

Writing Professional Looking Documentation With Docbook, PHP, Phing and Apache FOP: Part 1: Getting Started

Introduction

Documentation. The word illicit a mix of fear and depression in even the most hardened programmer. For many it's a hard slog through endless boredom which occurs throughout, or at the end of, the development process. Documentation is never the easiest task. Good documentation takes time, patience, lots of questions about the subject matter (no matter how familiar you think you are with the subject matter, you can be assured you have some misunderstandings), and a degree of ability in condensing knowledge to a form people can instantly connect with.

But even when you get it done there's the question of how to distribute it! A popular choice is HTML - it's portable since everyone has a browser, and as web developers we're all familiar with the syntax. Another common choice is plain text since "someone else" can always transfer it to another format down the line. Some people even believe its entertaining to rely solely on inline source code comments relying on the skills of the user to decipher their personalised coding style, thought process, and intent.

This article series proposes using Docbook XML as the ultimate source format for all documentation. The difference between most formats and Docbook, is that Docbook can be used to generate numerous final formats. That flexibility and the quality of it's output go a long way in explaining Docbook's popularity among documentation authors. If you doubt it's capabilities, bear in mind there are publishers who have adopted Docbook!

The series was written to introduce programmers to a PHP oriented publishing process which uses Docbook XML as the basis for generating professional looking HTML and PDF output. I say PHP oriented, because the Unix "toolchain" commonly associated with Docbook XML has been replaced almost entirely by PHP. This is useful because with PHP's power at your disposal writing various filters to handle stuff like PHP source code highlighting is extremely simple.

Meet The Ingredients!

Docbook XML


The Docbook standard seems to have a reputation for being complex. This is an outright misconception with little foundation - the format is broad in that there are hundreds of possible tags, but shallow in that outside the tag count the rest is very straightforward. Docbook is a simple XML format where a tiny subset of the standard syntax is sufficient for 99% of your requirements. It's as simple as plain old HTML and there are several excellent editors so you're not stuck editing XML by hand (which should be avoided since

it's...painful). However I do suggest setting up the shell book (see manual.xml further on) by hand and using an editor for individual chapters/appendices since it makes life easier than putting up with one giant file!


The downside is that you have to understand at least the basic tags and be aware that as with all XML, all elements do nest, with all Docbook XML files required to validate against the standard. XML Editors rely on tag knowledge and nesting consistency - they save time because you are not writing the tags and worrying about validation, appearance and other hand editing pains.

Beyond that it's a simple exercise in learning by doing which I leave to the reader. You'll see some samples later on to give you a feel for the basic syntax. You can read a crash course over on <http://opensource.bureau-cornavin.com/crash-course/en/introduction.html> but [the full reference manual](#) is a great deal bigger and more comprehensive. There also exist several useful examples in the PHP community including the Zend Framework Reference Manual and PHPUnit's Pocket Guide, to name a few, which you can checkout from their respective version control repositories.

The reason Docbook has gotten so popular for technical manuals, reference books, and even shorter articles and some magazines is that Docbook is agnostic to the final distributable format. From any Docbook source you can generate HTML, XHTML, RTF, HTML, CHM (Microsoft Help), PostScript, TeX and FO formatted creations among others. FO is itself an intermediary XML syntax which is easily generated into PDF form using a FO Processor (like Apache FOP which we'll meet later). With all these target formats available from a single source document it's easy to see that Docbook affords you flexibility. Why write RTF or HTML, when Docbook also gives you these, and more, with minimal fuss?

Did I mention that the only required tools for Docbook processing all happen to be free and open source? 

PHP 5

Transforming Docbook XML into other formats requires a toolchain. The standard setup is to install the Docbook DTDs, the Docbook XSL stylesheets (which instruct the transformation of the source Docbook XML into varied formats), and a collection of GNU tools on Linux like xsltproc. This is sometimes referred to as the "new way" since it's reasonably uncomplicated compared to what went before in the Linux world. Reasonably is in the eye of the beholder however 

PHP5 comes with the DOM and XSL Extensions and these both come with all the functions necessary to drive a completely PHP driven toolchain for Docbook. All one needs is the toolchain programmed in PHP so it can be reused. Luckily, Phing (a project build system based on Apache Ant), includes pre-written tasks and filters which serves this purpose admirably.

