Creating Boost HTML and PDF documentation using Quickbook, Doxygen and Auto-Indexing.

Paul A. Bristow

Table of Contents

Important

This is NOT (nowhere near) yet an official Boost library.

Note

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

A PDF version of this manual that is printer-friendly 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 complementing html because it is a convenient single readonly file, and because it can be **globally** searched for any text string, providing a tool similar to an index, but which does not depend on the author's choice of index terms.
- Quickbook files (.qbk like this file `quick_auto_dox_index.qbk`) are text files that can be edited with your favorite plain text editor. (Syntax coloring of Quickbook files can usually be arranged, and is extremely helpful).
- Portable to nearly all platforms.
- Boost License (so OK for commercial applications as well as non-profit).
- C++ specific, with C++ syntax colored code Quickbook blocks.
- Automatic syntax coloring of C++ code samples in html and pdf.
- CSS support. Changing Boostbook.css allows the user to chose his own syntax coloring, something that people are surprisingly sensitive to! The best method of customising this is not yet clear.
- 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 (re-)compiled, (re-)tested, and are updated automatically as the source C++ is changed.
- Simple markup to link to Doxygen generated entities.
- 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 and to text with filtering. For example, see AutoIndex.
- Macro system for simple text substitution.
- Markup for italics, bold, preformatted, blurbs, notes, code samples, tables, URLs, anchors, images, etc.
- Images (png, svg, jpg) of diagrams, graphs, equations, pictures... are embedded into PDF and linked from html.

⚠ To allow a standalone zipped version of html, the images must be copied into the doc\html\images sub-directory.

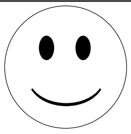
This document is intended to provide a sort of template to help new writers get started,F*.pp

with many handy hints and tips (based on the very many pitfalls encountered en route!).

It is written in Quickbook to demonstrate some features (but for much more see Quickbook manual (itself written in Quickbook) especially the Quick Reference table 28.8 Syntax Compendium).

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

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

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

Acknowledgements

- Docbook authors and maintainers.
- BoostBook (an extension of DocBook) was developed by Douglas Gregor.
- Quickbook was developed by Joel de Guzman and Eric Niebler, and now maintained by Daniel James.
- The link to Doxygen was pioneered by Doug Gregor, and developed by Steven Watanabe.
- Rendering to PDF was made to work by John Maddock.
- Improvements to the tool chain, especially XSLT, by Daniel James and a welcome speedup by Steven Watanabe.
- RenderX kindly provide free use of XEP to render the PDF files from XML.
- Automatic Indexing was developed by John Maddock.
- Doxygen is maintained by Dimitri van Heesch. It was developed by many people who are acknowledged at the Doxygen site.

Hints and Tips

Examples of using Quickbook

You will, of course, find studying other examples of using Quickbook to produce documentation very useful.

The Quickbook manual is, of course, written in Quickbook, and will be the first thing to bookmark.

A document written to demonstrate and test many features is also helpful.

The results are at http://svn.boost.org/svn/boost/trunk/doc/test/gold/index.html and the Quickbook source at test.qbk.

It exposes some things that don't work so well. For example, using */boost:* does not yield correct hyperlink in PDF, and so should not be used. Links to the Boost-trunk and Boost-sandbox are preferred.

Other Boost libraries that use Quickbook for their documentation are

- Boost.mpi
- Boost.Program_options is documented using Quickbook and good coverage of Doxygen comments (but is not autoindexed).
- Boost.Spirit (Autoindexed but does not use Doxygen).
- Boost.Units (but does not using Doxygen to fully describe classes, functions etc).
- Boost.Accumulators (but does not using Doxygen to fully describe classes, functions etc).
- Boost.Math (Autoindexed but does not using Doxygen to fully describe classes, functions etc).
- Boost.Geometry Using of Dxoygen with Quickbook a bit differently to document is described by Barend Gehrels in a another 'real life' story Doxygen & Quickbook.
- SVG_plot A sandbox GSoC project is documented with Quickbook and with Doxygen (with **all** of a very large number of classes and functions fully described) and is also autoindexed.

And several other new libraries with much improved documentation are appearing.

Copyright and license

Do not forget to include copyright, Boost license for **all files** (and include guard for all .hpp files).

```
// Copyright YOURNAME 2009

// Use, modification and distribution are subject to the
// Boost Software License, Version 1.0.
// (See accompanying file LICENSE_1_0.txt
// or copy at http://www.boost.org/LICENSE_1_0.txt)

#ifndef BOOST_QUICK_AUTO_DOX_INDEX_HPP
#define BOOST_QUICK_AUTO_DOX_INDEX_HPP

...

#endif // BOOST_QUICK_AUTO_DOX_INDEX_HPP
```

(And don't forget a final newline right at the end, to ensure full C++ Standard compliance. MSVC in "no extensions mode" /Za message "fatal error C1004: unexpected end-of-file found" signals this bug. You will also be named'n'shamed by the Boost Inspection Report!).

In any Cascading Stylesheets (.css), include your version of

```
/*
  Copyright YOURNAME 2009
  Use, modification and distribution are subject to the
  Boost Software License, Version 1.0.
  (See accompanying file LICENSE_1_0.txt
  or copy at http://www.boost.org/LICENSE_1_0.txt)
*/
```

in html/XML files (for example Doxygen headers and footers)

```
<!-- Copyright 2011 YOURNAME.            -->
<!-- Distributed under the Boost Software License, Version 1.0. -->
<!-- (See accompanying file LICENSE_1_0.txt or copy at          -->
<!-- http://www.boost.org/LICENSE_1_0.txt)                      -->
```

In jamfiles (where # starts comments):

```
# \boost-sandbox\tools\quick_auto_dox_index\libs\quick_auto_dox_index\doc\quick_auto_dox_index.qbk
# Runs Quickbook_Doxygen_indexing documentation example.
# Copyright 2011 YOURNAME.
# Distributed under the Boost Software License, Version 1.0.
# (See accompanying file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt)
```

Hardware and Software Tools

- Screens and Windows. You will soon find that you cannot have too many screens, so that you can keep as many as possible of the various manuals, C++ code, output html and PDF, and the Quickbook file(s) being edited as visible as possible.
- Quickbook files are plain ASCII 7-bit text so any text editor (not Word!) will suffice.
- Syntax coloring provided by your text editor is **very** helpful in seeing immediately what is comment and what is keyword and bracket. Even rudimentary coloring is a big help. Some contributed syntax coloring files for various text editors are at syntax_colors.
- Quickbook syntax is necessarily hugely unforgiving of mismatched `[]` brackets. A text editor that matches these will help avoid grief. Notepad++, Textpad, Emacs, Microsoft IDE Automatic Brace Matching and/or ctrl], and many others can all do this.

