeric output.
`showbase`	Generates a prefix indicating the numeric base of generated integer output.
`uppercase`	Replaces certain lowercase letters with their uppercase equivalents in generated output.
`showpoint`	Generates a decimal-point character unconditionally in generated floating-point output.
`boolalpha`	Inserts and extracts bool type in alphabetic format.
`unitbuf`	Flushes output after each output operation.
`internal`	Adds fill characters at a designated internal point in certain generated output, or identical to right if no such point is designated.
`left`	Adds fill characters on the right (final positions) of certain generated output.
`right`	Adds fill characters on the left (initial positions) of certain generated output.
`dec`	Converts integer input or generates integer output in decimal base.
`hex`	Converts integer input or generates integer output in hexadecimal base.

`oct`	Converts integer input or generates integer output in octal base.
`fixed`	Generates floating-point output in fixed-point notation.
`scientific`	Generates floating-point output in scientific notation.
`resetiosflagss (ios_base::fmtflags flag)`	Resets the `fmtflags` field `flag`
`setiosflags (ios_base::fmtflags flag)`	Sets up the flag `flag`
`setbase(int base)`	Converts integer input or generates integer output in base `base`. The parameter base can be 8, 10 or 16.
`setfill(char_type c)`	Sets the character used to pad (fill) an output conversion to the specified field width.
`setprecision(int n)`	Sets the precision (number of digits after the decimal point) to generate on certain output conversions.
`setw(int n)`	Sets the field with (number of characters) to generate on certain output conversions.
`endl`	Inserts a newline character into the output sequence and flush the output buffer.
`ends`	Inserts a null character into the output sequence.
`flush`	Flushes the output buffer.

Default Values

`precision()`	6
`width()`	0
`fill()`	the space character
`flags()`	skipws | dec
`getloc()`	locale::locale()

Examples

```
//
// wcout example one
//
#include<iostream>
#include<iomanip>

void main ( )
{
  using namespace std;
  int i;
  float f;

  // read an integer and a float from stdin
  cin >> i >> f;

  // output the integer and goes at the line
  wcout << i << endl;

  // output the float and goes at the line
  wcout << f << endl;

  // output i in hexa
  wcout << hex << i << endl;

  // output i in octal and then in decimal
  wcout << oct << i << dec << i << endl;

  // output i preceded by its sign
  wcout << showpos << i << endl;

  // output i in hexa
  wcout << setbase(16) << i << endl;

  // output i in dec and pad to the left with character
  // @ until a width of 20
  // if you input 45 it outputs 45@@@@@@@@@@@@@@@@@@
  wcout << setfill(L'@') << setw(20) << left << dec << i;
  wcout << endl;

  // output the same result as the code just above
  // but uses member functions rather than manipulators
  wcout.fill('@');
  wcout.width(20);
  wcout.setf(ios_base::left, ios_base::adjustfield);
  wcout.setf(ios_base::dec, ios_base::basefield);
  wcout << i << endl;

  // outputs f in scientific notation with
  // a precision of 10 digits
  wcout << scientific << setprecision(10) << f << endl;

  // change the precision to 6 digits
  // equivalents to wcout << setprecision(6);
  wcout.precision(6);

  // output f and goes back to fixed notation
  wcout << f << fixed << endl;

}

//
// wcout example two
//
#include <iostream>

void main ( )
{
  using namespace std;

  wchar_t p[50];

  wcin.getline(p,50);

  wcout << p;
}

//
// wcout example three
//
#include <iostream>
#include <fstream>

void main ( )
{
  using namespace std;

  // open the file "file_name.txt"
  // for reading
  wifstream in("file_name.txt");

  // output the all file to stdout
  if ( in )
    wcout << in.rdbuf();
  else
    {
      wcout << "Error while opening the file";
      wcout << endl;
    }
}
```

Warnings

Keep in mind that the manipulator `endl` flushes the stream buffer. Therefore it is recommended to use L'\n' if your only intent is to go at the line. It will greatly improve performance when C and C++ streams are not synchronized.

See Also

basic_ostream(3C++), ***iostream***(3C++), ***basic_filebuf***(3C++), ***cin***(3C++), ***cout***(3C++), ***cerr***(3C++), ***clog***(3C++), ***wcin***(3C++), ***wcerr***(3C++), ***wclog***(3C++), ***iomanip***(3C++), ***ios_base***(3C++), ***basic_ios***(3C++)

Standards Conformance

Working Paper for Draft Proposed International Standard for Information Systems--Programming Language C++, Section 27.3.1