I should note these were contributed for the benefit of all PHP inclined Docbook users by Bill Karwin, the former Zend Framework maestro. Thank you Bill!

The other facet of a PHP toolchain is that it enables PHP programmers to write custom Phing tasks and filters which can assist in customising output. We'll see my PHP source highlighting examples later.

Phing

Phing is one of those understated libraries eclipsed by the likes of Apache Ant or Ruby Rake in their respective languages. Phing's raison d'etre is to allow PHP programmers describe repetitive tasks in an XML syntax so they can be automatically re-run from a command-line whenever you want. For example, whenever I want to generate documentation from Docbook, I simply issue the command "phing docs" in a console!

Phing is written in PHP5, and that's the main reason I use it. You can create custom tasks and filters in PHP without fiddling with Java or embarking on an exploration of bash scripting (if you normally prefer makefiles). I use several custom tasks to fine tune the whole processing process which are basically PHP classes imported into Phing. Phing takes the pain out of applying your PHP knowledge to the task of generating and manipulating Docbook XML and any of its intermediary or final formats.

The other side of Phing is simple automation. Rather than bang on a console for two minutes, I can encode a Docbook run in Phing's XML syntax and let it automatically carry out all the tasks I defined, in the order I defined them. Two minutes over countless runs does add up and Phing removes that annoyance from my programming life. It excels at automating highly repetitive tasks.

Apache FOP

XSLT processing cannot, obviously, directly generate PDF. To get PDF documents it's necessary to have an intermediary format called XSL Formatting Objects (XSL-FO) created by an XSLT processor from the Docbook sources. This intermediary format can then itself be transformed in a second stage by a suitable XSL-FO processor into PDF, PostScript and a few other lesser used formats.

Apache FOP is chosen as the FO Processor for a few reasons. It's easy to setup and use. You can easily configure it to embed custom (or base 14) fonts into PDF. It's written in Java, accessible from the command line and runs on Windows XP/Vista with little effort. Oh, and it's free. Thou shalt not pay for a XSL-FO processor! No matter how easy the advertising promises to make it!

Installing All This Crap Without Going Insane

If your head is spinning from the deluge of information, take a break by engaging in some menial installation tasks.

PHP 5

I'm going to assume you can safely install PHP 5 without me holding your hand. Just make sure you also include PEAR! If you need to install PEAR separately consult the documentation at <http://pear.php.net>.

Phing

Installing Phing is done from the command line using PEAR. Visit [Phing's place on the web](#) where the User Manual exists if you intent attempting a non-PEAR install. Here's the usual steps needed:

```
pear channel-discover pear.phing.info
pear install phing/phing
```

You can also install one Phing task I release to my own PEAR channel to handle PHP source highlighting in HTML output.

```
pear channel-discover pear.phpspec.org
pear install phpspec/PhpDocbookHighlighterTask
```

I actually use two custom Tasks. The first highlights PHP code in HTML/XHTML documentation generated from Docbook and is found on the PHPSpec PEAR channel. The second is in the ZFBlog subversion repository at

<http://svn.astrumfutura.org/zfblog/branches/phing/PhpFoHighlighterTask.php> and should be copied to the PEAR/phing/tasks/ext/ directory of your system. This task deals with highlighting PHP source code in XSL-FO output so that in PDF content it is properly highlighted (for reference this code, like the HTML version, is licensed under a New BSD License unless otherwise stated).

