Chapter 55. Boost.System

Boost.System is a library that, in essence, defines four classes to identify errors. All four classes were added to the standard library with C++11. If your development environment supports C++11, you don't need to use Boost.System. However, since many Boost libraries use Boost.System, you might encounter Boost.System through those other libraries.

`boost::system::error_code` is the most basic class in Boost.System; it represents operating system-specific errors. Because operating systems typically enumerate errors, `boost::system::error_code` saves an error code in a variable of type `int`. Example 55.1 illustrates how to use this class.

Example 55.1. Using `boost::system::error_code`

```
#include <boost/system/error_code.hpp>
#include <iostream>

using namespace boost::system;

void fail(error_code &ec)
{
  ec = errc::make_error_code(errc::not_supported);
}

int main()
{
  error_code ec;
  fail(ec);
  std::cout << ec.value() << '\n';
}
```

Example 55.1 defines the function `fail()`, which is used to return an error. In order for the caller to detect whether `fail()` failed, an object of type `boost::system::error_code` is passed by reference. Many functions that are provided by Boost libraries use `boost::system::error_code` like this. For example, Boost.Asio provides the function `boost::asio::ip::host_name()`, to which you can pass an object of type `boost::system::error_code`.

Boost.System defines numerous error codes in the namespace `boost::system::errc`. Example 55.1 assigns the error code `boost::system::errc::not_supported` to **`ec`**. Because **`boost::system::errc::not_supported`** is a number and **`ec`** is an object of type `boost::system::error_code`, the function `boost::system::errc::make_error_code()` is called. This function creates an object of type `boost::system::error_code` with the respective error code.

In `main()`, `value()` is called on **`ec`**. This member function returns the error code stored in the object.

By default, 0 means no error. Every other number refers to an error. Error code values are operating system dependent. Refer to the documentation for your operating system for a description of error codes.

In addition to `value()`, `boost::system::error_code` provides the member function `category()`, which returns an object of type `boost::system::error_category`.

Error codes are simply numeric values. While operating system manufacturers such as Microsoft are able to guarantee the uniqueness of system error codes, keeping error codes unique across all existing applications is virtually impossible for application developers. It would require a central database filled with error codes from all software developers around the world to avoid reusing the same codes for different errors. Because this is impractical, error categories exist.

Error codes of type `boost::system::error_code` belong to a category that can be retrieved with the member function `category()`. Errors created with `boost::system::errc::make_error_code()` automatically belong to the generic category. This is the category errors belong to if they aren't assigned to another category explicitly.

Example 55.2. Using `boost::system::error_category`

```
#include <boost/system/error_code.hpp>
#include <iostream>

using namespace boost::system;

void fail(error_code &ec)
{
  ec = errc::make_error_code(errc::not_supported);
}

int main()
{
  error_code ec;
  fail(ec);
  std::cout << ec.value() << '\n';
  std::cout << ec.category().name() << '\n';
}
```

As shown in Example 55.2, `category()` returns an error's category. This is an object of type `boost::system::error_category`. There are only a few member functions. For example, `name()` retrieves the name of the category. Example 55.2 writes `generic` to standard output.

You can also use the free-standing function `boost::system::generic_category()` to access the generic category.

Boost.System provides a second category. If you call the free-standing function `boost::system::system_category()`, you get a reference to the system category. If you write the category's name to standard output, `system` is displayed.

Errors are uniquely identified by the error code and the error category. Because error codes are only required to be unique within a category, you should create a new category whenever you want to define error codes specific to your program. This makes it possible to use error codes that do not interfere with error codes from other developers.

Example 55.3. Creating error categories

```
#include <boost/system/error_code.hpp>
#include <string>
#include <iostream>
```

```
class application_category :
  public boost::system::error_category
{
public:
  const char *name() const noexcept { return "my app"; }
  std::string message(int ev) const { return "error message"; }
};

application_category cat;

int main()
{
  boost::system::error_code ec{129, cat};
  std::cout << ec.value() << '\n';
  std::cout << ec.category().name() << '\n';
}
```

A new error category is defined by creating a class derived from `boost::system::error_category`. This requires you to define various member functions. At a minimum, the member functions `name()` and `message()` must be supplied because they are defined as pure virtual member functions in `boost::system::error_category`. For additional member functions, the default behavior can be overridden if required.

While `name()` returns the name of the error category, `message()` is used to retrieve the error description for a particular error code. Unlike Example 55.3, the parameter **ev** is usually evaluated to return a description based on the error code.

An object of the type of the newly created error category can be used to initialize an error code. Example 55.3 defines the error code **ec** using the new category `application_category`. Therefore, error code 129 is no longer a generic error; instead, its meaning is defined by the developer of the new error category.

Note

To compile Example 55.3 with Visual C++ 2013, remove the keyword `noexcept`. This version of the Microsoft compiler doesn't support `noexcept`.

`boost::system::error_code` provides a member function called `default_error_condition()`, that returns an object of type `boost::system::error_condition`. The interface of `boost::system::error_condition` is almost identical to the interface of `boost::system::error_code`. The only difference is the member function `default_error_condition()`, which is only provided by `boost::system::error_code`.

Example 55.4. Using `boost::system::error_condition`

```
#include <boost/system/error_code.hpp>
#include <iostream>

using namespace boost::system;

void fail(error_code &ec)
{
  ec = errc::make_error_code(errc::not_supported);
}
```

```
int main()
{
  error_code ec;
  fail(ec);
  boost::system::error_condition ecnd = ec.default_error_condition();
  std::cout << ecnd.value() << '\n';
  std::cout << ecnd.category().name() << '\n';
}
```

`boost::system::error_condition` is used just like `boost::system::error_code`. That's why it's possible, as shown in Example 55.4, to call the member functions `value()` and `category()` for an object of type `boost::system::error_condition`.

While the class `boost::system::error_code` is used for platform-dependent error codes, `boost::system::error_condition` is used to access platform-independent error codes. The member function `default_error_condition()` translates a platform-dependent error code into a platform-independent error code of type `boost::system::error_condition`.

You can use `boost::system::error_condition` to identify errors that are platform independent. Such an error could be, for example, a failed access to a non-existing file. While operating systems may provide different interfaces to access files and may return different error codes, trying to access a non-existing file is an error on all operating systems. The error code returned from operating system specific interfaces is stored in `boost::system::error_code`. The error code that describes the failed access to a non-existing file is stored in `boost::system::error_condition`.

The last class provided by Boost.System is `boost::system::system_error`, which is derived from `std::runtime_error`. It can be used to transport an error code of type `boost::system::error_code` in an exception.

Example 55.5. Using `boost::system::system_error`

```
#include <boost/system/error_code.hpp>
#include <boost/system/system_error.hpp>
#include <iostream>

using namespace boost::system;

void fail()
{
  throw system_error{errc::make_error_code(errc::not_supported)};
}

int main()
{
  try
  {
    fail();
  }
  catch (system_error &e)
  {
    error_code ec = e.code();
    std::cerr << ec.value() << '\n';
    std::cerr << ec.category().name() << '\n';
  }
}
```

In Example 55.5, the free-standing function `fail()` has been changed to throw an exception of type `boost::system::system_error` in case of an error. This exception can transport an error code of type `boost::system::error_code`. The exception is caught in `main()`, which writes the error code and the error category to standard error. There is a second variant of the function `boost::asio::ip::host_name()` that works just like this.