Chapter II. Boost.Spirit

Table of Contents

This chapter introduces the library Boost.Spirit. Boost.Spirit is used to develop parsers for text formats. For example, you can use Boost.Spirit to develop a parser to load configuration files. Boost.Spirit can also be used for binary formats, although its usefulness in this respect is limited.

Boost.Spirit simplifies the development of parsers because formats are described with rules. Rules define what a format looks like – Boost.Spirit does the rest. You can compare Boost.Spirit to regular expressions, in the sense that it lets you handle complex processes – pattern searching in the case of regular expressions and parsing for Boost.Spirit – without having to write code to implement that process.

Boost.Spirit expects rules to be described using Parsing Expression Grammar (PEG). PEG is related to Extended Backus-Naur-Form (EBNF). Even if you are not familiar with these languages, the examples in this chapter should be sufficient to get you started.

There are two versions of Boost.Spirit. The first version is known as Spirit.Classic. This version should not be used anymore. The current version is 2.5.2. This is the version introduced in this chapter.

Since version 2.x, Boost.Spirit can be used to generate generators as well as parsers. While parsers read text formats, generators write them. The component of Boost.Spirit that is used to develop parsers is called Spirit.Qi. Spirit.Karma is the component used to develop generators. Namespaces are partitioned accordingly: classes and functions to develop parsers can be found in `boost::spirit::qi` and classes and functions to develop generators can be found in `boost::spirit::karma`.

Besides Spirit.Qi and Spirit.Karma, the library contains a component called Spirit.Lex, which can be used to develop lexers.

This chapter focuses on developing parsers. The examples mainly use classes and functions from `boost::spirit` and `boost::spirit::qi`. For these classes and functions, it is sufficient to include the header file `boost/spirit/include/qi.hpp`.

If you don't want to include a master header file like `boost/spirit/include/qi.hpp`, you can include header files from `boost/spirit/include/` individually. It is important to include header files from this directory only. `boost/spirit/include/` is the interface to the user. Header files in other directories can change in new library versions.