Here's a copy to examine - it uses a variation of the PHP Highlighting script for HTML rewritten to apply to XSL-FO using PHP DOM:

```
<?php
require_once 'phing/Task.php';
class PhpFoHighlighterTask extends Task
{
    private $_file = null;
    public function setFile($file)
    {
        $this->_file = $file;
    }
    public function init()
    {}
    public function main()
    {
        $this->_highlightFile($this->_file);
        $this->log('PHP in XSL-FO highlighted');
    }
    private function _highlightFile($file)
    {
        $dom = new DOMDocument();
        $dom->load($file);
        $xpath = new DOMXPath($dom);
        $selements = $xpath->query("//fo:block[@phing='phpfohighlightertask']");
        foreach ($selements as $block) {
            self::_highlightBlock($block, $dom);
        }
    }
}
```

```

    $block->removeAttribute('phing');
}
$dom->save($file);
}
private static function _highlightBlock($block, $fo)
{
    $toHighlight = str_replace(
        array('&gt;', '&lt;', '&amp;', '&quot;'),
        array('>', '<', '&', '"'),
        $block->nodeValue
    );
    // This basically prevents highlighting of non
    // HTML, XML and PHP source code. Note: All PHP to
    // be highlighted this way must have <?php at the top
    if (substr($toHighlight, 5) !== '<?php'
        && substr($toHighlight, 9) !== '<!DOCTYPE'
        && !preg_match("/^[^>]*>/", $toHighlight)) {
        return;
    }
    // Why manually highlight when it's built into PHP!
    // edit php.ini or add config to change colours
    $code = highlight_string($toHighlight, true);
    $code = str_replace(
        array('<code>', '</code>', '&nbsp;', '<br />', "\n"),
        array(" ", " ", "\n", "\n"),
        $code
    );
    $code = preg_replace("!\\n\\n+!", "\n\n", $code);
    $code = trim($code);
    $dom = new DomDocument;
    $dom->loadXML($code);
    $xpath = new DomXPath($dom);
    $parentSpan = $xpath->query('/span')->item();
    $style = $parentSpan->getAttributeNode('style')->value;
    $colour = substr($style, 7, 7);
    $content = $parentSpan->nodeValue;
    $inlineParent = $fo->createElement('fo:inline');
    $inlineParent->setAttribute('color', $colour);
    $nodes = $xpath->query('/span/node()');
    foreach ($nodes as $node) {
        if ($node->nodeType == XML_ELEMENT_NODE) {
            self::_appendInlineChild($node, $inlineParent, $fo);
        } else {
            $schild = $fo->importNode($node, true);
            $inlineParent->appendChild($schild);
        }
    }
}
// Side effect of XSL-FO complexity is the odd blank monospace box
// This strips them out - sort of a workaround. Means this code could
// be improved a bit so stripping is not needed to start with!
if (preg_match("/^\s+$/", $inlineParent->firstChild->textContent)) {
    $inlineParent->removeChild($inlineParent->firstChild);
}
}

```

```

foreach ($block->childNodes as $node) {
    $block->removeChild($node);
}
$block->appendChild($inlineParent);
}
private static function _appendInlineChild($span, $inlineParent, $fo)
{
    $style = $span->getAttributeNode('style')->value;
    $colour = substr($style, 7, 7);
    $content = $span->nodeValue;
    $inlineChild = $fo->createElement('fo:inline', $content);
    $inlineChild->setAttribute('color', $colour);
    $inlineParent->appendChild($inlineChild);
}
}
}

```

With Phing installed - our PHP environment is complete. Let's now grab the remaining elements.

Apache FOP

Download Apache FOP 0.95.0 from <http://www.apache.org/dyn/closer.cgi/xmlgraphics/fop>. After you pick a distribution mirror the download is located on the path /binaries/fop-0.95-bin.tar.gz. There is also a zip archive at the same location if you prefer.

Unpack into a directory and add the distribution's /bin directory to the system PATH environment variable. Linux users should have no problem there. Windows users will need to right click My Computer and edit the PATH variable from Advanced->Environmental Variables (XP; but similar idea for Vista). You may read the Quickstart Guide (<http://xmlgraphics.apache.org/fop/quickstartguide.html>) for an overview of its operation (we'll automate this later using Phing).