Included files and using statements

When you list #include files, it is often helpful to the reader to know **why** it is included, for example by listing the functions/classes used:

```
#include <iostream>
  using std::cout;
  using std::endl;
```

Although this introduces some clutter, it may help to reduce the 'implicit include syndrome' where missing includes only pop up in use, often with very puzzling error messages.

BUT you should **never** impose your `using` choices globally on an unsuspecting user who includes **your** files, so for .hpp files, either comment these out:

```
#include <iostream>
  // using std::cout;
  // using std::endl;
```

or you can, of course, still write using statements at **local** scope, within functions, but these do not document which functions are provided by an include.

When explaining code function is the main aim, for example tutorial examples, you could do **both**

- commenting the using statement(s) `// using std::cout;` after the `#include <iostream>`,
- **and** writing `using std::cout;` at local scope(s).

Links to source code and including snippets of code

Provide links to source code, especially examples.

In Quickbook, provide a link like

../../example/src/quick_auto_dox_index.cpp

This link assumes that the .qbk file is at `/libs/quick_auto_dox_index/doc` and the examples are at `/libs/quick_auto_dox_index/example/src`.

Quick access to header files is provided in the C++ Reference section.

But you can also provide a link to a header file like this:

../../../../boost/quick_auto_dox_index/quick_auto_dox_index.hpp

This assumes the Boost 'standard' folder layout.

Show the output from example programs.

It is also **really** helpful for users to see the output from example (and other) programs **together with** the program. This can be done as 'in-line' comments:

If you write

```
cout << "my_value = " << my_value << endl; // my_value = 9
```

You can also copy and paste the output from a screen in full, or just part elsewhere in the code.

(If you are using MSVC IDE, add a post build (or custom) event:

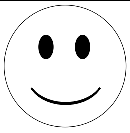
```
Select Project, Property, Build events, Post-build event, and add a Commandline of

"$(TargetDir)$(TargetName).exe"
```

and optionally a description of

```
"Autorun "$(TargetDir)$(TargetName).exe""
```

This will autorun the program after re-building and include the output in the IDE Output Window (removing any blank lines, a MSVC IDE 'feature' which may confuse you).

This speeds the *edit/compile/run/crash' cycle* 

It is then easy to copy and paste any or all of the output into the source code as a comment.

You can do this by enclosing it in `/* ... */` C-style comments but it may also be useful to add output to the Doxygen output by adding it to the `\details` section.

```
\file
  \brief Trivial example.
  \details Too trivial for details.

  Output:
  \verbatim
     This is the program output.
     my_value = 9
  \endverbatim
```

Note

\file picks up the real filename automatically, so is prefered.

Show the output using Quickbook 'snippets'

You can also make it a Quickbook template some or all of the output (or source code) as C++ comment, so it can be included in Quickbook source. Code snippet markup ensures that the C++ source and documentation cannot get out of sync.

```
//[quick_auto_dox_index_output
Autorun "I:\boost-sandbox\tools\quick_auto_dox_index\libs\quick_auto_dox_in↵
dex\doc\quick_auto_dox_index.cpp"
This is the demo output.
//] [/quick_auto_dox_index_output]
```

Adding the name of the template as a comment (just add a prefix '/') after the 'opening bracket' will help you (and others) to avoid confusion about the start and end of snippets sections. If you have more than one snippet, it is also useful to number label them, for example:

```
//[quick_auto_dox_index_1
... 1st snippet code
//] [\quick_auto_dox_index_1]

 //[quick_auto_dox_index_2
... 2nd snippet code
//] [\quick_auto_dox_index_2]
```

Images, equations and graphics

You can insert an image of specific file type:

```
[$my_image.jpg]
```

But you need to ensure that image is in the same directory as the html.

It is usually assumed that all images in a sub-directory, say `/images`, so you will need to preface for example (and making it small, or larger, and try to align in center or left)

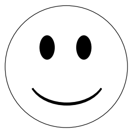
Or, for some images, making it mercifully small, and aligned right:

Note

[align right] works on html, but not on pdf!

[$images/smiley.png] produces the png smiley

Text or a newline is needed to avoid images on the same 'line'.

[$images/smiley.svg] produces the svg smiley 

But [valign top] does not seem to have any effect?

You can NOT insert an image of unspecified type [$my_image] but ...

This template graphic (template could be called `graph` or `equation` or `diagram`) automatically inserts the right type for html (PNG) and pdf (SVG) from the folder named images.

```
[template graphic[name]  '''<inlinemediaobject>
<imageobject role="html">
<imagedata align="center" fileref="./images/'''[name]'''.png"></imagedata>
</imageobject>
<imageobject role="print">
<imagedata align="center" fileref="./images/'''[name]'''.svg"></imagedata>
</imageobject>
</inlinemediaobject>''']
```

Tip

You will need to insert the right path - and it **must be relative**: `./images/` not absolute `/images/`

or you will get an error message perhaps like:

[error] Failed to create image file:/images/pc1.svg of type null [error] java.io.FileNotFoundException: \images\pc1.svg (The system cannot find the path specified)

rather than one like one warning where the directory is wrong:

[warning] Could not retrieve image from 'null': java.net.MalformedURLException: Invalid URL or non-existent file: I:\boost-sandbox\tools\quick_auto_dox_index\libs\quick_auto_dox_index\doc\html\images/./images/smiley.svg [error] Failed to create image file:/images/my_image.png of type null [error] java.io.FileNotFoundException: \images\my_image.png (The system cannot find the path specified)

where the mistake in directory is clear.

Note

align="right" valign = "bottom" does not seem to have any effect?

Use template graphic thus:

```
[graphic smiley]
```

You will have to provide both files types, for example, **both** smiley.png and smiley.svg.

Inkscape provides a convenient way to convert.

Caution

It is not permitted to insert an image (or anchor) outside a section.

See posts by Daniel James and John Maddock.[@http://lists.boost.org/boost-docs/2008/10/3659.php posts by Daniel James and John Maddock.

This **only applies to pdf generation**, and produces an error message like this

(validate error Element 'fo:inline' cannot be a child of 'fo:flow'. Only block-level elements are permitted in this context. Parse error: Invalid XSL FO source

You can add images to your html and pdf documentation (with optional size control and alignment using image metadata).

It is often neater to put all the images in a sub-folder, say `libs/your_library/doc/html/images/`, especially as it is often best to use JPEG or PNG images for html (because a commonly-used browser does not render SVG images without an add-in), but .PNG may not be rendered well into pdfs.

