DocBook: Write Once, Read Anywhere Documentation

DocBook enables developers to create technical documentation in XML. Learn how to use it to transform its XML output to PDF and HTML automatically and integrate it into your development environment.

by Lara D'Abreo

DocBook is an OASIS standard for creating technical documentation in XML. The documentation from Spring or Hibernate is generated from DocBook. Particularly suited for computer-related content, DocBook is a set of XML tags, defined by a Document Type Definition (DTD) and XML schema for technical content. In addition to the DTD, DocBook and other open source projects supply a collection of tools and frameworks that enable developers to transform DocBook-compliant XML into PDF, HTML, Eclipse Help, and even MAN pages. This alleviates the need to write the same material multiple times or manually convert from one format to another.

No big deal, you say. *I can write HTML, Eclipse Help, and PDF if I have to. Why is DocBook different?* In the DocBook model, you write the raw documentation once in XML and then transform or "compile" it to the desired target formats using DocBook tools—a paradigm familiar to developers, who already code and compile. So DocBook is ideal for developers who have to write documentation, not only because it is targeted for technical content but also because DocBook files—like any other XML files—can be managed and edited within an IDE. No longer is documentation someone else's problem; it's right there next to your code. This encourages you keep the software documentation in sync as you develop.

Because the DocBook XML structure is focused on structure and content rather than presentation, you do not have to worry about formatting and presentation aspects. The XML transformation tools and style sheets take care of these details. The transformation tools ensure a uniform look and feel to the generated documentation, and you can customize it separately from the raw content.

This article introduces DocBook and explains how to use it for the following:

1. Writing a simple document
2. Transforming it into PDF and HTML using the Velocity DocBook framework
3. Automatically interleaving source code extracts into your documentation

DocBook Introduction

A DocBook-compliant XML file doesn't require any special documentation tools—any XML editor or IDE plug-in will do, provided it can access the DocBook DTD to understand the DocBook tags and validate the XML. Several free and commercial WYSIWYG Eclipse plugins and standalone editors are available, but any standard XML editor or IDE plug-in, such as the ones that come with WTP or MyEclipse, will suffice.

Once the documentation is in the development environment, you can seamlessly integrate it into the development processes. Your DocBook artifacts can be:

- Placed under version control so that they can be checked-in, checked-out, tagged, and branched with releases
- Edited within your development environment using standard IDEs and readily available XML editor plug-ins
- Kept in sync with code and included in any code refactoring. When a classname changes so do the references in the documentation
- Integrated with your continuous and nightly build processes so that the documentation distributions (PDF, HTML, etc.) are built automatically

There are many tags in DocBook, such as <book>, <chapter>, and <title>, as well as more programming-specific elements such as <classname>, <programlisting>, and <database>. However, you don't need to know them all to get started—just a few basic elements. A simple DocBook XML file is comprised of a <Book>, which contains one or more <chapter> elements. A book is a single deliverable documentation artifact with chapters that correspond to logical sections within your end document. Chapters are hierarchical; a chapter can contain other chapters, and they can be nested indefinitely.

The following is a simple DocBook XML document that you will use as an example in this article. It contains a title, bookinfo, a table of contents, and two chapters, each containing some text:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.4//EN"
"http://www.oasis-open.org/docbook/xml/4.4/docbookx.dtd">
<book >
  <title>DocBook Framework Overview</title>

  <bookinfo>
    <releaseinfo>V 1.0</releaseinfo>
    <author>devx
    </author>

  </bookinfo>

  <toc/>

  <chapter id="intro">
    <title>Introduction</title>
    <para>DocBook is simple to use and provides a rich set of tags for common elements.</para>
  </chapter>

  <chapter id="basics">
    <title>Basic Elements</title>
    <para>This section describes the basic tags.</para>
  </chapter>

