

Producing documentation and reusing information in XML, Part 1: Document publishing using XML

Create, format, and publish documents using XML standards and open source tools

Skill Level: Intermediate

[William von Hagen \(wvh@vonhagen.org\)](mailto:wvh@vonhagen.org)
Systems Administrator, Writer
WordSmiths

24 Mar 2009

Updated 07 Jul 2009

XML provides a way to identify data items and subcomponents within any structured data set, but has its roots in documentation development and production. Robust, open standards for XML document markup and a rich set of freely available tools for XML document parsing and format conversion make it easy to install and configure a complete documentation development and formatting environment on any UNIX® or Linux® system.

XML, markup, and documentation

XML was designed as a mechanism to identify structured content within a data set by wrapping different hierarchical portions of that data within text tags that identify the role of the text that they delimit within that hierarchy. These text tags, known as markup, tags, or, more properly, elements, conform to a predefined structure. That structure is specific to different types of data and is known as a schema or, historically, a DTD. Schemas are written in XML, while DTDs have their own syntax.

Frequently used acronyms

- DTD: Document Type Definition
- HTML: Hypertext Markup Language
- XHTML: Extensible Hypertext Markup Language
- XML: Extensible Markup Language
- XSL: Extensible Stylesheet Language

The idea of markup languages, and therefore XML itself, has its roots in document production systems where text files contained processing instructions or structural identifiers for different portions of the document. These text files could then be processed by software that knew how to interpret these instructions and produce output that was formatted for a specific output device. The evolution of markup languages for documentation has added increasing amounts of abstraction to how markup is used.

Early markup languages used formatting instructions that specified low-level details such as font and font size changes. These quickly evolved into more generic markup that identified logical and structural components such as highlighted terms, paragraphs, lists, headings, and so on.

More modern markup languages, such as the Standard Generalized Markup Language (SGML) and XML, popularized more abstract structural markup such as books, chapters, sections, and so on. SGML also added the notion of a DTD, which enabled SGML tools to verify conformance to specific sets of markup elements and the contexts in which those elements can appear within the structure of a specific type of document.

Other articles in this series

- [Part 2: Reuse information in XML documentation: Reduce work, increase usability, and enforce consistency with document reuse](#)
- [Part 3: Creating multi-target XML documents: Single-source documents for multiple audiences and output formats](#)

XML explicitly freed markup from the realm of documentation, expanding its use to any type of structured data and adding a requirement for *well-formed* content, where both opening and closing tags for each element must be present. Though the initial versions of the XML specification inherited DTDs from SGML, later versions of the specification introduced the use of schemas to define the structure of XML data. Schemas are equivalent to DTDs but are themselves written in XML and provide a more flexible model for defining valid elements, their data types, and the context in which they can appear in an XML document.

The idea of separating the content of a document (the actual words or data that it contains) from its presentation (the way in which it is displayed in a specific output format or on a specific output device) is the key idea behind modern markup languages and, specifically, SGML and XML.

Though XML makes it easy to devise DTDs and schemas for specific applications and industries, one of its big advantages in the documentation space is the existence of a number of standard DTDs and schemas for documentation markup. The best known of these are DocBook, Text Encoding Initiative (TEI), XHTML, and the schemas and DTDs that are used by the Darwin Information Typing Architecture (DITA). This article focuses on DocBook, which is the best-known and most widely used schema for documentation markup.

Overview of the publishing process for XML documents

Because of its focus on separating content from presentation, XML is inherently a "write once, read many" mechanism that makes it easy to reuse information in multiple documents or customize information for a specific output format or audience. The other articles in this series will explore these topics in more detail.

In the context of this article, the term *publishing* refers to converting a document from a text file that contains DocBook markup to one or more output files in a specific presentation format, such as HTML, Adobe®'s Portable Document Format (PDF), PostScript, Microsoft®'s Rich Text Format (RTF), and so on. Generating HTML output from XML documents is a one-step process that uses a single application to generate HTML output. The process for producing PDF, PostScript, or RTF output requires only one additional step:

1. Resolve any included files or external references in the DocBook document and use an XSL stylesheet to generate a version of that document in Formatting Object (FO) format.
2. For PDF, PostScript, or RTF output, convert the FO version of the DocBook document into an output file in the specified format.