Caution

file separator **must** be backslash / - forward slash \ is the Quickbook trip character!

To specify the location (actual images folder) of the admonitions:

```
path-constant nav-images : $(local-boost-root)/doc/src/images ; # png and svg images for home, ↵
next, note, tip...
```

To ensure SVG admonitions are used for pdf:

```
<format>pdf:<xsl:param>admon.graphics.extension".svg"
  <format>pdf:<xsl:param>admon.graphics.path$(nav-images)/
```

To set the location of your images folder, usually /doc/html/ (not /doc/html/images/):

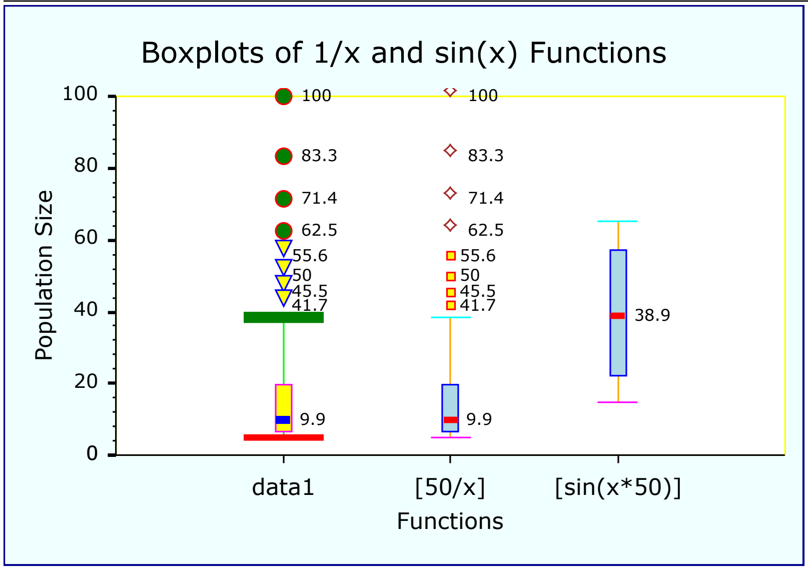
```
path-constant images_location : html ; # location of SVG images referenced by Quickbook.
```

To record in the bjam log file the location actually used:

```
echo "images_location" $(images_location) ;
```

For example: images_location I:\boost-sandbox\SOC\2007\visualization\libs\svg_plot\doc\html

And here is an image in `libs\svg_plot\doc\html\images`



Callouts

callouts are tiny icons linking notes to the C++ source, similar to footnotes in print.

(But it may be clearer to add short comments on the same line as the code.

```
cout << 2+2 << endl; // Outputs "4".
```

because to force the reader to move his eye down to the notes below is distracting, unless the note is rather long to add as a comment.

So there is a choice between comments distracting the code, and callouts jumping to and fro to the notes.)

To use callouts see callouts

The location of the callout icons can be specified in your jamfile.v2 with

```
#<xsl:param>callout.graphics.path=../../imagescallouts
```

The png icons themselves are at `/boost_1_46_0/tools/quickbook/doc/html/images/callouts`

and another set is at `/boost-sandbox/boost0x/doc/src/images/callouts`

and might reasonably be copied (installed) to `doc/html/images/callouts/`.

Suggestions when using Windows

- To run jamfiles, you will find opening a "Command Window Here" either when you *shift and right-click* on a folder, or you can patch the registry to make this option always available - saves pressing the shift key, and is recommended.

Command Window Here, and many others, describe how to do this.

Since you may be typing `bjam -a >my.log` quite often, this may be useful.

Batch files containing this may also be convenient, for example:

```
bjam -a html > quickdoc_%date%_%time:~0,2%%time:~3,2%_html.log
```

- You may find MS OS environment variables helpful (even if some regard them as evil!) In particular you may wish to define BOOST_ROOT and use these as $(BOOST_ROOT) in file specifications, for example: doc/html/boostbook.css.
- You can 'read' environment variables, and use to copy a 'master' boostbook.css thus: [pre path-constant boost-root : [modules.peek : BOOST] ; install css-install : $(boost-root)/doc/src/boostbook.css : <location>html ;]
- You can 'read' and display the location of other environment variables: [pre local boost_sandbox = [os.environ BOOST_SANDBOX] ; ECHO "os.environ boost-sandbox = " $(boost_sandbox) ; # Upper case for the value of environment variable(s).

```
os.environ boost-sandbox =  i:\boost-sandbox
```

] * To set (or change) MS environment variables, per user or globally, use Control Panel, System, Advanced System settings, Advanced tab, Environment variables. For examples, you could set BOOST_ROOT as `I:/boost_1_46_0/`. You need to close the command window and reopen it for this to take effect.

- To check a current value, open a command window anywhere and type `> set boost_root` Surprisingly, this will display the current value `BOOST_ROOT=I:/boost_1_46_0/` (and not set anything - unless you append something to set!).
- You should be able to refer to files relative to XSL parameter boost-root boost:/boost/units/detail/utility.hpp should make it relative to xsl parameter boost.root. in the `jamfile.v2`. [pre <xsl:param>boost.root=$(BOOST_ROOT)] Boost logo # xsl parameter boost.root, for example `I:/boost_1_46_0/boost.png`

Getting started with Doxygen

1. Doxygen documentation provides up-to-date information.
2. Doxygen manual and commands.
3. Doxygen download.
4. The Windows Doxygen GUI frontend doxywizard will get you off to a (prematurely?) quick start, but you will soon find you need to use the 'Expert' options, and then perhaps to hand edit the `doxyfile` where all the very many options are stored.
5. It is useful to (re-)name the doxyfile from the default. This annoyingly untyped so you would need to set mime-type manually to commit to SVN. It also means that you cannot pin the doxyfile to the Windows 7 Doxywizard taskbar icon. If you are maintaining more than one library this is a nuisance. It is also more convenient to have a separate name for each doxyfile, or you will have to wait for the tooltip to know doxyfile which is which. So this template example uses the name `quickdox_doxyfile.txt`.
6. It is worth getting a Doxygen-style index working well because the display is, for some purposes, more friendly and compact that the automatically produced 'Reference' section that is inserted into the Quickbook produced documentation.
7. Running Doxygen is much quicker than running the whole Quickbook tool chain, so it is much quicker to iron out any bugs and missing/broken Doxygen comments in include files using Doxygen's Doxywizard first.
8. It is prudent to save to a Doxygen log file, for example `doxywarn.log`. It is tidier to ensure that there are no warnings, and to check that the log file is empty!
9. The main thing you need to get started is to select the directory containing C++ files. You may well need more than one (`/examples/detail` and `/src` ...), so you immediately need the expert tab and chose 'input'. You save your setting into an untyped file called Doxyfile. A reasonable place for this file is `/doc/Doxygen`. Examining this (well documented) file with a text editor will reveal the very many options. You probably do NOT want LaTex or RTF or other output formats.
10. You might want LaTeX for producing formula (see Boost.Accumulators documentation for an example).
11. When Doxygen is run automatically by the Quickbook auto-doc, you will probably need to copy some settings from your doxyfile as parameters into your jamfile.v2.

