
2.1. History of the C++ Standards

The standardization of C++ was started in 1989 by the International Organization for Standardization (ISO), which is a group of national standards organizations, such as ANSI in the United States. To date, this work has resulted in four milestones, which are more or less C++ standards available on different platforms throughout the world:

1. **C++98**, approved in 1998, was the first C++ standard. Its official title is *Information Technology — Programming Languages — C++*, and its document number is ISO/IEC 14882:1998.
2. **C++03**, a so-called "technical corrigendum" ("TC"), contains minor bug fixes to C++98. Its document number is ISO/IEC 14882:2003. Thus, both C++98 and C++03 refer to the "first C++ standard."
3. **TR1** contains library extensions for the first standard. Its official title is *Information Technology — Programming Languages — Technical Report on C++ Library Extensions*, and its document number is ISO/IEC TR 19768:2007. The extensions specified here were all part of a namespace `std::tr1`.
4. **C++11**, approved in 2011, is the second C++ standard. C++11 has significant improvements in both the language and the library, for which the extensions of TR1 have become part of namespace `std`). The official title is again *Information Technology — Programming Languages — C++*, but a new document number is used: ISO/IEC 14882:2011.

This books covers C++11, which long had the working title "C++0x," with the expectation that it would be done no later than 2009.[1] So, both C++11 and C++0x mean the same thing. Throughout the book, I use the term C++11.

[1] The usual joke here is that `x` finally became a hexadecimal `b`.

Because some platforms and environments still do not support all of C++11 (both language features and libraries), I mention whether a feature or behavior is available only since C++11.

2.1.1. Common Questions about the C++11 Standard

Where Is the Standard Available?

The latest freely available draft of the C++11 standard is available as document N3242 (see [*C++Std2011Draft*]). While that draft should be adequate for most users and programmers, those who need the real standard have to pay ISO or a national body a price for it.

Why Did the Standardization Take So Long?

You may wonder why the standardization process for both standards took 10 years or more and why it is still not perfect. Note, however, that the standard is the result of many people and companies suggesting improvements and extensions, discussing them with others, waiting for implementations to test them, and solving all problems caused by the intersection of all the features. Nobody was working as a full-time employee for the new C++ standard. The standard is not the result of a company with a big budget and a lot of time. Standards organizations pay nothing or almost nothing to the people who work on developing standards. So, if a participant doesn't work for a company that has a special interest in the standard, the work is done for fun. Thank goodness a lot of dedicated people had the time and the money to do just that. Between 50 and 100 people regularly met about three times a year for a week to discuss all topics and finish the task and used email throughout the rest of the year. As a result, you won't get anything perfect or consistently designed. The result is usable in practice but is not perfect (nothing ever is).

The description of the standard library took up about 50% of the first standard, and that increased to 65% in the second standard. (With C++11, the number of pages covering the library rose from about 350 to about 750 pages.)

Note that the standard has various sources. In fact, any company or country or even individuals could propose new features and extensions, which then had to get accepted by the whole standardization organization. In principle, nothing was designed from scratch.[2] Thus, the result is not very homogeneous. You will find different design principles for different components. A good example is the difference between the string class and the STL, which is a framework for data structures and algorithms:

[2] You may wonder why the standardization process did not design a new library from scratch. The major purpose of standardization is not to invent or to develop something; it is to harmonize an existing practice.

- String classes are designed as a safe and convenient component. Thus, they provide an almost self-explanatory interface and check for many errors in the interface.
- The STL was designed to combine different data structures with different algorithms while achieving the best performance. Thus, the STL is not very convenient and is not required to check for many logical errors. To benefit from the powerful framework and great performance of the STL, you must know the concepts and apply them carefully.

Both of these components are part of the same library. They were harmonized a bit, but they still follow their individual, fundamental design philosophies.

Nevertheless, another goal of C++11 was to simplify things. For this reason, a lot of proposals were introduced in C++11 to solve problems, inconsistencies, and other flaws people found in practice. For example, the way to initialize values and objects was harmonized with C++11. Also, the more or less broken smart pointer class `auto_ptr` was replaced by multiple improved smart pointer classes, previously matured in Boost, a Web site dedicated to free peer-reviewed portable C++ source libraries (see [*Boost*]) to gain practical experience before being included in a new standard or another technical corrigendum.

Is This the Last C++ Standard?

C++11 is not the end of the road. People already have bug fixes, additional requirements, and proposals for new features. Thus, there will probably be another "technical corrigendum" with fixes of bugs and inconsistencies, and sooner or later, there might be a "TR2" and/or a third standard.

2.1.2. Compatibility between C++98 and C++11

A design goal of C++11 was that it remain backward compatible with C++98. In principle, everything that compiled with C++98 or C++03 should compile with C++11. However, there are some exceptions. For example, variables cannot have the name of newly introduced keywords anymore.

If code should work with different C++ versions but benefit from the improvements of C++11, if available, you can evaluate the predefined macro `__cplusplus`. For C++11, the following definition holds when compiling a C++ translation unit:

```
#define __cplusplus 201103L
```

By contrast, with both C++98 and C++03, it was:

```
#define __cplusplus 199711L
```

Note, however, that compiler vendors sometimes provide different values here.

Note that backward compatibility applies only to the source code. Binary compatibility is not guaranteed, which leads to problems, especially when an existing operation got a new return type, because overloading by the return type only is not allowed (for example, this applies to some STL algorithms and to some member functions of STL containers). So, compiling all parts, including the libraries, of a C++98 program using a C++11 compiler should usually work. Linking code compiled using a C++11 compiler with code compiled using a C++98 compiler might fail.