

BC1

Standard Library Header Files

The interface to the C++ Standard Library consists of 51 header files, 18 of which present the C standard library. It's often difficult to remember which header files you need to include in your program files, so this material provides a brief description of the contents of each header, organized into seven categories:

- ❑ The C Standard Library
- ❑ Containers
- ❑ Algorithms, iterators, and allocators
- ❑ Utilities
- ❑ Exceptions
- ❑ Numerical processing
- ❑ I/O Streams

The C Standard Library

The C++ Standard Library includes the entire C Standard Library. The header files are generally the same, except for two points:

- ❑ The header names are `<cname>` instead of `<name.h>`.
- ❑ All the names declared in the header files are in namespace `std`.

For backward compatibility, you can still include `<name.h>` if you want. That puts the names into the global namespace instead of into namespace `std`. We recommend avoiding this feature.

Bonus Chapter 1

The following table provides a summary of the most useful functionality:

Header File Name	Contents
<cassert>	assert() macro.
<cctype>	Character predicates and manipulation functions, such as isspace() and tolower().
<cerrno>	Defines errno expression.
<cfloat>	C-style defines related to floating-point arithmetic, such as FLT_MAX
<ciso646>	In C, the <iso646.h> file defines macros and, or, etc. In C++, those are keywords, so they are not in this header.
<climits>	C-style limit defines, such as INT_MAX.
<locale>	A few localization macros and functions like LC_ALL and setlocale().
<cmath>	Math utilities, including trigonometric functions, sqrt(), fabs(), and others.
<setjmp>	setjmp() and longjmp().
<signal>	signal() and raise().
<stdarg>	Macros and types for processing variable-length argument lists. See Chapter 12.
<stddef>	Important constants and types such as NULL and size_t.
<stdio>	File operations, including fopen() and fclose(). Formatted I/O: printf(), scanf(), and family. Character I/O: getc(), putc(), and family. File positioning: fseek(), ftell(), and family.
<stdlib>	Random numbers with rand() and srand(). abort() and exit() functions. C-style memory allocation functions: calloc(), malloc(), realloc(), free(). C-style searching and sorting with qsort() and bsearch(). String to number conversions: atof(), atoi(), etc.
<string>	Low-level memory management functions, including memcpy() and memset(). C-style string functions, such as strcpy() and strcmp(). Defines NULL and size_t as well.
<time>	Time-related functions, including time() and localtime().
<wchar>	Versions of string, memory, and I/O functions for wide characters.
<wctype>	Versions of functions in <ctype> for wide characters: iswspace(), towlower(), and so on.

Containers

The definitions for the STL containers can be found in eight header files:

Header File	Contents
<code><bitset></code>	The <code>bitset</code> class template
<code><deque></code>	The <code>deque</code> class template
<code><list></code>	The <code>list</code> class template
<code><map></code>	The <code>map</code> and <code>multimap</code> class templates
<code><queue></code>	The <code>queue</code> and <code>priority_queue</code> class templates
<code><set></code>	The <code>set</code> and <code>multiset</code> class templates
<code><stack></code>	The <code>stack</code> class template
<code><vector></code>	The <code>vector</code> class template and the <code>vector<bool></code> specialization

Each of these header files contains all the definitions you need to use the specified container, including iterators. Chapter 21 describes these containers in detail.

Algorithms, Iterators, and Allocators

The “rest” of the STL can be found in five different header files:

Header File Name	Contents
<code><algorithm></code>	Prototypes for most of the algorithms in the STL.
<code><numeric></code>	Prototypes for the rest of the algorithms: <code>accumulate()</code> , <code>inner_product()</code> , <code>partial_sum()</code> , and <code>adjacent_difference()</code> .
<code><functional></code>	Defines the built-in function objects, negators, binders, and adaptors.
<code><iterator></code>	Definitions of <code>iterator_traits</code> , iterator tags, <code>iterator</code> , <code>reverse_iterator</code> , insert iterators, and stream iterators.
<code><memory></code>	Defines the default allocator, some utility functions for dealing with uninitialized memory inside containers, and the <code>auto_ptr</code> class template.