12. You probably don't need a main page - using the \mainpage command (because the Doxygen output will be automatically incorporated into the Docbook). But if you provide one, users will be able to better use Doxygen 'free-standing' - this is sometimes useful, especially if you have fully Doxygenated your classes, functions etc (highly recommended, especially using `\pre \post \param \tparam \returns \warning \remark \see` commands, but hard work). You should provide a link to the full Quickbook documention.

Tip

The Doxygen is **not** a compiler! - it is well worth checking from time to time that your added Doxygen or other comments have not caused failure to compile. Not surprisingly, Doxygen can get confused by stuff that fails to compile!

Writing C++ with Doxygen comments

A Reference section, using Doxygen (and Doxygen comments in the C++ code where 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 read individual files easily.

- It is useful to always add a Doxygen `\file` info block as the first item in every `.*pp` file in your library, and adding at least `\brief` description, and often also `\details`. `\date` and `\author` might also be useful, but `\author` is usually redundant, and `\date` liable to be out-of-date. For example:

```
/*!
  \file quick_auto_dox_index_cpp.hpp
  \brief Template for documentation.
  \details Example file for creating Boost documentation using Quickbook, Doxygen and Auto-Index↵
ing.
*/
```

- Using the otherwise blank line(s) (see braces policy - popular but not mandatory) starting with `{` to hold Doxygen comments thus `{ //!<` saves 'vertical space'. (Note the < to 'pin' the comments to the item to the left. Doxygen generally assumes that comments apply to the item *below*).
- There seem to be occasions when Doxygen mis-attaches the comment to the variable.
- Class information, in particular, seems **to have to be before** the class definition. (Using `\class my_class` ... does not seem to help).
- See examples in .cpp and .hpp for adding Doxygen comments.
- See Boost coding guidelines boost:/more/writingdoc/design.html
- Take special care not to use any html characters in Doxygen brief or details comments. A popular mistake is using *n > 0* in a parameter precondition. You need to preface > with the trip character \>. See the several symbols to which this applies.

You will only get a uninformative warning about this when xsltproc tries to process this!

```
warning: Unsupported xml/html tag<_>  found xslt-xsltproc.windows
```

- It is **much** quicker to run Doxygen from the screen or a command file, so it is better to do this first before you try to generate html or pdf - which may take many minutes, especially if you have lots of code and want indexes. Until you get no warnings (a blank warnings file) it is may be premature to run the jamfile.
- Doxygen is a bit unpredictable about when it links the comments with functions, so you can check that your code comments have the desired effect much more quickly with the compact Doxygen display.

Jamfiles

1. Jam and Bjam are the Outer Mongolian of computer languages, so expect to be surprised and puzzled ;-) If all goes well, it works well: if things go wrong, diagnosis is difficult. Ading -d2 (or even a higher debug level) to the bjam command may help.
2. Whitespace MUST terminate variable name! so spaces or newlines BEFORE ; and : and AFTER too (because : and ; are keywords!).
3. The strange layout you will see in jamfiles is designed to make this mistake less likely.
4. Colored syntax provided by your text editor will reduce the risk of misplaced (or mis-spaced) ; and :
5. \ is the jam trip character, so if you really want a \, for example in path and filename specifications, you **must** write \\.
6. It also means you **cannot** just copy a windows file specification from Windows Explorer because it will have back slashes :-(
7. If you specify path and filename that include a space you **must** enclose the whole specification in quotes. It is probably wise to always do this, and to always use / rather than \.
8. Always include your copyright and Boost license as comments (# is the jamfile comment start indicator).

BoostBook

1. The manual is at BoostBook documentation.
2. C++ Syntax coloring can be changed to suit using CSS named colors in boostbook.css. If you want to impose your own C++ syntax colors in pdf, add this line to your `jamfile.v2` and copy your own version of `boostbook.css` to `libs/your_lib/doc/html/`. A 'Boost Standard' way of customising syntax colors has not been established.

```
# Use your local preference for syntax coloring.
  <format>pdf:<xsl:param>html.stylesheet="./html/boostbook.css"
```

Note

The path has to be relative to the destination directory.

1. To add multiple stylesheets, you can provide a list, **space separated** [pre <xsl:param>html.stylesheet="./html/boostbook.css ./html/my_style_preferences.css"]
2. To limit the width of text to 80 character width, you could add to a stylesheet (perhaps a second stylesheet) [pre html { max-width: 80em; }]

Quickbook

- The manual is at Quickbook.
- Always append copyright and Boost license to Quickbook files (.qbk) as comments - see at the end of this file.
- Always give the section an explicit ID, for example: `section:hints` (**no** spaces before or after the colon) rather than relying on the default conversion of section_title to an id. This is because the title may produce a difficult to predict long string, inconvenient to type in for links.
- After each `endsect` block copy the full corresponding `section` block just after, and turn it into a comment by adding a / after the open bracket. This is because nesting can make it difficult to see which is the matching `endsect`. Readers of the quickbook source file, and even those doing maintenance will thank you for this - it might even be you :-).
- Make sure you do not include any 8-bit (Latin-1) codes in any C++ files or in Quickbook files. Common offendors are accents, umlaut, letter n with tilde, plusminus sign, degrees sign. C++ must only use ANSI 7-bit code (blame paper tape!) - **even in comments**.

Sorry that this means that some names are not spelt right - M Bezier, for example, but it's the C++ rules. MSVC IDE will warn you that you are going to do this - say no! If you make this mistake, the error will reported in the pdf production stage when it is really difficult to trace the original file and location of the offending character(s).