Many open source tools are available to perform these conversions, but this article focuses on the three that I've found to provide the most useful and most flexible solutions. [Table 1](#) describes these tools.

Table 1. Open source conversion tools covered in this article

Tool	Description
fop	A Java™ application that converts XML input to output formats such as PDF, PostScript, RTF, and so on. You must install a Java Virtual Machine (JVM) on your system to execute this

	application.
<code>xmlto</code>	A simple Bourne-Again shell script and set of related stylesheets that produce HTML output from XML input. This script requires some other open source utilities, such as GNU <code>find</code> and <code>mktemp</code> .
<code>xsltproc</code>	A compiled application that uses XSL stylesheets to transform XML input into another XML output format. To do this transformation, you also need to install the DocBook XSL stylesheets. While <code>fop</code> can accept DocBook XML input given an appropriate stylesheet, using a standalone XML to FO converter such as <code>xsltproc</code> simplifies debugging if there are problems in your DocBook input files.

The next few sections explain how to download and install these applications. A [subsequent section](#) provides a shell script that ties them all together and provides some convenient options for specifying output formats, custom stylesheets, and so on.

Getting and installing required software

See [Resources](#) for information about where to obtain the packages that provide the software and stylesheets discussed in the [previous section](#).

Note: If you use a Linux system, there's a good chance that the `xmlto` and `xsltproc` utilities are already installed on your system or are at least available from the repositories or installation media for your distribution. You must also install an actual JVM to run `fop`—the GNU Compiler for Java does not provide all the capabilities required by `fop`.

Many of these packages install software in specific locations so that your system's package management software can monitor and update the software as needed. This is not true of the DocBook XSL stylesheets and Apache Formatting Objects Processor (FOP), which are downloaded as archive files that contain a single top-level directory and various subdirectories.

To unify a documentation parsing and formatting environment, it's a good idea to install any packages that do not have hard-wired pathnames in a single directory that does not already exist, such as `/home/doc`. This provides a single location where you can look for related files, upgrade software as needed, and collect additional tools and stylesheets that you may develop or accumulate.

For the packages discussed in the [previous section](#) to work with the unifying script that is provided in the next section, execute the commands in [Listing 1](#) as the root

user on your UNIX or Linux system.

Listing 1. Commands to execute so fop, xmlto and xsltproc work with the unifying script

```
mkdir /home/doc
mkdir /home/doc/bin
mkdir /home/doc/external
mkdir /home/doc/external/SAVE
mkdir /home/doc/XSL
```

After downloading the compressed (.zip) files for FOP and the DocBook XSL stylesheets, place them in the /home/doc/external directory, change to that directory, and extract their contents there, using the commands in [Listing 2](#).

Listing 2. Commands to extract FOP and the DocBook XSL stylesheets

```
cd /home/doc/external
unzip fop-0.95-bin.zip
unzip docbook-xsl-1.74.3.zip
chmod 755 fop-0.95/fop
```

To verify that all the necessary packages are installed on your system, execute the commands in [Listing 3](#).

Listing 3. Commands to verify all the necessary packages are installed

```
xmlto --version
xsltproc --version
java --version
```

If the `xmlto` and `xsltproc` applications and a JVM are correctly installed on your system and are in your execution path, each of these commands returns version information about that application.

Writing a wrapper script for publishing documents

Now that all the software you need for DocBook publishing is installed on your system, all you have to do is remember all the necessary options and execute things in the correct order. Alternately, you can write a simple wrapper script that ties things together correctly and provides some control over the publishing process, like the one shown in [Listing 4](#), [Listing 5](#), and [Listing 6](#).

This script generates HTML, PDF, or RTF output based on the arguments that you supply. When generating PDF or RTF output, it uses a two-step process to produce the PDF output, first executing `xsltproc` to generate the FO file from your DocBook input, using the XSL stylesheet identified in the `driverfile` variable to identify the conversion rules. The wrapper script then uses the `fop` application to

generate the target output format from that file. Each of these steps captures messages from the applications that are used (`xsltproc` and `fop`) in files called `xsltproc.out` and `fop.out`. The script uses these intermediate files to monitor the conversion process and capture potential debugging information if problems arise.