Depending on your Java configuration, you may find Apache FOP runs out of memory regularly. To fix this you can edit its fop script (or BAT file) located in its base directory to manually add a higher maximum memory allowance to the Java options. I think the default on many systems is 64MB for a basic Java install and you can easily bump this up just for Apache FOP since PDF generation does need quite a chunk of RAM. To do so, you can add the following option (512MB is an arbitrary figure based on past attempts with large documentation bases):

```
-Xmx512m
```

Adding the option depends on your OS. Find this line and change to (for the Windows BAT file):

```
set JAVA_OPTS=-Denv.windir=%WINDIR% -Xmx512m
```

For Linux:


```
fop_exec_command="exec \"\$JAVACMD\" -Xmx512m
$LOGCHOICE $LOGLEVEL -classpath \"\$LOCALCLASSPATH\" $FOP_OPTS
org.apache.fop.cli.Main $fop_exec_args"
```

Docbook DTDs

All Docbook XML must be validated against the Docbook standard. This standard is represented by a set of schemas in RELAX NG, DTD and W3C XML Schema formats. These are used to validate your Docbook files to ensure their syntax is compliant before anything tries to process them. We'll be using Docbook 5 published during 2008 (you're probably more likely to see Docbook 4.x if you go looking for Docbook examples pre-existing in the PHP world like the Zend Framework Reference Manual or the PHPUnit Pocket Guide). While Docbook 5 is technically a RELAX NG reimplementation of Docbook, we will only use the non-normative DTD version with PHP 5. It's syntax is, with some exceptions and new additions, perfectly similar to later Docbook 4.x versions.

You can download the Docbook DTD files from <http://www.docbook.org/xml/5.0/docbook-5.0.zip>. Copy these files to a known location (anywhere will do so long as you remember the path!). We could reference online versions but that would slow down the entire Docbook processing stage.

Docbook XSL

In order to transform Docbook XML into varied formats including XHTML and XSL-FO, you need relevant stylesheets. Docbook XSL is the generic name given to some standardised sets which we'll use in this tutorial. Since this is XSL, it's a simple matter to add custom changes in a separate file which you will see later since I like pretty highlighted PHP source code in my documentation 

Download the current (at time of writing) version, 1.74.0, at http://sourceforge.net/project/showfiles.php?group_id=21935&package_id=16608. You can check the Docbook.org website for any updates over 1.74.0 but remember you only need download the basic XSL package - not DSSSL or the other XSL variants with the same version number.

Similar to the DTDs, store these at a known location. You'll need the path to that location later on but any location will do here.

Docbook Basics

It's easy to let Docbook syntax overwhelm you - so take my advice and stick to the basics unless a genuine need for additional syntax elements arises. The Zend Framework subversion repository has tons of examples in the most current syntax of Docbook 4.x (which is almost identical to Docbook 5 so it's a fairly solid

reference).

Everyone has their own Docbook document structure. Some use the "One Big File (TM)" approach while others take advantage of XIncludes to break a single document into much smaller chunks which are aggregated during processing. Here I follow the Zend Framework XInclude convention since it's simple to follow, and explains at least one example you can refer to!

We start by creating a new project directory - say /usr/padraic/projects/book or C:/book. Into /book we create a /docs and /build directory. Inside /docs we create the following tree of directories:

```
/docs
  /manual
    /en
      /chapters
      /html
      /images
      /references
```

Might as well get used right away to a standard which will later make multilingual documentation easy to persue!

We'll start by creating a new XML file called manual.xml at /docs/manual/en/manual.xml:


```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V5.0//EN"
  "##DOCBOOK_DTD##">
<book version="5.0" xmlns="http://docbook.org/ns/docbook"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:xi="http://www.w3.org/2001/XInclude"
  xmlns:svg="http://www.w3.org/2000/svg"
  xmlns:mml="http://www.w3.org/1998/Math/MathML"
  xmlns:html="http://www.w3.org/1999/xhtml"
  xmlns:db="http://docbook.org/ns/docbook">
<info>
  <title>Writing Professional Looking Documentation With Docbook, PHP, Phing
  and Apache FOP</title>
  <subtitle>Example Manual</subtitle>
  <author>
    <firstname>Pádraic</firstname>
    <surname>Brady</surname>
  </author>
  <pubdate><?dbtimestamp format="Y-m-d"?></pubdate>
  <copyright>
    <year><?dbtimestamp format="Y"?></year>
    <holder>Pádraic Brady</holder>
  </copyright>
  <legalnotice>
    <para>
      This work is licensed under the Creative Commons Attribution 3.0 License. To view a copy of
```