Adding a Doxygen Reference section in Quickbook

- A reference section prepared from C/C++ files is extremely valuable to users, and is automatically derived from the actual code, so cannot get out of date as code changes. This is a major benefit for maintenance of documentation. (Using code snippets also helps avoid stale code in documentation).
- The usefulness of a reference section is greatly increased by added Doxygen comments. These can explain the general function, and gives pre- and post- conditions, and return values. Sadly, commenting is a major task, best done at the time of writing the code. But it will increase the quality and maintainability of the code; (This comes at the price of cluttering the code with comments, but syntax coloring is a massive help in allowing the reader to mentally filter out the comments).
- You can ensure that you have documented **all** classes and functions with the Dxoygen options `WARN_IF_UNDOCUMENTED=YES` and `WARN_NO_PARAMDOC=YES`. Any omissions (or mistakes) are logged in the `doxywarn.log` file. If you plan to provide complete Doxygen-style comments (recommended), you will want to enable this option in the doxyfile, and also in your jamfile.

```
<doxygen:param>WARN_IF_UNDOCUMENTED=YES # Default NO but useful if you aim to Doxygen document ↵
all members.
<doxygen:param>WARNINGS=YES # Default NO, but useful to see warnings, especially in a logfile.
<doxygen:param>WARN_NO_PARAMDOC=YES # Default no, but YES useful if you aim to document all func↵
tion parameters.
```

The last option ensures that you **always** provide a Doxygen comment like

```
\tparam Template parameter like FPT floating-point type float, double ...
\param n Number of things required.
\returns Number of previous things for all parameters.
```

- You will want to save any auto-doc Doxygen warnings, probably to a different log file from the Doxywizard 'hand-run' warnings. You will need to consult these logs to resolve mistakes. The pdf log also provides a useful list of missing/broken links (that the html log does not).
- You will find it useful to run the jamfiles from batch file(s) and send the output to log files because there is usually too much output for a screen display. This also allows you to generate both html and pdf with one batch file. And new builds of Quickbook 1.4 upwards also set a return code so you can see immediately if all went well.
  Sadly it does not signal all errors, so you may need to inspect the log file and the resultant html or pdf to be sure.

```
bjam -a html > html.log
echo %errorlevel%
bjam -a pdf > pdf.log
echo %errorlevel%
```

- You will almost certainly find it necessary to set some Doxygen parameters. See jamfile for some documented examples, including suggestions.
- Default section name/title for the Doxygen reference section is just *Reference* but you can (optionally) provide your own title for your reference section like this example:

```
<xsl:param>"boost.doxygen.reftitle=QuickbookDoxygenAutoindex Reference"
```

- The position of this `[xinclude autodoc.xml]` command in the Quickbook determines the location of the Doxygen references section. By convention, it is at the end, but before other index(es).

How Doxygen parses and displays macros

```
<doxygen:param>ENABLE_PREPROCESSING=YES # YES (default) evaluates conditional compilation state↲
ments (like #if) and evaluates macro definitions, but it does not perform macro expansion.
    <doxygen:param>MACRO_EXPANSION=YES # YES will expand all macro names in the source code (de↲
fault = NO).
    <doxygen:param>EXPAND_ONLY_PREDEF=YES # If the EXPAND_ONLY_PREDEF and MACRO_EXPANSION tags ↲
are both set to YES

    # If EXPAND_ONLY_PREDEF tag can be used to specify a list of macro names that should be ex↲
panded (as defined).
    # The PREDEFINED tag can be used to specify one or more macro names that are defined
    # before the preprocessor is started (similar to the -D option of gcc).
    # The argument of the tag is a list of macros of the form:
    #    name or namedefinition (no spaces).
    # If the definition and the "" are omitted, "1" is assumed.
    # To prevent a macro definition from being undefined via #undef or
    # recursively expanded use the : operator instead of the = operator.
    # See http:/www.stack.nl~dimitri/doxygen/config.html#cfg_predefined.
    # static char *malloc BOOST_PREVENT_MACRO_SUBSTITUTION(const size_type bytes);
    # will not produce a helpful Doxygen output, so
    # replace some with more helpful text, or none, for example:

    <doxygen:param>"PREDEFINED\\
      # BOOST_PREVENT_MACRO_SUBSTITUTION \\ # Don't show anything for this macro.
      \"BOOST_MPL_ASSERT(expr)" \
      "BOOST_STATIC_ASSERT(x)=" \
      "BOOST_DEDUCED_TYPENAME=typename" \  # Replace text with just typename.
      "BOOST_STRING_TYPENAME=typename" \
      "BOOST_CONSTEXPR=constexpr" \ # Replace text.
      "BOOST_STATIC_CONSTANT(T,V)=static x const y" \ # Replace text.
      "BOOST_UNITS_AUTO_STATIC_CONSTANT(a,b)=static const auto a = b" \
      "BOOST_UNITS_TYPEOF(a)=typeof(a)" \
      "BOOST_UNITS_HAS_TYPEOF=1" \
      "BOOST_UNITS_HAS_TYPEOF=1" \
      "list_impl=list" \ # Replace text.
      "
    # BOOST_PREVENT_MACRO_SUBSTITUTION,  will not be replaced by ,
    # BOOST_STATIC_CONSTANT will be replaced by static x const y,
    # BOOST_DEDUCED_TYPENAME will be replaced by typename,
    # BOOST_CONSTEXPR will be replaced by constexpr.

    # The syntax hoops to jump through are interesting for more than one PREDEFINED item,
    # and to permit spaces within definitions (use double quotes).
    # Do not forget that every double quote needs a preceeding \trip character!
    # and that each trailing continuation needs a preceeding \trip character too!
    # And finally that if more than one item is included (as here) the whole is
    # enclosed in PREDEFINED... , but without any leading \.  Go figure...

    # A grep for PREDEFINED in jamfiles will reveal even more complex examples.
   # Boost Libraries with useful examples are: Accumulators, Interprocess, MPI, Random, Units, ↲
Expressive.
```

- If there's a 1-to-1 correspondence between the macro uses and the user-visible entities that they create, then you can just have Doxygen expand the macros to something that looks sane in the documentation. The problem is when one macro defines multiple things that need to be documented independently.

- Doxygen commands start with a backslash (\) or at-sign (@). A useful convention is that @ is used for markup like @c to get a typewriter font (by convention, used for code and prefixed by the *backwards single tick* in Quickbook), but backslash is used for \pre, \post \returns etc that become part of the output text. So a C++/Doxygen comment example might be

```
\\param n @c number of things \returns number of unique types of thing
```

.]

Autodoc internals

`doc/bin/msvc/debug/auto-index-internal-on/threading-multi/` contains the files generated for html if this option is chosen.

`<format>html:<auto-index-internal>on`

`/doc/bin` folder contains all the XML files created by Doxygen from your chosen files.

`autodoc-xml.doxyfile` contains the Doxygen's doxyfile from <doxygen:param> settings that you have set.