[Listing 4](#) shows the beginning of the sample wrapper script, which defines a usage message that displays if you execute the script with no arguments or the wrong number of arguments. It also sets some default values for the script, and then parses the command-line arguments using the standard Bash shell's `getopt` function. ([Listing 7](#) shows the output of this usage message.)

Listing 4. Initialization section of the simple wrapper script

```
#!/bin/bash
#
# Simple script to generate HTML output from a DocBook file using
# xsltproc, or to generate PDF/RTF output by generating a fo file,
# and then using Apache's FOP to format that.
#
# wvh@vonhagen.org
#

function usage {
    echo "Usage: $0 [OPTIONS] document-name"
    echo "  -D : produce draft mode document"
    echo "  -d : use specified xsl driver (requires name of XSL file) "
    echo "  -h : generate HTML output (requires output directory name)"
    echo "  -k : attempt PDF generation even if xsltproc errors were encountered"
    echo "  -n : suppress TOC"
    echo "  -r : produce RTF output rather than PDF"
    exit 1
}

dir=`dirname $0`

XSLTPROC="xsltproc"
PATH=$dir/../../external/fop-0.95:${PATH}

if [ $# == 0 ] ; then
    usage
    exit
fi

driverfile="$dir/../../external/docbook-xsl-1.74.3/fo/docbook.xsl"
# driverfile="$dir/../../XSL/generic-print-driver.xsl"
draft="no"
keepgoing="no"
RTFoutput="no"
HTMLoutput="no"

while getopts ":d:h:Dknr" opt
do
    case $opt in
        d )
            driverfile=$OPTARG
            echo "Using driver file: $driverfile"
            if [ ! -f $driverfile ] ; then
                echo "  Driver file not found"
                exit 1
            fi
            ;;
        D )
            draft="yes"
    esac
done
```

```

        echo "Producing Draft Mode Document"
        ;;
    h )
        HTMLdir=$OPTARG
        HTMLoutput="yes"
        echo "Producing HTML output to $HTMLdir instead of PDF"
        if [ -d $HTMLdir ] ; then
            echo " Output directory $HTMLdir already exists"
            exit 1
        fi
        ;;
    k )
        keepgoing="yes"
        echo "Will not stop if errors are encountered"
        ;;
    n )
        notoc="--stringparam generate.toc ''"
        echo "Suppressing TOC..."
        ;;
    r )
        RTFoutput="yes"
        echo "Producing RTF output rather than PDF"
        ;;
    \? )
        usage
        exit 1;;
esac
done
shift $(( $OPTIND - 1 ))

if [ ! -f $1 ] ; then
    echo " Input file $1 not found - exiting."
    exit
fi

```

[Listing 5](#) shows the portion of the script that uses the `xmlto` application to generate HTML when producing HTML output or the `xsltproc` application to create FO output from the input XML file if you are producing PDF or RTF output. The output from the XML to FO conversion step is written to a temporary file named `xsltproc.out`, which simplifies debugging if the `xsltproc` application encounters errors in your input file. If any variation of the word *error* is detected in this file, the script exits after it generates the FO file. You can then examine the `xsltproc.out` file, which should help you identify and resolve the problem. As in [Listing 4](#), you can also specify the `-k` command-line option to force the script to continue even if some errors are detected during `xsltproc` processing.

Listing 5. Generating FO output from the input XML file

```

doc_base=`basename $1 .xml`

echo "Processing base name: $doc_base"

if [ x$HTMLoutput != "xno" ] ; then
    xmlto -o $HTMLdir html $1
    exit
fi

echo " Generating FO file: $doc_base.fo"

$XSLTPROC --output $doc_base".fo" \

```

```

--stringparam draft.mode $draft \
--stringparam fop.extensions 1 \
--xinclude \
$notoc \
$driverfile \
$1 1> xsltproc.out 2> xsltproc.out

probs=`grep -i error xsltproc.out | wc -l | sed -e 's; ;g'`

if [ x$probs != "x0" ] ; then
    echo ""
    echo "PROBLEMS:"
    echo ""
    cat xsltproc.out
    echo ""
    if [ x$keepgoing = "xno" ] ; then
        exit
    fi
fi

echo "    Fixing FO file references to system images..."
cat $doc_base".fo" | sed -e "s;\\"url(\.\\.\\.external/;"url($dir/\\.\\.\\.external/;g" > $$
mv $$ $doc_base".fo"

```

Listing 6 shows the last stage of the script, which uses the `fop` application to generate PDF or RTF output from an FO file. As in the XML to FO conversion process, the output from the FO to PDF/RTF conversion step is written to a temporary file named `fop.out`, which simplifies debugging if the `fop` application encounters errors in the FO file. If PDF/RTF generation fails, you can then examine the `fop.out` file, which should help you identify and resolve the problem. As in [Listing 4](#), you can also specify the `-k` command-line option to force the script to continue even if some errors have been detected during `xsltproc` processing.

