Chapter 57. Boost.Integer

Boost.Integer provides the header file `boost/cstdint.hpp`, which defines specialized types for integers. These definitions originate from the C99 standard. This is a version of the standard for the C programming language that was released in 1999. Because the first version of the C++ standard was released in 1998, it does not include the specialized integer types defined in C99.

C99 defines types in the header file `stdint.h`. This header file was taken into C++11. In C++, it is called `cstdint`. If your development environment supports C++11, you can access `cstdint`, and you don't have to use `boost/cstdint.hpp`.

Example 57.1. Types for integers with number of bits

```
#include <boost/cstdint.hpp>
#include <iostream>

int main()
{
  boost::int8_t i8 = 1;
  std::cout << sizeof(i8) << '\n';

#ifndef BOOST_NO_INT64_T
  boost::uint64_t ui64 = 1;
  std::cout << sizeof(ui64) << '\n';
#endif

  boost::int_least8_t il8 = 1;
  std::cout << sizeof(il8) << '\n';

  boost::uint_least32_t uil32 = 1;
  std::cout << sizeof(uil32) << '\n';

  boost::int_fast8_t if8 = 1;
  std::cout << sizeof(if8) << '\n';

  boost::uint_fast16_t uif16 = 1;
  std::cout << sizeof(uif16) << '\n';
}
```

The types from `boost/cstdint.hpp` are defined in the namespace `boost`. They can be divided into three categories:

- Types such as `boost::int8_t` and `boost::uint64_t` carry the exact memory size in their names. Thus, `boost::int8_t` contains exactly 8 bits, and `boost::uint64_t` contains exactly 64 bits.
- Types such as `boost::int_least8_t` and `boost::uint_least32_t` contain at least as many bits as their names say. It is possible that the memory size of `boost::int_least8_t` will be greater than 8 bits and that of `boost::uint_least32_t` will be greater than 32 bits.
- Types such as `boost::int_fast8_t` and `boost::uint_fast16_t` also have a minimum size. Their actual size is set to a value that guarantees the best performance. Example 57.1 compiled with Visual C++ 2013 and run on a 64-bit Windows 7 system displays `4` for `sizeof(uif16)`.

Please note that 64-bit types aren't available on all platforms. You can check with the macro `BOOST_NO_INT64_T` whether 64-bit types are available or not.

Example 57.2. More specialized types for integers

```
#include <boost/cstdint.hpp>
#include <iostream>

int main()
{
  boost::intmax_t imax = 1;
  std::cout << sizeof(imax) << '\n';

  std::cout << sizeof(UINT8_C(1)) << '\n';

#ifndef BOOST_NO_INT64_T
  std::cout << sizeof(INT64_C(1)) << '\n';
#endif
}
```

Boost.Integer defines two types, `boost::intmax_t` and `boost::uintmax_t`, for the maximum width integer types available on a platform. Example 57.2 compiled with Visual C++ 2013 and run on a 64-bit Windows 7 system displays `8` for `sizeof(imax)`. Thus, the biggest type for integers contains 64 bits.

Furthermore, Boost.Integer provides macros to use integers as literals with certain types. If an integer is written in C++ code, by default it uses the type `int` and allocates at least 4 bytes. Macros like `UINT8_C` and `INT64_C` make it possible to set a minimum size for integers as literals. Example 57.2 returns at least 1 for `sizeof(UINT8_C(1))` and at least 8 for `sizeof(INT64_C(1))`.

Boost.Integer provides additional header files that mainly define classes used for template meta programming.