`autodoc-xml.doxygen` contains XML index entries created by Doxygen.

`autodoc-xml.boostbook` contains the XML generated by Quickbook.

`my_library.docbook` contains the html generated from Quickbook with added Doxygen.

`my_library.modified.docbook` contains XML additions from auto-index.

`autodoc-xml.xml-dir` contains the word *Stamped* after some processing stage.

Folder `autodoc-xml` contains xml and xsd created by Doxygen about your `.*pp` files.

XSL Parameters

There are many parameters to control layout that will need to be passed into html and pdf generation through your `jamfile.v2`.

The `jamfile` for this project contains many examples (deliberately including many that may not be needed, and deliberately over-commented).

DocBook XSL Stylesheets: Reference Documentation.

Printer margins

Make sure that the margins of pdf are enough (unless you are desparate to take revenge on US writers for all those documents they have produced that are unreadable on paper by the rest of the world!).

Making Postscript and PDF International by J Palme.

You should use the International Standard paper size A4, then the jamfile should contain

```
<format>pdf:<xsl:param>paper.type=A4
  # Margins sizes:
  <format>pdf:<xsl:param>page.margin.top=1.3in
  <format>pdf:<xsl:param>page.margin.inner=0.8in
  <format>pdf:<xsl:param>page.margin.bottom=1.3in
  <format>pdf:<xsl:param>page.margin.outer=0.8in
```

This should ensure that it is printable on both A4 and US letter paper sizes, even if an option 'resize to fit' is not available.

PDF versions

A major advantage of this toolchain is that PDF version can be produced. These are single standalone files that can be read on all systems. HTML versions consist of many files. A zip can be produced and downloaded to be isntalled on a users computer, but that is much more trouble, and potentially troublesome.

The jamfile for this project is an example of the extra steps needed.

Sadly the Apache FOP program just does not work for the examples that have been tried.

Caution

Do not make the mistake of removing setting of apparently redundant FOP options in the jamfile - these remain essential even when using RenderX.

So a commercial program in Java Render XEP has been used, kindly provided free-of-charge for Boost (at the price of a discreet logo on the bottom of each page).

RenderX XEP converts XML (and SVG) documents into a printable form (PDF or PostScript) by applying XSL Formatting Objects with program XSLTproc.exe.

You need to install the RenderX XEP program at a location of your choice, and enter the address in your user-config.jam. An example (at `C:\users\paul\`) is

```
#  Copyright (C) Vladimir Prus 2003. Permission to copy, use, modify, sell and
  #  distribute this software is granted provided this copyright notice appears in
  #  all copies. This software is provided "as is" without express or implied
  #  warranty, and with no claim as to its suitability for any purpose.

  #  This file is used to configure your Boost.Build installation. Please read
  #  the user manual to find out where to put it.

  #  Toolset declarations are most important in this file. They tell Boost.Build
  #  what compilers are available and where to look for them. The first toolset
  #  will become "default" one.
  #  Some important libraries can also be configured.
  #  Uncomment relevant parts to suite your local configuration and preferences.

  # Copyright Paul A. Bristow 2010
  # for hetpA Windows 7 msvc 10 2 Aug 2010

  ECHO $(BOOST_ROOT) ;

  import toolset : using ;

  #  MSVC configuration
  #  Configure msvc (default version, searched in standard location and PATH).
  using msvc : 10.0 ;

  using quickbook
    : "C:/Program Files/Quickbook/quickbook.exe" # Note essential quotes!
    ;

  using xsltproc
    : "C:/Program Files/xsltproc_win32/xsltproc.exe" # OK.
    # actually "C:/Program Files/xsltproc_win32/xsltproc.exe" according to Windows.
    ;

  # BoostBook configuration
  # DOCBOOK_XSL_DIR is "C:/Program Files/Docbook_xsl_1_70_1"
  # DOCBOOK_DTD_DIR is "C:/Program Files/DocbookXML/42"
  # using boostbook <<<<  NOT THIS FILE FORMAT - MUST BE DOS filename!!!!
  #  AND must use forward slash /, or double backslash \ and NOT single backslash \
  #  : "C:/Program Files/DocBookXSL/docbook_xsl_1_70_1"
  #  ;

  using boostbook
    : "C:\Program Files\docbook\docbook-xsl-1.74.0" # OK
    # Contains eXtended Style Sheet programs.
    : "C:\Program Files\DocbookXML\42" # dir exists and contains docbookx.dtd 4.2
    #  actually C:\Program Files\DocbookXML\42
    ;

  # DTD (a document type definition), or schema, that provides instructions
  # about the structure of your particular XML document.
  # It's a rule book that states which tags <a_tag> are legal, and where.

  # Doxygen info must come after boostbook info
  using doxygen
    #: "C:/Program Files/doxygen" # NOT OK - because doesn't search from path into /bin?
    : "C:/Program Files/Doxygen/bin/doxygen.exe" # OK
    ;

  # XSLT-FO processor from RenderX.com
  #  actually C:\Program Files\RenderX\XEP\xep.bat
  #  actually  C:\Program Files\Java\jre6
  using fop
    : # XEP batch filename
     "C:/Progra~1/renderx/xep/xep.bat" # OK
    #  C:/Progra~1/renderx/xep/xep.bat # OK
    # "C:/Program Files/renderx/xep/xep.bat" # Definitely does NOT seem to work.
    # "C:\Program Files\renderx\xep\xep.bat" # NOT OK
    # 'C:\Program' is not recognized as an internal or external command, operable program or ↵
batch file.

    : # Java executable called by xep.bat.
    # C:/PROGRA~1/Java/jre6 #OK
    # "C:/PROGRA~1/Java/jre6" # OK
    # "C:/Program files/Java/jre6" # OK
      "C:\Program files\Java\jre6" # OK
    # "C:\Program files\Java\jre6" # NOT OK!
    ;


  #I:\boost-sandbox\tools\auto_index\build
  using auto-index
    :
    "I:/boost-sandbox/tools/auto_index/build/auto_index.exe"
    ;

  # I:\boost-sandbox\tools\auto_index\auto-index.jam
  # Must copy this auto-index.jam manually for each Boost release to tools/build/v2,
  #  for example to I:\boost_1_46_0\tools\build\v2 ...
  # Or bjam build will fail complaing of missing auto-index.jam.
  # This requirement will go away when autoindex is accepted as a Boost library.
```

Caution

All lines about FOP options in the jamfile are essential, even when using RenderX.

Note

You may need to increase the Java heap size.

