mylibrary - a template for Creating Boost HTML and PDF documentation using Quickbook, Doxygen and Auto-Indexing.

Paul A. Bristow

Table of Contents

Important

This is a template, NOT an official Boost library.

Note

Comments and suggestions (even bugs!) to Paul.A.Bristow (at) hetp (dot) u-net (dot) com.

Note

A printer-friendly PDF version of this manual is also available.

Introduction

There are many ways of generating documentation, but this combination of tools, (while absurdly complex and troublesome to set up, and a bit slow to build), produces some of the nicest and most comprehensive C++ documentation.

Some features are:

- Documentation is written in Quickbook, a Wiki-Wiki-style language.
- Output in both html and/or Adobe PDF formats. PDF format is useful to readers in a complementary way because it is a convenient single readonly file, and because it can be **globally** searched for any text string, providing a tool complementary to an index (where the index terms are chosen by the author).
- Quickbook files (.qbk like this file `mylibrary.qbk`) are text files that can be edited with your favorite text editor. (Syntax coloring of the Quickbook files can usually be selected for many text editors like GNU emacs, Textpad, and is **very** helpful).

- Portable to nearly all platforms.
- Boost License (so OK for commercial applications as well as non-profit commercial).
- C++ specific, with C++ syntax colored code Quickbook blocks.
- Automatic syntax coloring of C++ code samples (in both html and pdf).
- Cascading Style Sheet support. Deriving from Boostbook.css allows the user to chose his own syntax coloring.
- Quickbook documentation can be embedded into C++ code (as comments) to document the code itself and also provide separate html and pdf documentation. This ensures that the code snippets shown in the documentation have been compiled, and are updated automatically as the source C++ is changed. The output can also be saved, perhaps appended to the example file, and made visible as another snippet.
- Simple markup to link to Doxygen generated entities (namespaces, classes, functions, typedefs).
- Doxygen commands (as C++ comments) provide additional information both in Quickbook automatically added reference section, and also in Doxygen standalone documentation.
- Automatic indexing to Doxygen generated entities (namespaces, classes, functions, typedefs). and author-chosen indexing of text, see AutoIndex.
- Macro system for simple text substitution.
- Simple markup for italics, bold, preformatted, blurbs, code samples, tables, URLs, anchors, images, etc.
- Images (png, svg, jpg) of diagrams, graphs, equations, pictures... are embedded into pdf and linked from html.

 But to allow a standalone zipped version of html, all images must be copied (installed) into the `doc\html\images` sub-directory.

This document is intended to provide a sort of template to help new writers get started, with some handy hints and tips (based on the many pitfalls discovered en route).

It is written in Quickbook to demonstrate some features (but for much more see Quickbook (itself written in Quickbook) especially the Quickbook Syntax Compendium.

This documentation can be rebuilt to confirm that the whole complex toolchain is working correctly.

If all works OK, copy the whole `mylibrary directory` structure to you own workspace. Check that it works and then use each file as a starting point to add you own files.

 I'm sure you will find it useful to do this before you start to write your own stuff! :-)

How to get Index(es) of your documentation

Users get great benefit from an **index** of nearly all documentation, however short. The only downside is that total size of documentation can be much increased by the index. Since space is not usually an issue, hyperlinking (in both html and pdf) makes the size increase an insignificant disadvantage.

A C++ Reference section, using Doxygen (and Doxygen comments in the C++ code provided you can put in the considerable work to provide them) will allow users to get an overview of the program, see what classes and function do, and where used, and to get to view individual files easily.

AutoIndexing - see AutoIndex for guidance.

The AutoIndex "Getting Started and Tutorial " section tells what you **really** need to know.

Step 1 is not really optional - the indexing process is slow, even with explicit specification of the auto-index.exe. So follow these instructions carefully, including essential changes to your user-config.jam.

This is some *junk* that should **never** be indexed! (because it is excluded in the index script file mylibrary.idx).

Here is a mention of degrees ° kelvin which should **not** put the great man's name in the index linking to here.

Voltaire should get an (silly) entry only in the indexing section linking this section or page.

Twain should be indexed whereever it is found.

Idols

You can restrict which sections are indexed for a particular term. So assuming that the docbook document has the usual hierarchical names for section ID's hierarchical names for section IDs(as Quickbook generates, for example), you can easily place a constraint on which sections are examined for a particular term.