</book>
```

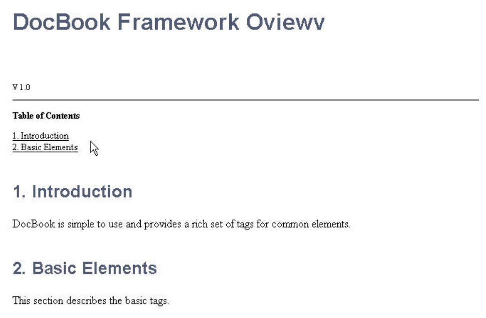
Note the DTD reference at the top. This reference must be present for your XML editor to understand and validate your XML against the DocBook structure and provide content assistance.

Transforming to HTML and PDF
Apache Velocity provides a framework for transforming DocBook XML files into HTML and PDF. Velocity uses a set of ANT build files for these transformations. For this article, download the Velocity DocBook framework and follow the installation instructions. In an IDE such as Eclipse, create a new project and import the velocity framework. You will also need the Xerces library. Download it from xerces.apache.org and copy it into the lib folder of your new project, alongside the third party jars. When this is done, your project should contain the Velocity tools as well as the Velocity documentation (written using DocBook) underneath the docs/src/docbook/dbf folder. You will be creating your own sample documentation here. Begin by creating the simple DocBook XML document from the previous page in this folder as simple.xml.

To transform simple.xml into PDF and HTML you will need to edit the Velocity build script to pick up the new file. Long term you can tailor the project structure and ANT scripts to suit your project and integrate more tightly with your build processes, but for this demonstration open the ANT file build.xml and change the entry from `<property name="docbook.file" value="DBFUserGuide"/>` to `<property name="docbook.file" value="simple"/>`. This ensures that it points to your new file.

To generate the PDF and HTML in Eclipse simply right-click on the build.xml file and select Run As –> ANT Build. When the build is complete, navigate to the doc/target/dbf/singlehtml folder to see the HTML output and doc/target/dbf/pdf to see the PDF output.

The resulting documentation looks like Figure 1 (HTML) and Figure 2 (PDF).



DocBook Framework Oviewv

V 1.0

Table of Contents

1. Introduction
2. Basic Elements

1. Introduction

DocBook is simple to use and provides a rich set of tags for common elements.

2. Basic Elements

This section describes the basic tags.

Figure 1. Simple.xml Rendered to HTML

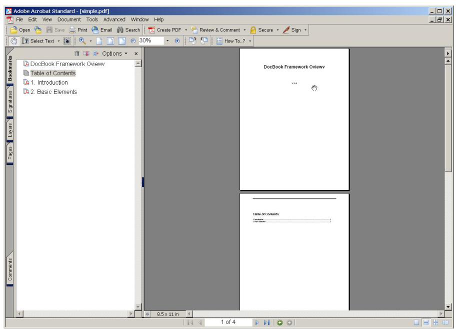


Figure 2. Simple.xml Rendered to PDF

The DocBook Vocabulary
The DocBook vocabulary contains a rich set of tags for different content, including basic lists, tables, links, and images, as well as more specific programming constructs. To create an itemized list use a <orderedlist> or <itemizedlist> tag with <listitem> child elements as follows:

```
...
<chapter id="list">
  <title>Lists</title>
  <para>You can use:</para>
    <itemizedlist mark="opencircle">
      <listitem>
        <para> itemizedlist; or </para>
      </listitem>

      <listitem>
        <para> orderedlist </para>
      </listitem>

    </itemizedlist>
</chapter>
```

You can incorporate images and links using the <mediaobject> and <ulink> tags. For example, you'd use this code to include the image logo.png:

```
<mediaobject>
  <imageobject>
    <imagedata fileref="images/logo.png"/>
  </imageobject>
</mediaobject>
```

And you'd use this code to add a link to the DocBook web site:

```
<ulink url="http://www.docbook.org">docBook</ulink>
```

To include source code use the <programlisting> element:

```
<programlisting><![CDATA[
  protected void test() {
    System.out.println("This is a sample extract");
  }
...]]>
</programlisting>
```

Experiment with a few tags by adding them to the simple.xml file and running the ANT build to see how they render.

Splitting Your Book

For a large document, putting it all in one XML file can quickly become unmanageable—especially if it is being edited by multiple people. One way to overcome this is to break up your document into smaller files, each dedicated to a chapter or a group of logically related chapters. This way, multiple people can edit different parts of the document without affecting any other parts. And you know when you've made a mistake because the DocBook build will fail.

You also can factor common chapters out into separate files so that they can be reused in different books. For example, you may want to include the standard product copyright section at the top of your training manual as well as in your installation guide. Each document is a separate book, but they contain some common content. DocBook files can be included in other DocBook files using <xi:include>. For example, to structure your sample document into a book file and two chapter files, change the main file simple.xml to this:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.4//EN"
"http://www.oasis-open.org/docbook/xml/4.4/docbookx.dtd">
<book >
  <title>DocBook Framework Overview</title>

  <bookinfo>
    <releaseinfo>V 1.0</releaseinfo>
    <author>devx
    </author>

  </bookinfo>

  <toc/>

  <xi:include href="intro.xml" xmlns:xi="http://www.w3.org/2003/XInclude" />
  <xi:include href="basics.xml" xmlns:xi="http://www.w3.org/2003/XInclude" />


</book>
```

And decompose each of the chapter elements into intro.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.4//EN"
  "http://www.oasis-open.org/docbook/xml/4.4/docbookx.dtd">
<chapter id="intro">
    <title>Introduction</title>
    <para>DocBook is simple to use and provides a rich set of tags for common elements.</para>
</chapter>
```

and basics.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.4//EN"
  "http://www.oasis-open.org/docbook/xml/4.4/docbookx.dtd">
<chapter id="basics">
    <title>Basic Elements</title>
    <para>This section describes the basic tags.</para>
</chapter>
```

