Toward Boost.Enums 0.1.0

Vicente J. Botet Escriba

Table of Contents

> [C enumerations constitute a curiously half-baked concept.
>
>]
>
> **-Stroustrup**

Warning

Enums is not a part of the Boost libraries.

Overview

How to Use This Documentation

This documentation makes use of the following naming and formatting conventions.

- Code is in `fixed width font` and is syntax-highlighted.

- Replaceable text that you will need to supply is in *`italics`*.
- If a name refers to a free function, it is specified like this: `free_function()`; that is, it is in code font and its name is followed by `()` to indicate that it is a free function.
- If a name refers to a class template, it is specified like this: `class_template<>`; that is, it is in code font and its name is followed by <> to indicate that it is a class template.
- If a name refers to a function-like macro, it is specified like this: `MACRO()`; that is, it is uppercase in code font and its name is followed by `()` to indicate that it is a function-like macro. Object-like macros appear without the trailing `()`.
- Names that refer to *concepts* in the generic programming sense are specified in CamelCase.

Note

In addition, notes such as this one specify non-essential information that provides additional background or rationale.

Finally, you can mentally add the following to any code fragments in this document:

```
// Include all of the core Enums files
#include <boost/enums.hpp>

using namespace boost;
```

Motivation

The David E. Miller, Herb Sutter and Bjarne Stroustrup's proposal (**N1891: Strongly Typed Enums (revision 3)** includes a clear motivation for "Strongly Typed Enums".[@http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2005/n1891.pdf **N1891: Strongly Typed Enums (revision 3)** includes a clear motivation for "Strongly Typed Enums".

On compilers not providing "Strongly Typed Enums" we can make a quite close emulation. This allows to write programs that are portable on comilers providing this feature natively and using the emulation on the others.

Description

Boost.Enums intends to provide a library partial solution to this problem.

Boost.Enums provides:

- Some language-like macros helping to define scoped enum classes.
- Some meta-functions and functions helping to write portable programs using scoped enum classes under comilers supporting them natively or by an emulation on the others.

Users' Guide

Getting Started

Installing Boost.Enums

Getting Boost.Enums

You can get the last stable release of **Boost.Enums** by downloading `enums.zip` from the Boost Vault Utilities directory

You can also access the latest (unstable?) state from the Boost Sandbox.

Building Boost.Enums

There is no need to compile **Boost.Enums**, since it's a header only library. Just include your Boost header directory in your compiler include path.

Requirements

Boost.Enums depends only on Boost.Conversion and Boost.Config (and all libraries they depends on).

Exceptions safety

All functions in the library are exception-neutral and provide strong guarantee of exception safety as long as the underlying parameters provide it.

Thread safety

All functions in the library are thread-unsafe except when noted explicitly.

Tested compilers

The implementation will eventually work with most C++03 conforming compilers. Current version has been tested on:

Windows with

- MSVC 10.0

Cygwin 1.7 with

- GCC 4.3.4

MinGW with

- GCC 4.4.0
- GCC 4.5.0
- GCC 4.5.0 C++0x
- GCC 4.6.0
- GCC 4.6.0 C++0x

Ubuntu 10.10

- GCC 4.4.5
- GCC 4.4.5 -std=c++0x
- GCC 4.5.1
- GCC 4.5.1 -std=c++0x
- clang 2.8

Note

Please let us know how this works on other platforms/compilers.

Note

Please send any questions, comments and bug reports to boost <at> lists <dot> boost <dot> org.

Hello World!

Tutorial

How to define a scoped enum class?

You can define the equivalent of

```
enum class EnumClass : unsigned char) {
  Enum0 = 0,
  Enum1,
  Enum2
};
```

using the provided macros as follows

```
BOOST_ENUM_CLASS_START(EnumClass, unsigned char) {
  Enum0 = 0,
  Enum1,
  Enum2
} BOOST_ENUM_CLASS_END(EnumClass, unsigned char)
```

Simple, isn't it?

How to use scoped enum classes and the associated literals?

Scoped enums classes and the associated literals can be used as the C++0x counterparts in almost all the situations.

Are scoped enum classes convertible to the underlying type?

Scoped enums classes are strong types and the conversion to the underlying type is not implicit. You will need to use the enums::get_value function to get explicitly the value.

How to define a scoped enum type?

scoped enum types add implicit conversion to the underlying type. They can be defined as follows

```
BOOST_ENUM_TYPE_START(EnumType, int) {
  Enum0 = 0,
  Enum1,
  Enum2
} BOOST_ENUM_TYPE_END(EnumType, int)

EnumType e = EnumType::Enum2;
int i = e;
```

Can these scoped enums be used inside unions?

All this depends on your compiler. If the compiler support user classes with constructors, there is no problem. But in the oposite case, you will need to inhibit the constructor.

How to inhibit the constructors generation?

You will need to use the BOOST_ENUM_XXX_NO_CONS_END macros to inhibit the constructor generation

The problem with removing the constructors is that we are unable to have default constructor and copy constructors syntax. So we will need to use a different syntax to get portable programs.

How to replace the default constructor?