Listing 6. Generating PDF or RTF output from the FO file

```

if [ x$RTFoutput = "xno" ] ; then
    echo "    Generating PDF file: $doc_base".pdf"
    fop -fo $doc_base".fo" -pdf $doc_base".pdf" 1> fop.out 2> fop.out
else
    echo "    Generating RTF file: $doc_base".rtf"
    fop -fo $doc_base".fo" -rtf $doc_base".rtf" 1> fop.out 2> fop.out
fi

probs=`grep -i Exception fop.out | wc -l | sed -e 's; ;g'`

if [ x$probs != "x0" ] ; then
    echo ""
    echo "EXCEPTION OCCURRED DURING PROCESSING:"
    echo ""
    cat fop.out
    echo ""
fi

```

You can download a copy of this sample script, called `xml-format.sh` (see `xml-format.zip` in [Downloads](#)). To install this script on your system, download the zip file and extract `xml-format.sh` to the `/home/doc/bin` directory. Use the following command to make sure the script is executable:

```
chmod 755 /home/doc/bin/xml-format.sh
```

Finally, make sure that the `/home/doc/bin` directory is in your execution path by editing your `~/.bashrc` file and adding the following line at the end of that file or by executing that same line as a command at a shell prompt on your system:

```
export PATH=/home/doc/bin:${PATH}
```

If you add this command to your `~/.bashrc` file, source that file in your current login session to have your changes take effect in the current shell. To do this, execute the following command:

```
source ~/.bashrc
```

Publishing a DocBook document

Executing the `xml-format.sh` script with no arguments displays summary information about how the script should be used, as in [Listing 7](#).

Listing 7. Output from `xml-format.sh` script when executed with no arguments

```
Usage: xml-format.sh [OPTIONS] document-name
-D : produce draft mode document
-d : use specified xsl driver (requires name of XSL file)
-h : generate HTML output (requires output directory name)
-k : attempt PDF generation even if xsltproc errors were encountered
-n : suppress TOC
-r : produce RTF output rather than PDF
```

To generate the PDF version of a DocBook document, simply execute the script with the name of that document, as in the following example:

```
xml-format.sh sample-document.xml
```

To generate the HTML version of a DocBook document using this script, you must use the `-h` option and specify the name of the destination directory for the HTML output files. For example, to create an HTML version of a DocBook document in the `HTML-output` directory (relative to your current working directory), execute the following command:

```
xml-format.sh -h HTML-output sample-document.xml
```

Adding an XSL customization layer for PDF output

By default, the `xml-format.sh` script that is provided with this article uses the default output stylesheets provided in the DocBook XSL package. To add your own customizations, you can create your own stylesheet that first loads the default DocBook XSL stylesheet and then provides custom values for any parameters that you want to override. An example of a customization layer is provided in the `generic-print-driver.xsl` stylesheet (see `generic-print-driver.zip` in [Downloads](#)).

To use a different stylesheet with the `xml-format.sh` script, you can either use the `-d` option to specify the full or relative pathname name of the stylesheet that you want to use or you can modify the value of the `driverfile` variable in the `xml-format.sh` script. The sample script includes a line that enables you to use the example `generic-print-driver.xsl` script. Simply comment out the preceding value for the `driverfile` variable and uncomment the following line, which contains the new definition.

Summary

The power and flexibility of XML, sets of existing standards, and a rich set of tools for working with and converting XML documents makes it easy to install and configure your own set of tools to generate output in various formats from XML documents. Though many commercial tools are available to edit and format XML documents, creating your own set of tools to do this helps you better understand the process, and you can't beat the price.