this license, visit <http://creativecommons.org/licenses/by/3.0/> or send a letter to Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.

```
</para>
</legalnotice>
</info>
<!-- Include Chapters and Appendices -->
<xi:include href="chapters/Introduction.xml" />
<xi:include href="references/copyrights.xml" />
</book>
```

This is what I call the base file. It's the file which defines the overall layout of the manual or book by setting the order in which chapters, appendices and other documents are presented in the final distributable format.

You'll immediately notice one oddity, "##DOCBOOK_DTD##". This is a placeholder tag. When processed, we'll get Phing to first replace this tag with the real location of the Docbook 5.0 DTD on our system. It's configurable and our Phing properties file will later default to an online version (which is not preferred since it slows everything down!).

At the bottom we also see two XInclude references to other Docbook files on relative paths. These will be included into the manual.xml content during processing in much the same way as PHP includes operate (C level mechanics aside 

). You'll also see a reference to "dbtimestamp" which is merely an instruction to the processor, enclosed in PHP-like short tags, to insert the current date in the specified format. Just remember this is NOT PHP though! If you need to perform any PHP operation on the Docbook sources you should look into writing some custom Phing tasks or filters like my PhpFoHighlighterTask example from earlier.

Take a moment to get used to the format - it's quite simple and there is not much more that any Docbook formatted Manual would need to add.

Let's add a simple Introduction chapter at /docs/manual/en/chapters/Introduction.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<chapter version="5.0" xml:id="introduction"
  xmlns="http://docbook.org/ns/docbook"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:xi="http://www.w3.org/2001/XInclude"
  xmlns:svg="http://www.w3.org/2000/svg"
  xmlns:mml="http://www.w3.org/1998/Math/MathML"
  xmlns:html="http://www.w3.org/1999/xhtml"
  xmlns:db="http://docbook.org/ns/docbook">
  <title>Introduction</title>
  <section xml:id="introduction.why.docbook">
    <title>Why Docbook?</title>
    <para>The Docbook standard seems to have a reputation for being complex.
    This is a lie. Docbook is a simple XML format where a tiny subset of the
```

standard syntax is sufficient for 99% of your requirements. It's as simple as plain old HTML and there are several excellent editors so you're not stuck editing XML by hand.</para>

<para>The downside is that you have to understand at least the basic tags and be aware that as XML, all elements do nest. Beyond that it's a simple exercise in learning by doing which I leave to the reader. You'll see some samples later on to give you a feel for the basic syntax. You can read a crash course over on the <uri

xlink:href="http://opensource.bureau-cornavin.com/crash-course/en/introduction.html" xlink:type="simple">Docbook Crash Course</uri> but the full reference manual is a great deal bigger and comprehensive.</para>

<para>The reason Docbook has gotten so popular for technical manuals, reference books and anything published by O'Reilly :), is that Docbook is format agnostic. From any Docbook source you can generate HTML, XHTML, RTF, HTML, CHM (Microsoft Help), PostScript, TeX and FO formatted creations (with a few others less likely to be seen for major documentation). FO is itself an intermediary XML syntax which is easily generated into PDF form using a FO Processor (like Apache FOP which we'll meet later). With all these target formats available from a single source document it's easy to see that Docbook affords you flexibility. Why write RTF or HTML, when Docbook also gives you all the others with minimal fuss?</para>

<para>It even handles source code:</para>

<programlisting><?php