The following compiles but the test can fails depending on the implementation

```
{ // defaults to the enum default
  EnumClass e ;
  BOOST_TEST(e==EnumClass::Enum0);
}
```

You need to state explicitly that you want the defaul value

```
{ // defaults to the enum default
  EnumClass e =EnumClass();
  BOOST_TEST(e==EnumClass::Enum0);
}
```

If you have inhibited the constructors, the preceding code fail to compile. The library provides a function that creates scoped enums instances initialized with the default value.

```
{ // defaults to the enum default
  EnumClass e = default_value<EnumClass>();
  BOOST_TEST(e==EnumClass::Enum0);
}
```

How to replace the copy constructor?

The following fails to compile if constructors have been removed:

```
{ // copy constructor emulation
  EnumClass e(EnumClass::Enum2);  // COMPILE ERROR HERE
}
```

The library provides a function convert_to that creates scoped enums instances

```
{ // copy constructor emulation
  EnumClass e=boost::convert_to<EnumClass>(EnumClass::Enum2);
  BOOST_TEST(e==EnumClass::Enum2);
}
```

How to use scoped enums in siwtch statements?

The following fails to compile if constructors have been removed:

```
const char* c_str(EnumClass e)
{
    switch (e) // COMPILE ERROR HERE
    {
    case EnumClass::Enum0 :   return("EnumClass::Enum0");
    case EnumClass::Enum1:    return("EnumClass::Enum1");
    case EnumClass::Enum2 :   return("EnumClass::Enum2");
    default:
      return("EnumClass::???");
    }
}
```

The library provides a function get_value that return the native enum in a portable way.

```
const char* c_str(EnumClass e)
{
    switch (boost::enums::get_value(e))
    {
    case EnumClass::Enum0 :   return("EnumClass::Enum0");
    case EnumClass::Enum1:    return("EnumClass::Enum1");
    case EnumClass::Enum2 :   return("EnumClass::Enum2");
    default:
      return("EnumClass::???");
    }
}
```

How to use scoped enums as non type template parameters?

The following fails to compile if constructors have been removed:

```
template <EnumClass e>
struct ex;
```

The library provides a meta-function enum_type that return the native enum type in a portable way.

```
template <enums::enum_type<EnumClass>::type e>
struct ex;
```

Examples

aligned

cv_status

External Resources

N2347: Strongly Typed Enums (revision 3)	Alisdair Meredith
PC-lint/FlexeLint Strong Type Checking	Gimpel Software
Enumerations	Herb Sutter and Jim Hyslop
Enumerations Q & A	Dan Saks

Reference

Header <boost/enums.hpp>

Include all the enums public header files.

```
#include <boost/enums/include.hpp>
```

Header <boost/enums/include.hpp>

Include all the enums public header files.

```
#include <boost/enums/default_value.hpp>
#include <boost/enums/underlying_type.hpp>
#include <boost/enums/enum_type.hpp>
#include <boost/enums/get_value.hpp>
#include <boost/enums/emmulation.hpp>
```

Header <boost/enums/default_value.hpp>

Builds a enum class with the default value

```
namespace boost {
  namespace enums {

    template <typename EC>
    EC default_value();

  }
}
```

Header <boost/enums/underlying_type.hpp>

Metafunction to retrieve the underlying type of a scoped enum.

```
namespace boost {
  namespace enums {
    template <typename EC>
    struct underlying_type
    {
      typedef <see below> type;
    };

  }
}
```

Header <boost/enums/enum_type.hpp>

Metafunction to retrieve the native enumeration type of a scoped enum.

```
namespace boost {
  namespace enums {
    template <typename EC>
    struct enum_type
    {
      typedef <see below> type;
    };

  }
}
```

Header <boost/enums/get_value.hpp>

Returns the associated native enumeration type of an enum class.

```
namespace boost {
  namespace enums {

    template <typename EC>
    typename enum_type<EC>::type
    get_value(EC e);

  }
}
```

Header <boost/enums/emulation.hpp>

Macro language-like emulating scoped enum classes and types.

```
#define BOOST_ENUM_CLASS_START(EC, UT)
#define BOOST_ENUM_CLASS_END(EC, UT)
#define BOOST_ENUM_CLASS_NO_CONS_END(EC, UT)

#define BOOST_ENUM_TYPE_START(EC, UT)
#define BOOST_ENUM_TYPE_END(EC, UT)
#define BOOST_ENUM_TYPE_NO_CONS_END(EC, UT)
```

Appendices

Appendix A: History

Version 0.1.0, Feb 27, 2011

Initial version committed on the sandbox

Features:

- Some language-like macros helping to define scoped enum classes.
- Some meta-functions and functions helping to write portable programs using scoped enum classes under comilers supporting them natively or by an emulation on the others.

Appendix B: Design Rationale

lala

Appendix C: Implementation Notes

lala

Appendix D: Acknowledgements

Thanks to Beman Dawes for opening the initial discussion. Daniel James for giving the idea of the alternative implementation and to Matt Calabrese for his inshighfull comments on the ML. This library will never be created without the exchanges they made on the ML (see here).

Appendix E: Tests

scoped_enum_class

Table 1. Contructors and Assignement

Name	kind	Description	Result

scoped_enum_type

Table 2. Contructors and Assignement

Name	kind	Description	Result

scoped_enum_no_cons_class

Table 3. Contructors and Assignement

Name	kind	Description	Result

scoped_enum_no_cons_type

Table 4. Contructors and Assignement

Name	kind	Description	Result

Appendix F: Tickets

Appendix F: Future plans

Tasks to do before review

- Complete the doc and the tests

For later releases

- Add first, last, next, prior, index functions.
- Add enum_array.
- Add enum_set.
- Conversion to and from strings.