Downloads

Description	Name	Size	Download method
Sample DocBook formatting script	xml-format.zip	2KB	HTTP
Sample XSL customization for DocBook	generic-print-driver.zip	1KB	HTTP

[Information about download methods](#)

Resources

Learn

- [Producing documentation and reusing information in XML, Part 2: Reuse information in XML documentation](#) (William von Hagen, developerWorks, March 2009): Structure XML documents and fragments for reuse. With XInclude, include external document sections, and with XPointer, include small document fragments into a larger document in Part 2 of this three-part series.
- [Producing documentation and reusing information in XML, Part 3: Creating multi-target XML documents](#) (William von Hagen, developerWorks, July 2009): Customize documentation for specific audiences and output formats using XML attributes and a pre-processing step in Part 3 of this three-part series. Using DocBook as an example, simplify and automate production of finished documentation with shell scripts and Makefiles.
- [DocBook: The Definitive Guide](#): Find complete information about DocBook markup in SGML and XML. The complete text of this book is available online, but it's well worth buying a copy if you're going to do serious publishing work with DocBook.
- [DocBook XSL: The Complete Guide](#): Read definitive information about DocBook stylesheets and publishing. The complete text of this book is available online, but it's well worth buying a copy if you're going to do serious publishing work with DocBook.
- [What is XSL-FO?](#) (G. Ken Holman, O'Reilly xml.com, March 2002): Peruse a great introduction to the XSL Formatting Object specification.
- [OASIS DocBook Technical Committee](#): Get information about existing and future versions of the DocBook standard. The [OASIS](#) Web site provides a great deal of information about XML and other open standards.
- [DocBook-related mailing lists at OASIS](#): Join general discussions of DocBook markup and applications that process and format DocBook documents.
- [XML Schema 1.1, Part 1: An introduction to XML Schema 1.1](#) (Neil Delima, Sandy Gao, Michael Glavassevich, Khaled Noaman; developerWorks; December 2008): Get detailed information about defining and organizing XML Schema.
- [DTD, XML Schema, and DSD](#): Find a good comparison of DTDs and schemas and a pointer to [An Introduction to XML and Web Technologies](#), a good book that lives up to its title.
- [IBM XML certification](#): Find out how you can become an IBM-Certified Developer in XML and related technologies.
- [XML technical library](#): See the developerWorks XML Zone for a wide range of

technical articles and tips, tutorials, standards, and IBM Redbooks.

- [developerWorks technical events and webcasts](#): Stay current with technology in these sessions.
- [developerWorks podcasts](#): Listen to interesting interviews and discussions for software developers.

Get products and technologies

- [Apache FOP](#): Download the latest version for your platform. Click `Download` on the left side.
- [xsltproc](#): Download the latest version for UNIX and Linux systems.
- [DocBook DTD](#): Download the latest version and get started.
- [DocBook XSL Stylesheets](#): Download the latest version. (Go to **DocBook Project site > file releases** and select **docbook-xsl**.)
- [xmllto](#): Download the latest version if it is not already available on your system. Depending on the version of UNIX or Linux that you are running, you might also need to download and install [Bash, the Bourne-Again shell](#), [GNU find and friends](#), and [mktemp](#).
- [IBM Java SDK](#): Download this developers kit for your platform. You might need to download and install the IBM Java SDK to install and use the Apache FOP formatter on certain UNIX and Linux distributions.
- [IBM product evaluation versions](#): Download or [explore the online trials in the IBM SOA Sandbox](#) and get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.

Discuss

- [Participate in the discussion forum for this content](#).
- [XML zone discussion forums](#): Participate in any of several XML-related discussions.
- [developerWorks blogs](#): Check out these blogs and get involved in the [developerWorks community](#).

About the author

William von Hagen

William von Hagen has been a writer and UNIX systems administrator for more than 20 years and a Linux advocate since 1993. Bill is the author or co-author of books on

subjects such as Ubuntu Linux, Xen Virtualization, the GNU Compiler Collection (GCC), SUSE Linux, Mac OS X, Linux file systems, and SGML. He has also written numerous articles for Linux and Mac OS X publications and Web sites.

Trademarks

IBM, the IBM logo, ibm.com, DB2, developerWorks, Lotus, Rational, Tivoli, and WebSphere are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. See the current list of [IBM trademarks](#).

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both. UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.