Logos, Navigation Admonition Icons

The location of the many icons for tip, note, warning, next, prev etc.

```
<format>pdf:<xsl:param>admon.graphics.path=$(nav-images)/
```

You probably DO want these navigation icons:

```
<xsl:param>nav.layout=none # NO home, prev, next navigation icons
```

Logo - Boost reviewed and accepted libraries

You will want to have the Boost C++ libraries standard logo on your first page.

```
<xsl:param>boost.image=Boost # options are: none (no logo), Boost (for boost.png), or your own ↵
logo like inspired_by_boost.png
  <xsl:param>boost.image.w=170 # Width of logo in pixels.
  <xsl:param>boost.image.h=90 # Height of logo in pixels.
```

Proposed_for_boost Logo - Boost not yet reviewed libraries

You may want to have some Boost C++ libraries logo on your first page.

```
<xsl:param>boost.imageBoost # options are: none (no logo), Boost (for boost.png), or your own ↵
logo like inspired_by_boost.png
  <xsl:param>boost.image.src./images/Proposed_for_boost.png
  # A variant for projects intended as candidates for Boost.
  <xsl:param>boost.image.w=170 # Width of logo in pixels.
  <xsl:param>boost.image.h=90 # Height of logo in pixels.
```

Powered_by_boost Logo - libraries wishing to acknowledge use of Boost Libraries

```
<xsl:param>boost.image.src=./images/Powered_by_boost.png
  # A variant for projects that want to acknowledge that they use Boost.
```

Projects entirely outside Boost

```
<xsl:param>boost.image.srcimages/my_project_logo.png
  <xsl:param>boost.image.alt""My Project""
  # Alternative text if .png srcimages can't be displayed.
  <xsl:param>boost.image.w=100
  <xsl:param>boost.image.h=50

  #<xsl:param>nav.layout=none
   # NO home, prev, next navigation icons (but you probably DO want these).
```

Autoindex

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.

AutoIndexing - see AutoIndex for guidance.

The "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 changes to your `user-config.jam`.

When writing Quickbook that is to be autoindexed, bear in mind that the indexing is limited to referencing sections, not individual heading,lines or locations. Thus you may find creating not-too-long sections, rather than using long sections with many headings.

Creating an index files, for example, called `quick_auto_dox_index.idx` is the step that requires your expert knowledge, and your skill reading the users' minds, by choosing the right index terms (including variants like plurals).

Making AutoIndex optional

It is considerate to make the **use of auto-index optional** in Boost.Build, to allow users who do not have AutoIndex installed to still be able to build your documentation.

This also very convenient while you are refining your documentation, to allow you to decide to build indexes, or not: building indexes can take long time, if you are just correcting typos, you won't want to wait while you keep rebuilding the index!

One method of setting up optional AutoIndex support is to place all AutoIndex configuration in a the body of a bjam if statement:

```
if --enable-index in  [ modules.peek : ARGV ]
  {
     ECHO "Building the  docs with automatic index generation enabled." ;

     using auto-index ;
     project : requirements
          <auto-index>on
          <auto-index-script>index.idx

           ... other auto-index options here...

        # And tell Quickbook that it should enable indexing.
        <quickbook-define>
    ;
  }
  else
  {
    ECHO "Building the my_library docs with automatic index generation disabled. To get an auto-
index, try building with --enable-index." ;
  }
```

You will also need to add a conditional statement at the end of your Quickbook file, so that the index(es) is/are only added after the last section if indexing is enabled.

```
[? enable_index
'''
  <index/>
'''
]
```

To use this jamfile, you need to cd to your docs folder, for example:

```
cd \boost-sandbox\guild\mylibrary\libs\mylibrary\doc
```

and then run `bjam` to build the docs without index, for example:

```
bjam -a html > mylibrary_html.log
```

or with index(es)

```
bjam -a html --enable-index > mylibrary_html_index.log
```

QuickbookDoxygenAutoindex Reference

Header <boost/quick_auto_dox_index/detail/FP_compare.hpp>

Two types of floating-point comparison "Very close" and "Close enough" to a chosen tolerance.

Derived from Boost.Test Copyright Gennadiy Rozental 2001-2007. See http://www.boost.org/libs/test for the Boost.Test library home page. Deliberately removed any treatment of percent to avoid further potential confusion!

Mar 2009

Paul A. Bristow

```

template<typename FPT = double> class close_to;
template<typename FPT = double> class smallest;
typedef close_to< double > neareq;
typedef smallest< double > tiny;

// epsilon for type T (about 1e-16 for double)
template<typename FPT> FPT epsilon(FPT);
template<typename FPT> FPT fpt_abs(FPT);

// maximum value for floating-point type T.
template<typename FPT> FPT max_value(FPT);

// minimum value for floating-point type T.
template<typename FPT> FPT min_value(FPT);
template<typename FPT> FPT safe_fpt_division(FPT, FPT);
```

Class template close_to

close_to — Test if two floating-point values are close within a chosen tolerance.

Synopsis

```
// In header: <boost/quick_auto_dox_index/detail/FP_compare.hpp>

template<typename FPT = double  // floating-point type.
         >
class close_to {
public:
  // construct/copy/destruct
  template<typename FPT>
    explicit close_to(FPT, floating_point_comparison_type = FPC_STRONG);
  close_to();

  // public member functions
  bool operator()(FPT, FPT) const;
  FPT size();
  floating_point_comparison_type strength();
};
```

Description

Close can be one of two types of floating-point comparison "Very close" and "Close enough". equations in Dougles E. Knuth, Seminumerical algorithms (3rd Ed) section 4.2.4, Vol II, pp 213-225, Addison-Wesley, 1997, ISBN: 0201896842. Strong requires closeness relative to BOTH values begin compared, Weak only requires only closeness to EITHER ONE value.

`close_to` public construct/copy/destruct

1.
```
template<typename FPT>
  explicit close_to(FPT tolerance,
                    floating_point_comparison_type fpc_type = FPC_STRONG);
```

   Constructor for `close_to` from tolerance and strength. (By design, percent is NOT implemented).

2.
```
close_to();
```

   Default is two epsilon for the FPT.
   Note that some user-defined floating-point types may not specialize std::numeric_limits<FPT>::epsilon() so it is convenient to use boost::math::tools::epsilon<FPT>(); instead.

`close_to` public member functions

1.
```
bool operator()(FPT left, FPT right) const;
```

   Test if two floating-point values are close within a chosen tolerance.

   Tolerance can be interpreted as Knuth's "Very close" (equation 1), the default, or "Close enough" (equation 2).

