
Chapter 16. Internationalization

As the global market has increased in importance, so too has *internationalization*, or *i18n* for short,[1] for software development. As a consequence, the C++ standard library provides concepts to write code for international programs. These concepts influence mainly the use of I/O and string processing. This chapter describes these concepts. Many thanks to Dietmar Kühl, who is an expert on I/O and internationalization in the C++ standard library and wrote major parts of this chapter.

[1] The common abbreviation for *internationalization*, *i18n*, stands for the letter *i*, followed by *18* characters, followed by the letter *n*.

The C++ standard library provides a general approach to support national conventions without being bound to specific conventions. This goes to the extent, for example, that strings are not bound to a specific character type to support 16-bit characters in Asia. For the internationalization of programs, two related aspects are important:

1. Different character sets have different properties, so flexible solutions are required for such problems as what is considered to be a letter or, worse, what type to use to represent characters. For character sets with more than 256 characters, type `char` is not sufficient as a representation.
2. The user of a program expects to see national or cultural conventions obeyed, such as the formatting of dates, monetary values, numbers, and Boolean values.

For both aspects, the C++ standard library provides related solutions.

The major approach toward internationalization is to use *locale objects* to represent an extensible collection of aspects to be adapted to specific local conventions. Locales are already used in C for this purpose. In the C++ standard, this mechanism was generalized and made more flexible. In fact, the C++ locale mechanism can be used to address all kinds of customization, depending on the user's environment or preferences. For example, it can be extended to deal with measurement systems, time zones, or paper size.

Most of the mechanisms of internationalization involve only minimal, if any, additional work for the programmer. For example, when doing I/O with the C++ stream mechanism, numeric values are formatted according to the rules of some locale. The only work for the programmer is to instruct the I/O stream classes to use the user's preferences. In addition to such automatic use, the programmer may use locale objects directly for formatting, collation, character classification, and so on.

Strings and streams use another concept for internationalization: *character traits*. They define fundamental properties and operations that differ for different character sets, such as the value of "end-of-file" as well as functions to compare, assign, and copy strings.

Recent Changes with C++11

C++98 specified most features of the localization library. Here is a list of the most important features added with C++11:

- For locales and facets, you can pass a `std::string` now, not only a `const char*` (see Section 16.2.1, page 863).
- A few new manipulators were introduced: `get_money()`, `put_money()`, `get_time()`, and `put_time()` (see Section 16.4.3, page 890, and Section 16.4.2, page 882).
- The `time_get<>` facet now provides a member function `get()` for a complete formatting string (see Section 16.4.3, page 887).
- The facets for numeric I/O now also support `long long` and `unsigned long long`.
- The new character class mask value `blank` and the corresponding convenience function `isblank()` were introduced (see Section 16.4.4, page 894, and Section 16.4.4, page 895).
- Character traits are now also provided for types `char16_t` and `char32_t` (see Section 16.1.4, page 853).
- The new classes `wstring_convert` and `wbuffer_convert` support additional conversions between different character sets (see Section 16.4.4, page 901, and Section 16.4.4, page 903).