cppreference.com -> C++ I/O -> Details

C++ I/O

Constructors

Syntax:

```
fstream( const char *filename, openmode mode );
ifstream( const char *filename, openmode mode );
ofstream( const char *filename, openmode mode );
```

The fstream, ifstream, and ofstream objects are used to do file I/O. The optional *mode* defines how the file is to be opened, according to the ios stream mode flags. The optional *filename* specifies the file to be opened and associated with the stream. For example, the following code reads input data and appends the result to an output file.

```
ifstream fin( "/tmp/data.txt" );
ofstream fout( "/tmp/results.txt", ios::app );
while( fin >> temp )
  fout << temp + 2 << endl;
fin.close();
fout.close();
```

Input and output file streams can be used in a similar manner to C++ predefined I/O streams, **cin** and **cout**.

Related topics:
close(), open()

---

bad

Syntax:

```
bool bad();
```

The bad() function returns **true** if a fatal error with the current stream has occurred, **false** otherwise.

Related topics:
good()

---

clear

Syntax:

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

The function clear() clears the flags associated with the current stream. The default flag is goodbit, which clears all flags. Otherwise, only *flags* are cleared.

Related topics:
rdstate()

---

close

Syntax:

```
void close();
```

The close() function closes the associated file stream.

Related topics:
open()

---

eof

Syntax:

```
bool eof();
```

The function eof() returns **true** if the end of the associated input file has been reached, **false** otherwise. For example:

```
char ch;
ifstream fin( "temp.txt" );
while( !fin.eof() ) {
  fin >> ch;
  cout << ch;
}
fin.close();
```

Related topics:
bad(), **fail()**, **good()**, **rdstate()**, **clear()**

---

fail

Syntax:

```
bool fail();
```

The fail() function returns **true** if an error has occurred with the current stream, **false** otherwise.

Related topics:
good(), **eof()**, **bad()**, **clear()**, **rdstate()**

---

fill

Syntax:

```
char fill();
char fill( char ch );
```

The function fill() either returns the current fill character, or sets the current fill character to *ch*. The fill character is defined as the character that is used for padding when a number is smaller than the specified width. The default fill character is the space character.

Related topics:
precision(), **width()**

flags

Syntax:

```
fmtflags flags();
fmtflags flags( fmtflags f );
```

The flags() function either returns the format flags for the current stream, or sets the flags for the current stream to be *f*.

Related topics:
unsetf(), **setf()**

---

flush

Syntax:

```
ostream &flush();
```

The flush() function causes the buffer for the current output stream to be actually written out to the attached device. This function is useful for printing out debugging information, because sometimes programs abort before they have a chance to write their output buffers to the screen. Judicious use of flush() can ensure that all of your debugging statements actually get printed.

Related topics:
put(), **write()**

---

gcount

Syntax:

```
streamsize gcount();
```

The function gcount() is used with input streams, and returns the number of characters read by the last input operation.

Related topics:
get(), **getline()**, **read()**

---

get

Syntax:

```
int get();
istream &get( char &ch );
istream &get( char *buffer, streamsize num );
istream &get( char *buffer, streamsize num, char delim );
istream &get( streambuf &buffer );
istream &get( streambuf &buffer, char delim );
```

The get() function is used with input streams, and either:

- reads a character and returns that value,
- reads a character and stores it as *ch*,

- reads characters into *buffer* until *num* - 1 characters have been read, or EOF or newline encountered,
- reads characters into *buffer* until *num* - 1 characters have been read, or EOF or the *delim* character encountered (*delim* is not read until next time),
- reads characters into *buffer* until a newline or EOF is encountered,
- or reads characters into *buffer* until a newline, EOF, or *delim* character is encountered (again, *delim* isn't read until the next get()).

For example, the following code displays the contents of temp.txt, character by character:

```
char ch;
ifstream fin( "temp.txt" );
while( fin.get(ch) )
  cout << ch;
fin.close();
```

Related topics:
put(), **read()**, **getline()**

---

getline

Syntax:

```
istream &getline( char *buffer, streamsize num );
istream &getline( char *buffer, streamsize num, char delim );
```

The getline() function is used with input streams, and reads characters into *buffer* until either:

- *num* - 1 characters have been read,
- a newline is encountered,
- an EOF is encountered,
- or, optionally, until the character *delim* is read. The *delim* character is not put into *buffer*.

Related topics:
get(), **read()**

---

good

Syntax:

```
bool good();
```

The function good() returns **true** if no errors have occurred with the current stream, **false** otherwise.

Related topics:
bad(), **fail()**, **eof()**, **clear()**, **rdstate()**

---

ignore

Syntax:

```
istream &ignore( streamsize num=1, int delim=EOF );
```

The ignore() function is used with input streams. It reads and throws away characters until *num* characters have been read (defaults to 1) or until the character *delim* is read (which defaults to EOF).