General Utilities

The Standard Library contains some general-purpose utilities in five different header files:

Header File Name	Contents
<code><locale></code>	Defines the <code>locale</code> class, the <code>use_facet()</code> and <code>has_facet()</code> template functions, and the various facet families. See Chapter 14.
<code><new></code>	Defines <code>bad_alloc</code> exception and <code>set_new_handler()</code> function. Prototypes for all six forms of <code>operator new</code> and <code>operator delete</code> . See Chapter 16.
<code><string></code>	Defines the <code>basic_string</code> class template and the typedef instantiations of <code>string</code> and <code>wstring</code> .
<code><typeinfo></code>	Defines <code>bad_cast</code> and <code>bad_typeid</code> exceptions. Defines <code>type_info</code> class, objects of which are returned by <code>typeid</code> operator. See Chapter 10 for details on <code>typeid</code> .
<code><utility></code>	Defines the <code>pair</code> class template. See Chapter 21.

Mathematical Utilities

As described in Chapter 4, C++ provides some facilities for numeric processing. These capabilities are not described in detail in this book; for details, consult one of the Standard Library references listed in the Annotated Bibliography.

Header File Name	Contents
<code><limits></code>	Defines <code>numeric_limits</code> class template, and specializations for most built-in types.
<code><complex></code>	Defines the <code>complex</code> class template for processing complex numbers.
<code><valarray></code>	Defines the <code>valarray</code> and related classes and class templates for processing mathematical vectors and matrices.

Exceptions

Exceptions and exception support is covered in Chapter 15 of this book. Two header files provide most of the requisite definitions, but some exceptions for other domains are defined in the header file for that domain.

Header File Name	Contents
<exception>	Defines the <code>exception</code> and <code>bad_exception</code> classes, and the <code>set_unexpected()</code> , <code>set_terminate()</code> , and <code>uncaught_exception()</code> functions.
<stdexcept>	Non-domain-specific exceptions not defined in <exception>.

I/O Streams

Normally your applications should include only <fstream>, <iomanip>, <iostream>, <istream>, <ostream>, and <sstream>. Consult Chapter 14 for details.

Header File Name	Contents
<fstream>	Defines the <code>basic_filebuf</code> , <code>basic_ifstream</code> , <code>basic_ofstream</code> , and <code>basic_fstream</code> classes. Declares the <code>filebuf</code> , <code>wfilebuf</code> , <code>ifstream</code> , <code>wifstream</code> , <code>ofstream</code> , <code>wofstream</code> , <code>fstream</code> , and <code>wfstream</code> typedefs.
<iomanip>	Declares the I/O manipulators not declared elsewhere (mostly in <ios>).
<ios>	Defines the <code>ios_base</code> and <code>basic_ios</code> classes. Declares most of the stream manipulators. You should rarely include this header directly.
<iosfwd>	Forward declarations of the templates and typedefs found in the other I/O streams header files.
<iostream>	Declares <code>cin</code> , <code>cout</code> , <code>cerr</code> , <code>clog</code> , and the wide-character counterparts. Note that it's not just a combination of <istream> and <ostream>!
<istream>	Defines the <code>basic_istream</code> and <code>basic_iostream</code> classes. Declares the <code>istream</code> , <code>wistream</code> , <code>iostream</code> , and <code>wiostream</code> typedefs.
<ostream>	Defines the <code>basic_ostream</code> class. Declares the <code>ostream</code> and <code>wostream</code> typedefs.
<sstream>	Defines the <code>basic_stringbuf</code> , <code>basic_istringstream</code> , <code>basic_ostringstream</code> , and <code>basic_stringstream</code> classes. Declares the <code>stringbuf</code> , <code>wstringbuf</code> , <code>istringstream</code> , <code>wistringstream</code> , <code>ostringstream</code> , <code>wostringstream</code> , <code>stringstream</code> , and <code>wstringstream</code> typedefs.
<strstream>	Deprecated.
<streambuf>	Defines the <code>basic_streambuf</code> class. Declares the typedefs <code>streambuf</code> and <code>wstreambuf</code> . You should rarely include this header directly.

