Chapter 65. Boost.Uuid

Boost.Uuid provides generators for *UUIDs*. UUIDs are universally unique identifiers that don't depend on a central coordinating instance. There is, for example, no database storing all generated UUIDs that can be checked to see whether a new UUID has been used.

UUIDs are used by distributed systems that have to uniquely identify components. For example, Microsoft uses UUIDs to identify interfaces in the COM world. For new interfaces developed for COM, unique identifiers can be easily assigned.

UUIDs are 128-bit numbers. Various methods exist to generate UUIDs. For example, a computer's network address can be used to generate a UUID. The generators provided by Boost.Uuid are based on a random number generator to avoid generating UUIDs that can be traced back to the computer generating them.

All classes and functions from Boost.Uuid are defined in the namespace `boost::uuids`. There is no master header file to get access to all of them.

Example 65.1. Generating random UUIDs with `boost::uuids::random_generator`

```
#include <boost/uuid/uuid.hpp>
#include <boost/uuid/uuid_generators.hpp>
#include <boost/uuid/uuid_io.hpp>
#include <iostream>

using namespace boost::uuids;

int main()
{
  random_generator gen;
  uuid id = gen();
  std::cout << id << '\n';
}
```

Example 65.1 generates a random UUID. It uses the class `boost::uuids::random_generator`, which is defined in `boost/uuid/uuid_generators.hpp`. This header file provides access to all generators provided by Boost.Uuid.

`boost::uuids::random_generator` is used like the generators from the C++11 standard library or from Boost.Random. This class overloads `operator()` to generate random UUIDs.

The type of a UUID is `boost::uuids::uuid`. `boost::uuids::uuid` is a *POD* – plain old data. You can't create objects of type `boost::uuids::uuid` without a generator. But then, it's a lean type that allocates exactly 128 bits. The class is defined in `boost/uuid/uuid.hpp`.

An object of type `boost::uuids::uuid` can be written to the standard output stream. However, you must include `boost/uuid/uuid_io.hpp`. This header file provides the overloaded operator to write objects of type `boost::uuids::uuid` to an output stream.

Example 65.1 displays output that looks like the following: `0cb6f61f-be68-5afc-8686-c52e3fc7a50d`. Using dashes is the preferred way of displaying UUIDs.

Example 65.2. Member functions of `boost::uuids::uuid`

```
#include <boost/uuid/uuid.hpp>
#include <boost/uuid/uuid_generators.hpp>
#include <iostream>

using namespace boost::uuids;

int main()
{
  random_generator gen;
  uuid id = gen();
  std::cout << id.size() << '\n';
  std::cout << std::boolalpha << id.is_nil() << '\n';
  std::cout << id.variant() << '\n';
  std::cout << id.version() << '\n';
}
```

`boost::uuids::uuid` provides only a few member functions, some of which are introduced in Example 65.2. `size()` returns the size of a UUID in bytes. Because a UUID is always 128 bits, `size()` always returns 16. `is_nil()` returns `true` if the UUID is a nil UUID. The nil UUID is 00000000-0000-0000-0000-000000000000. `variant()` and `version()` specify the kind of UUID and how it was generated. In Example 65.2, `variant()` returns 1, which means the UUID conforms to RFC 4122. `version()` returns 4, which means that the UUID was created by a random number generator.

`boost::uuids::uuid` also provides member functions like `begin()`, `end()`, and `swap()` .

Example 65.3. Generators from Boost.Uuid

```
#include <boost/uuid/uuid.hpp>
#include <boost/uuid/uuid_generators.hpp>
#include <boost/uuid/uuid_io.hpp>
#include <iostream>

using namespace boost::uuids;

int main()
{
  nil_generator nil_gen;
  uuid id = nil_gen();
  std::cout << std::boolalpha << id.is_nil() << '\n';

  string_generator string_gen;
  id = string_gen("CF77C981-F61B-7817-10FF-D916FCC3EAA4");
  std::cout << id.variant() << '\n';

  name_generator name_gen(id);
  std::cout << name_gen("theboostcpplibraries.com") << '\n';
}
```

Example 65.3 contains more generators from Boost.Uuid. `nil_generator` generates a nil UUID. `is_nil()` returns `true` only if the UUID is nil.

You use `string_generator` if you want to use an existing UUID. You can generate UUIDs at sites such as http://www.uuidgenerator.net/. For the UUID in Example 65.3, `variant()` returns 0, which means that the UUID conforms to the backwards compatible NCS standard. `name_generator` is used to generate UUIDs in namespaces.

Please note the spelling of UUIDs when using `string_generator`. You can pass a UUID without dashes, but if you use dashes, they must be in the right places. Case (upper or lower) is ignored.

Example 65.4. Conversion to strings

```
#include <boost/uuid/uuid.hpp>
#include <boost/uuid/uuid_generators.hpp>
#include <boost/uuid/uuid_io.hpp>
#include <boost/lexical_cast.hpp>
#include <string>
#include <iostream>

using namespace boost::uuids;

int main()
{
  random_generator gen;
  uuid id = gen();

  std::string s = to_string(id);
  std::cout << s << '\n';

  std::cout << boost::lexical_cast<std::string>(id) << '\n';
}
```

Boost.Uuid provides the functions `boost::uuids::to_string()` and `boost::uuids::to_wstring()` to convert a UUID to a string (see Example 65.4). It is also possible to use `boost::lexical_cast()` for the conversion.