Related topics:
get(), getline()

---

open

Syntax:

```
void open( const char *filename );
void open( const char *filename, openmode mode );
```

The function open() is used with file streams. It opens *filename* and associates it with the current stream. The optional *mode* can be:

Mode	Meaning
ios::app	append output
ios::ate	seek to EOF when opened
ios::binary	open the file in binary mode
ios::in	open the file for reading
ios::out	open the file for writing
ios::trunc	overwrite the existing file

If open() fails, the resulting stream will evaluate to false when used in a Boolean expression. For example:

```
ifstream inputStream("file.txt");
if( !inputStream ) {
  cerr << "Error opening input stream" << endl;
  return;
}
```

Related topics:
close(), fstream(), ifstream(), ofstream(),

---

peek

Syntax:

```
int peek();
```

The function peek() is used with input streams, and returns the next character in the stream or EOF if the end of file is read. peek() does not remove the character from the stream.

Related topics:
get(), putback()

---

precision

Syntax:

```
streamsize precision();
streamsize precision( streamsize p );
```

The precision() function either sets or returns the current number of digits that is displayed for floating-point variables. For example, the following code:

```
float num = 314.15926535;
cout.precision( 5 );
cout << num;
```

displays

```
314.16
```

Related topics:
width(), **fill()**

---

put

Syntax:

```
ostream &put( char ch );
```

The function put() is used with output streams, and writes the character *ch* to the stream.

Related topics:
write(), **get()**

---

putback

Syntax:

```
istream &putback( char ch );
```

The putback() function is used with input streams, and returns the previously-read character *ch* to the input stream.

Related topics:
peek()

---

rdstate

Syntax:

```
iostate rdstate();
```

The rdstate() function returns the status of the current stream. The **iostate** object has the these flags:

Flag	Meaning
badbit	a fatal error has occurred
eofbit	EOF has been found
failbit	a nonfatal error has occurred

goodbit no errors have occurred

Related topics:
eof(), good(), bad(), clear(), fail()

---

read

Syntax:

```
istream &read( char *buffer, streamsize num );
```

The function read() is used with input streams, and reads *num* bytes from the stream before placing them in *buffer*. If EOF is encountered, read() stops, leaving however many bytes it put into *buffer* as they are. For example:

```
struct {
  int height;
  int width;
} rectangle;

input_file.read( (char *)(&rectangle), sizeof(rectangle) );
if( input_file.bad() ) {
  cerr << "Error reading data" << endl;
  exit( 0 );
}
```

Related topics:
gcount(), get(), getline(), write()

---

seekg

Syntax:

```
istream &seekg( off_type offset, ios::seekdir origin );
istream &seekg( pos_type position );
```

The function seekg() is used with input streams, and it repositions the "get" pointer for the current stream to *offset* bytes away from *origin*, or places the "get" pointer at *position*.

Related topics:
seekp(), tellg(), tellp()

---

seekp

Syntax:

```
ostream &seekp( off_type offset, ios::seekdir origin );
ostream &seekp( pos_type position );
```

The seekp() function is used with output streams, but is otherwise very similar to seekg().

Related topics:
seekg(), tellg(), tellp()

---

setf

Syntax:

```
fmtflags setf( fmtflags flags );
fmtflags setf( fmtflags flags, fmtflags needed );
```

The function setf() sets the formatting flags of the current stream to *flags*. The optional *needed* lets only the flags that are in both *flags* and *needed* be set. The return value is the previous configuration of flags. For example:

```
int number = 0x3FF;
cout.setf( ios::dec );
cout << "Decimal: " << number << endl;
cout.unsetf( ios::dec );
cout.setf( ios::hex );
cout << "Hexadecimal: " << number << endl;
```

Note that the preceding code is functionally identical to:

```
int number = 0x3FF;
cout << "Decimal: " << number << endl << hex << "Hexadecimal: " << number << dec << endl;
```

thanks to manipulators.

Related topics:
flags(), unsetf()

---

sync_with_stdio

Syntax:

```
static bool sync_with_stdio( bool sync=true );
```

The sync_with_stdio() function allows you to turn on and off the ability for the C++ I/O system to work with the C I/O system.

---

tellg

Syntax:

```
pos_type tellg();
```

The tellg() function is used with input streams, and returns the current "get" position of the pointer in the stream.

Related topics:
seekg(), seekp(), tellp()

---

tellp

Syntax:

```
pos_type tellp();
```

The tellp() function is used with output streams, and returns the current "put" position of the pointer in the stream. For example, the following code displays the file pointer as it writes to a stream:

```
string s("In Xanadu did Kubla Khan...");
ofstream fout("output.txt");
for( int i=0; i < s.length(); i++ ) {
  cout << "File pointer: " << fout.tellp();
  fout.put( s[i] );
  cout << " " << s[i] << endl;
}
fout.close();
```

Related topics:
seekg(), **seekp()**, **tellg()**

---

unsetf

Syntax:

```
void unsetf( fmtflags flags );
```

The function unsetf() is used to clear the given *flags* associated with the current stream. What flags?

Related topics:
setf(), **flags()**

---

width

Syntax:

```
int width();
int width( int w );
```

The function width() returns the current width. The optional *w* can be used to set the width. Width is defined as the minimum number of characters to display with each output. For example:

```
cout.width( 5 );
cout << "2";
```

displays

```
    2
```

(that's four spaces followed by a '2')

Related topics:
precision(), **fill()**

---

write

Syntax:

```
ostream &write( const char *buffer, streamsize num );
```

The write() function is used with output streams, and writes *num* bytes from *buffer* to the current output stream.

Related topics:
read(), put()