Note that since each chapter resides in its own file, if you want to be able to edit it using an XML editor or plugin it must also reference the correct DTD. To enable the Velocity framework to handle multiple files per book, ensure that Saxon (the underlying XSLT engine used by Velocity) is configured to correctly resolve <xi:include> elements. Edit the build-docbook.xml file and modify the Saxon entry to enlist xerces to handle the <xi:include> elements:

```
<java classname="com.icl.saxon.StyleSheet" fork="true"
    dir="${basedir}" classpathref="dbf.classpath">
        <jvmarg value="-Djavax.xml.parsers.DocumentBuilderFactory=
                org.apache.xerces.jaxp.DocumentBuilderFactoryImpl" />
        <jvmarg value="-Djavax.xml.parsers.SAXParserFactory=
                org.apache.xerces.jaxp.SAXParserFactoryImpl" />
        <jvmarg value="-Dorg.apache.xerces.xni.parser.XMLParserConfiguration=
                org.apache.xerces.parsers.XIncludeParserConfiguration" />
    <arg line="-x org.apache.xml.resolver.tools.ResolvingXMLReader"/>
    <arg line="-y org.apache.xml.resolver.tools.ResolvingXMLReader"/>
    <arg line="-r org.apache.xml.resolver.tools.CatalogResolver"/>
    <arg value="-o"/>
    <arg value="@{output}"/>
    <arg value="@{input}"/>
    <arg value="@{style}"/>
</java>
```

Incorporating Source Code Using <xi:include>

The <xi:include> element is useful for not only structuring your book but also for including source code. One of the common challenges of technical documentation—especially in agile environments—is keeping it in sync with the code. Sample source code in the documentation can get old fast. Source code that is incorporated manually using copy/paste is error prone, hard to test, and likely to get out of date quickly unless someone is willing to continually maintain the code samples. DocBook offers another way.

You can place code that you want included in your documentation into unit tests where it can be automatically tested, refactored as normal, and then mark it up for inclusion in DocBook. It's a simple process to create an extraction program that can crawl the codebase for markup and extract the annotated code into DocBook XML files. You then can include these files, like any other DocBook files using <xi:include>.

To see <xi:include> in action, create a sample Java class called SampleClass.java. Mark up the method test() using Java comments: //@extract-start <extractname> to mark the beginning of the extract and //@extract-end <extractname> to mark the end:

```java
public class SampleClass {

    //@extract-start test
    protected void test() {
        System.out.println("This is a sample extract");
    }
    //@extract-end test
}
```

The markup can then be processed by an extractor program, such as the one included in the code download for this article, so that the code inside test() can be extracted into the DocBook XML file test.xml. The name of the file is specified by the markup tags.

Test.xml contains a <programlisting> element with the annotated Java inside:

```
<?xml version="1.0" encoding="UTF-8"?>
<para>
<programlisting><![CDATA[
    protected void test() {
        System.out.println("This is a sample extract");
    }
...]]>
</programlisting>
</para>
```

The following code shows how you would include a <programlisting> element in a chapter:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.4//EN"
  "http://www.oasis-open.org/docbook/xml/4.4/docbookx.dtd">
<chapter id="programlisting">
    <title>including source code</title>
    <xi:include href="test.xml" xmlns:xi="http://www.w3.org/2003/XInclude" />
</chapter>
```

The advantage of this approach is that you can maintain the code in once place—the source files where it belongs—and unit test it as part of your build. Code included in your documentation is guaranteed to be correct if your unit tests pass. Extraction, along with the DocBook ANT build scripts, can be integrated using an ANT task to ensure that your documentation is always generated and always up to date with your code.

DocBook in Perspective
DocBook is an ideal strategy for technical, developer-centric documentation because it enables you to work closely with documentation in the same way you would your code. Popular open-source projects such as Spring and Hibernate use DocBook to generate their documentation. However, as with all technologies it has its downside: it requires a bit of up-front investment to learn how to use XML and the DocBook tags and how to integrate the rendering tools and style sheets into your build processes. You can use other standards such as DITA, also based on XML, as alternatives if they are easier for you though.

Of course, DocBook won't make poorly written documentation better. However, by integrating it into your development environment and leveraging the many DocBook resources available on the Web, you can evolve your documentation with your code—keeping it in sync, even in agile environments—and ensure that you get a professional-looking result at the end, and in the distribution format of your choosing.

***Lara D'Abreo** is an independent consultant with over 10 years experience in commercial product development in the US, Japan, and the UK. She's currently based out of Sydney, Australia and spends her time trying to make J2EE systems run faster.*