For example, if you want to index occurrences of Lord Kelvin's name, but only in the idols section, and Voltaire only in the indexing section, but Twain in all sections, you might then add:

```
Kelvin "" ".idols."
Voltaire "" ".indexing."
```

to the index script file, assuming that the section ID of the intro is "some_library_or_chapter_name.idols", here "mylibrary.idols".

This section contains quotes from some scientist idols two of whose names should **only** be indexed in this section.

"It ain't what you don't know that gets you into trouble. It's what you know for sure that just ain't so." *Mark Twain*

"No problem can stand the assault of sustained thinking." *Voltaire 1694-1778*

"When you measure what you are speaking about and express it in numbers, you know something about it, but when you cannot (or do not) measure it, when you cannot (or do not) express it in numbers, then your knowledge is of a meagre and unsatisfactory kind." *Lord Kelvin*

Lord Kelvin was a famous scientist, but we should **only** want an index to him in the idols section: we don't want a index entry every time we state a temperature in degrees kelvin!

Mark Twain **should** get an index entry both here and in the idols section.

Voltaire should (perversely) only get an "how to get indexes.. " index entry, and **not** in idols section.

Version Info

Last edit to Quickbook file I:\boost-sandbox\guild\mylibrary\libs\mylibrary\doc\mylibrary.qbk was at 12:54:01 PM on 2011-Apr-26.

Tip

This should appear on the pdf version (but may be redundant on a html version where the last edit date is on the first (home) page).

Warning

Home page "Last revised" is GMT, not local time. Last edit date is local time.

Boost.Mylibrary C++ Reference

Header <boost/mylibrary/mylibrary.hpp>

Template for Boost documentation.

Header file for use by example. Also using Quickbook, Doxygen, and indexed by AutoIndex.

```
namespace boost {
  namespace mylibrary {
    class myclass;
    int donowt(int);
  }
}
```

Class myclass

boost::mylibrary::myclass — A test class - a comment description that preceeds the class.

Synopsis

```
// In header: <boost/mylibrary/mylibrary.hpp>


class myclass {
public:
  enum test_enum;
  // construct/copy/destruct
  myclass();
  ~myclass();

  // public member functions
  int test_me(int, const char *);
  void test_me_too(char, char);

  // public data members
  int mypublic_var;  // A public variable.
};
```

Description

`myclass` public construct/copy/destruct

1. ```
   myclass();
   ```

   A constructor.

   A more elaborate description of the constructor.

   This constructor does nothing much.

2. ```
   ~myclass();
   ```

   A destructor.

   A more detailed description of the destructor.

   Warning! This destructor may explode in your face! (The Doxygen command warning will NOT be found by the index term 'warning', but words in the warning, like 'explode', WILL be found).

`myclass` public member functions

1. ```
   int test_me(int a, const char * s);
   ```

   A normal member function taking two arguments, and returning an integer value.

   See Also:

   Test(), ~Test(), testMeToo() and publicVar().

   Parameters: `a` an integer argument.
   `s` a constant character pointer.

   Returns: The test result.
   ```

2.
```
void test_me_too(char c1, char c2);
```

A pure virtual member with descriptions of parameters. And a 'see also' reference to another version of the function.

See Also:

test_me()

Parameters:

`c1`	the first argument.
`c2`	the second argument.

`myclass` public public data members

1.
```
int mypublic_var;
```

Details about the variable. My public class variable.

Type test_enum

boost::mylibrary::myclass::test_enum — An fully useless enum with 3 values.

Synopsis

```
// In header: <boost/mylibrary/mylibrary.hpp>


enum test_enum { test_enum_val1, test_enum_val2, test_enum_val3 };
```

Description

More detailed enum description, needing more than one line, so using C style comment markers.

`test_enum_val1`	Enum value TVal1 (Note: the use of < to link this Doxygen comment to the same line).
`test_enum_val2`	Enum value TVal2.
`test_enum_val3`	Enum value TVal3. (Note: using C style comment markers).

Function donowt

boost::mylibrary::donowt

Synopsis

```
// In header: <boost/mylibrary/mylibrary.hpp>


int donowt(int i);
```

Description

Non-member free function that does nowt useful at all.

Parameters:	`i` is an argument that is ignored completely.
Requires:	No preconditions.
Postconditions:	No side effects.
Returns:	-1 always.

Index

H

I

W