```
class Foo {
    public $bar = 'Bar';
    public function echo()
    {
        echo $this-&gt;bar;
    }
}
$foo = new Foo;
$foo-&gt;bar = "Hello, World!";
$foo-&gt;echo();</programlisting>
</section>
</chapter>
```

Here, the simplicity of Docbook is revealed. All content is enclosed in Sections, inside Chapters (or Prefaces, Appendices, etc). Sections actually have varying importance (a bit like how H1, H2, H3 tags work in HTML) depending on their nesting depth. In practice, for HTML, each top-level section is assumed to be a single page within that chapter group. PDF doesn't care about this, however, since it's a continuous document format. Paragraphs of content are enclosed in PARA tags (same idea as HTML's P tag). I gave an ID to the Chapter and Section elements. This doesn't add anything for PDF, but it can be used to dictate HTML file names when generating chunked (multi-page) HTML output.

While we're here let's add /docs/manual/en/references/copyrights.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<appendix version="5.0" xml:id="copyrights"
```

```
xmlns="http://docbook.org/ns/docbook"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xi="http://www.w3.org/2001/XInclude"
xmlns:svg="http://www.w3.org/2000/svg"
xmlns:mml="http://www.w3.org/1998/Math/MathML"
xmlns:html="http://www.w3.org/1999/xhtml"
xmlns:db="http://docbook.org/ns/docbook">
<title>Copyright Information</title>
<section xml:id="copyrights.copyright">
  <title>Copyright</title>
  <para>Copyright © 2008, Pádraic Brady. All rights reserved.</para>
</section>
<section xml:id="copyrights.licensing">
  <title>Licensing</title>
  <para>This work is licensed under the Creative Commons Attribution 3.0
  License. To view a copy of this license, visit
  http://creativecommons.org/licenses/by/3.0/ or send a letter to Creative
  Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.</para>
</appendix>
```


Et voila! We have one complete Docbook XML formatted book ready for processing into HTML and PDF for online distribution.

To reiterate an earlier point, if you have [XMLmind XML Editor](#) (one of those impressively useful Docbook editors) installed try opening the Introduction.xml file and doing some editing. Editors for XML take a bit of getting used to since you still need to tell them which tags to insert (see the panel of tag names on the right, with tag attributes listed on bottom right of the XMLmind editor) but it's miles easier than doing it by hand, checking validity, managing text column width for the source code, etc.

In Conclusion

In Part 2 we'll cover Phing and how document generation actually works. For now though you should have the stage set and at least have a rudimentary understanding of what Docbook XML is and why it's a popular choice for technical documentation.

There's a lot of information above to absorb and noted links to take a quick look at. Don't worry if you don't understand everything since tomorrow's Part 2 will bring it all together by using what we learned today to create a PDF example.

If you have specific questions however, don't hesitate to comment! 

Posted by Pádraic Brady in PHP Security at 01:13

Thank you for another great series!

BTW, you forgot to close in Copyright.xml Anonymous on Nov 18 2008, 00:40


Consider using latex (or other forms of Tex) for writing books. It is very powerful and flexible. The syntax is also very simple and less angle bracket tax. And probably has the best text layout engine there is (by Donald Knuth). Yes, you should care about text layouts.

If sticking with docbook, CDATA will help to remove the ugly entities such as > Anonymous on Nov 27 2008, 19:56

LaTeX is excellent. My needs however converged on a PHP manipulated format and Docbook fits the bill with it's shallow learning curve (everyone knows XML) and simple automation/customisation of outputs via XML manipulation using Phing and PHP DOM.

If anyone wants to try out LaTeX there are free editors for every platform (yes, even Windows!). Anonymous on Nov 28 2008, 02:45

Your terrific piece on docbook (part 1) got me all fired up and ready to go with the new toolchain, php based and all. But then silence! What ever happened to to part 2??? Anonymous on Jan 12 2009, 06:41

Don' worry - I'll get to it soon 

Anonymous on Jan 12 2009, 08:11

Interesting. I'm looking forward to trying this out to help expand our documentation sources at work. Thanks for the information.
Anonymous on Jan 13 2009, 11:29