2.
```
FPT size();
```

   Returns: the chosen tolerance, as a **fraction** (not a percentage).

3.
```
floating_point_comparison_type strength();
```

   Returns: strength of comparison, Knuth's "Very close" (equation 1), the default, or "Close enough" (equation 2).

Class template smallest

smallest — Check floating-point value is smaller than a chosen small value, default is twice min_value() for the floating-point type FPT.

Synopsis

```
// In header: <boost/quick_auto_dox_index/detail/FP_compare.hpp>

template<typename FPT = double  // A floating-point type, float, double, long
                                // double or user-defined like NTL quad_float
                                // or RR.
         >
class smallest {
public:
  // construct/copy/destruct
  template<typename FPT> explicit smallest(FPT);
  smallest();

  // public member functions
  bool operator()(FPT, FPT);
  bool operator()(FPT);
  FPT size();
};
```

Description

It is somewhat common for beginners to add a comparison check to 0 before computing a division, in order to avoid possible division-by-zero exceptions or the generation of infinite results.
A first objection to this practise is that, anyway, computing 1/x for x very close to zero will generate very large numbers that will most probably result in overflows later.
Another objection, which few programmers know about and that we wish to draw attention to, is that it may actually fail to work, depending on what the compiler does, that is, the program may actually test that x == 0, then, further down, find that x = 0 without any apparent change to x!
David Monniaux, http://arxiv.org/abs/cs/0701192v4.

`smallest` public construct/copy/destruct

1.
```
template<typename FPT> explicit smallest(FPT s);
```

   Constructor with user defined value of smallest, for example 10 * min_value<FPT>. Note that some user-defined floating-point types may not specialize std::numeric_limits<FPT>::min_value() so it is convenient to use boost::math::tools::min_value<FPT>(); instead.

2.
```
smallest();
```

   Default Constructor with smallest_ = 2. * boost::math::tools::min_value<double>();
   multiplier m = 2 (must be integer or static_cast<FPT>()) is chosen to allow for a few bits of computation error.
   Pessimistic multiplier is the number of arithmetic operations, assuming every operation causes a 1 least significant bit error, but a more realistic average might be half this.

`smallest` public member functions

1.
```
bool operator()(FPT fp_value, FPT s);
```

   True if value is smaller than a smallest value s.

2.
```
bool operator()(FPT fp_value);
```

   True if value is smaller than chosen smallest value.

3.
```
FPT size();
```

Returns: chosen smallest value that will be counted as effectively zero.

Type definition neareq

neareq — Allow neareq as a shorthand for twice double epsilon = 4.44e-16.

Synopsis

```
// In header: <boost/quick_auto_dox_index/detail/FP_compare.hpp>


typedef close_to< double > neareq;
```

Description

Since double and the default close_to value 2 * epsilon = std::numeric_limits<double>::epsilon = 2 * 2.220446e-016 = 4.440892e-016 is a very common requirement, provide an convenience alias for this.

Type definition tiny

tiny — Allow tiny as a shorthand for twice the double min_value 4.45e-308.

Synopsis

```
// In header: <boost/quick_auto_dox_index/detail/FP_compare.hpp>


typedef smallest< double > tiny;
```

Description

Since double and the default smallest value 2 * std::numeric_limits<double>::min_value() = 2 * 2.22507e-308 + 4.45015e-308 is a very common requirement, provide an convenience alias for this.

Function template fpt_abs

fpt_abs

Synopsis

```
// In header: <boost/quick_auto_dox_index/detail/FP_compare.hpp>


template<typename FPT> FPT fpt_abs(FPT arg);
```

Description

abs function (just in case abs is not defined for a user-defined FPT).

Function template safe_fpt_division

safe_fpt_division

Synopsis

```
// In header: <boost/quick_auto_dox_index/detail/FP_compare.hpp>


template<typename FPT> FPT safe_fpt_division(FPT f1, FPT f2);
```

Description

Division safe from underflow and overflow. (Both f1 and f2 must be unsigned here).

Header <boost/quick_auto_dox_index/quick_auto_dox_index.hpp>

Template for documentation.

Example file for creating Boost documentation using Quickbook, Doxygen and Auto-Indexing. Example code with comment taken from /doxygen/html/docblocks.html using Boost naming conventions.
The style of this example takes a lot of lines, so other Boost-style examples are much more compact. They do rely on syntax coloring to make the comments stand clear and not obscure the real C++ code.

Mar 2011

Paul A. Bristow

```

class test;
```

Class test

test — A test class - a comment description that preceeds the class.

Synopsis

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


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

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

  // public data members
  int(* handler;  // A function variable.
  int public_var;  // A public variable.
};
```

Description

a class by comment that is tucked in under the class (to reduce the line count). BUT you must explicitly link to the class or it will be applied to the next member. Using a < doesn't work for classes as it does for members.

`test` public construct/copy/destruct

1.
```
test();
```

   A constructor.

   A more elaborate description of the constructor.

2.
```
~test();
```

   A destructor.

   A more elaborate description of the destructor.

   This descructor may blow up?

Requires:	No preconditions.
Postconditions:	A test object constructed, with no side effects.

`test` 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.
Requires:	No preconditions.	
Postconditions:	No side effects.	
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()
   <xrefsect>
   <xreftitle>Deprecated</xreftitle>
   <xrefdescription>

   Will be removed in the next but one version. Use test_me() instead.
   </xrefdescription>
   </xrefsect>

Parameters:	`c1`	the first argument.
	`c2`	the second argument.
Returns:	The test result.	

`test` public public data members

1.
   ```
   int(* handler;
   ```

   Details about what the function does.

   This is an implementation detail and not for user code.

2.
   ```
   int public_var;
   ```

   Details about the variable.

Type test_enum

test::test_enum — Example of documenting an enum.

Synopsis

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


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

Description

More detailed enum description. This description needs more than one line, so convenient to use C style comment markers.

`test_enum_val1`	Enum value TVal1 (Note the < to link to the same line).
`test_enum_val2`	Enum value TVal2. (Using C++ // comment markers, and < to link to the same line.
`test_enum_val3`	Enum value TVal3. (using C style comment markers).

Version Info

Last edit to Quickbook file I:\boost-sandbox\tools\quick_auto_dox_index\libs\quick_auto_dox_index\doc\quick_auto_dox_index.qbk was at 10:24:05 AM on 2011-Jul-29.

Tip

This version information should appear on the pdf version (but is redundant on html where the last revised date is on the bottom line of the home page).

Warning

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

Caution

It does not give the last edit date of other included .qbk files, so may mislead!

Class Index

Typedef Index

Function Index

Macro Index

Index