Chapter 61. Boost.NumericConversion

The library Boost.NumericConversion can be used to convert numbers of one numeric type to a different numeric type. In C++, such a conversion can also take place implicitly, as shown in Example 61.1.

Example 61.1. Implicit conversion from `int` to `short`

```
#include <iostream>

int main()
{
  int i = 0x10000;
  short s = i;
  std::cout << s << '\n';
}
```

Example 61.1 will compile cleanly because the type conversion from `int` to `short` takes place automatically. However, even though the program will run, the result of the conversion depends on the compiler used. The number `0x10000` in the variable **`i`** is too big to be stored in a variable of type `short`. According to the standard, the result of this operation is implementation specific. Compiled with Visual C++ 2013, the program displays `0`, which clearly differs from the value in **`i`**.

To avoid these kind of problems, you can use the cast operator `boost::numeric_cast` (see Example 61.2).

Example 61.2. Overflow detection with `boost::numeric_cast`

```
#include <boost/numeric/conversion/cast.hpp>
#include <iostream>

int main()
{
  try
  {
    int i = 0x10000;
    short s = boost::numeric_cast<short>(i);
    std::cout << s << '\n';
  }
  catch (boost::numeric::bad_numeric_cast &e)
  {
    std::cerr << e.what() << '\n';
  }
}
```

`boost::numeric_cast` is used exactly like the existing C++ cast operators. The correct header file must be included; in this case, the header file `boost/numeric/conversion/cast.hpp`.

`boost::numeric_cast` does the same conversion as C++, but it verifies whether the conversion can take place without changing the value being converted. In Example 61.2, this verification fails, and an exception of type `boost::numeric::bad_numeric_cast` is thrown because `0x10000` is too big to be placed in a variable of type `short`.

Strictly speaking, an exception of type `boost::numeric::positive_overflow` will be thrown. This type specifies an overflow – in this case for positive numbers. There is also `boost::numeric::negative_overflow`, which specifies an overflow for negative numbers (see Example 61.3).

Example 61.3. Overflow detection for negative numbers

```
#include <boost/numeric/conversion/cast.hpp>
#include <iostream>

int main()
{
  try
  {
    int i = -0x10000;
    short s = boost::numeric_cast<short>(i);
    std::cout << s << '\n';
  }
  catch (boost::numeric::negative_overflow &e)
  {
    std::cerr << e.what() << '\n';
  }
}
```

Boost.NumericConversion defines additional exception types, all derived from `boost::numeric::bad_numeric_cast`. Because `boost::numeric::bad_numeric_cast` is derived from `std::bad_cast`, a `catch` handler can also catch exceptions of this type.