

Producing documentation and reusing information in XML, Part 2: Reuse information in XML documentation

Reduce work, increase usability, and enforce consistency with document reuse

Skill Level: Intermediate

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

26 May 2009

Updated 07 Jul 2009

Discover simple solutions to reuse information in XML documentation, such as how to use XInclude to include other documents at a given point in a document and how to use XPointer to include small document fragments from other documents or a similar pool of information in XML format. Also, get tips for structuring XML documentation to simplify information reuse, and learn how to maintain stand-alone documents that you can incorporate into larger documents.

Reuse information

Other articles in this series

- [Part 1: Document publishing using XML: Create, format, and publish documents using XML standards and open source tools](#)
- [Part 3: Creating multi-target XML documents: Single-source documents for multiple audiences and output formats](#)

As I discussed in [Part 1](#) in this series, the conceptual predecessor of XML was the Standard Generalized Markup Language (SGML). The primary goal of SGML was to separate the *content* of a document—the actual words and data it contains—from its *presentation*—the way in which the content is displayed in a specific output format or on a specific output device. Using special notation known as *markup* to identify the structure and type of information contained in a document enables you to focus on the content of the document rather than how the information is eventually used or presented. Delivering structured information in a generic format that is independent of how that information is being used is even more important in XML, which is not limited to the realm of documentation.

Frequently used acronyms

- **DTD**: Document Type Definition
- **W3C**: World Wide Web Consortium
- **XML**: Extensible Markup Language

In the documentation space, one important side effect of maintaining information in a structured, text-based format is that XML information does not rely on any particular software package. You can work with XML documentation using applications that range from text editors to sophisticated graphical tools. However, an even more important side effect of structural and logical markup is that it enables you to easily reuse portions of the information in any context where that information is structurally valid.

DITA

Other XML documentation solutions, such as the Darwin Information Typing Architecture (DITA), offer similar flexibility and are designed to simplify information reuse in XML documentation environments. DITA requires substantial fragmentation of the information you are writing and managing, which might not be feasible or manageable (depending on staff and software requirements), but it is definitely worth considering as an alternative to the more traditional, DocBook-focused solutions discussed in this article.

By definition, structured documents logically consist of multiple sub-components: Books consist of multiple chapters, and chapters consist of multiple sections—all of which are presented in a specific order. Like programming concepts such as modular programming, thinking of documents as multiple, discrete, and logical units of information is a good way to structure your writing tasks. This model also provides a good foundation for information reuse if you store each logical information unit separately (known as *chunking*) and are able to assemble documents by threading them together in a specific order.

To support creating large documents from discrete sub-documents, XML provides a fundamental mechanism, known as *XInclude*, for including external XML files into an XML document. This mechanism facilitates use of any discretely maintained chunk of information in multiple documents or in multiple locations within a single document. However, as the next section shows, XML provides several mechanisms for reusing information from other XML documents, each of which is appropriate in different circumstances.

XML content inclusion mechanisms

XML, like SGML before it, initially enabled you to include documents in other documents by declaring entities (known as *XML external entities* in XML). Entities provide a logical name for a specific file that you want to include. The basic syntax of an external entity is:

```
<!ENTITY name [PUBLIC "public-identifier"]  
            SYSTEM "system-identifier">
```

The `name` is simply the logical name that you associate with the `system-identifier`, which is a Uniform Resource Identifier (URI) for an XML file that you want to insert in your document when you invoke the entity. The `PUBLIC` keyword and `public-identifier` are optional and allow you to use a more flexible, location-independent URI to locate the entity if your system supports public identifiers. Here is a good example of an XML external entity definition:

```
<!ENTITY chapter1 SYSTEM "chapter1.xml">
```

If the file that you want to reference through an external entity is not an XML file, you must also specify the format of that file following the system identifier, as in this example for a PNG file:

```
<!ENTITY figure1 SYSTEM "figure1.png" PNG>
```

Notation identifiers are designed to make it easy for an XML processing or formatting system to locate a helper application that will handle the types of files associated with that notation.

External entities are handy but have some limitations. The most significant, in terms of reusing information, are:

- An XML file that is identified by an external entity can only contain a document fragment: It cannot contain a `DOCTYPE` declaration or a

stand-alone XML declaration. This means that an external entity cannot reference other external entities, which limits the granularity of external entities as an `include` mechanism.

- You cannot include only a portion of an XML file that an external entity identifies. This means that you must chunk every bit of information that you want to be able to reuse in multiple documents or document locations.

Limitations such as these eventually resulted in a proposal for more flexible and general XML inclusion mechanisms (*XML Inclusions*, more commonly referred to as *XInclude*) that was submitted to the W3C by IBM® and Microsoft® in 1999. This proposal was accepted and published as a recommendation by the W3C in 2004 and is now in its second revision, which was itself published as a recommendation in 2006.

The XInclude recommendation specifies two somewhat different ways to include external information:

- Identify the location (URI) of an external resource that you want to include in its entirety. This is typically referred to as a *simple XInclude statement*.
- Identify a unique chunk of XML information that is contained in an external resource and that can be pulled from that resource into another XML document. To do this, use an XInclude statement with specific attributes that specify the URI for the external resource that contains the chunk of information to include, an attribute that specifies that the target URI points to a parsable XML document, and an `xpointer` statement that identifies the XML element in that resource that you want to include. This method is often referred to simply as *XPointer*, which is actually the name of an extension of a specification on how to identify information in an XML file, known as the *XPath specification*.

Of these two approaches to include and reuse information, the first is generally used with larger chunks of information that you want to maintain as both a stand-alone document and an included one, while the second is extremely convenient to extract relatively small chunks of information from such documents.

Whether to include an entire external resource or pull a chunk of information from an external resource depends on how you organized the chunks of information that you manage and the type of information that you reuse.

After discussing how to obtain open source software that supports XInclude and XPath (and therefore XPointer), the remainder of this article explains how to use XInclude and XInclude/XPointer to include external information and discusses the types of scenarios in which each is most useful.

Get and install the required software

Although most if not all XML documentation software supports XPath and XInclude, this article focuses on open XML solutions that support these specifications. The best-known open source package for XML processing is `xsltproc`, which intrinsically supports XPath as well as XInclude processing when you specify the `--xinclude` command-line option. See [Resources](#) for information about where to obtain the `xsltproc` package.

If you use a Linux® system (or followed the instructions in [Part 1](#) of this series to set up an XML document processing environment), you probably have installed the `xsltproc` utility on your system already or it is at least available from the repositories or installation media for your distribution.

To verify that `xsltproc` is installed on your system, execute the following command:

```
xsltproc --version
```

If the `xsltproc` application is correctly installed on your system and is in your execution path, you should see the information about the version of the command that you are running.

Include external information using XInclude

The core of the XInclude recommendation is the `include` statement, which enables you to include other XML documents in a given document with the simplest form of an XInclude statement:

```
<xi:include href="../../../common/tutorial1.xml" />
```

The `href` attribute is an optional value that provides the URI of the resource to include.

As you can see from this example, the `include` statement is located in its own namespace (`xi`). To use the `include` statement in your documents without generating an error or warning in your documentation tools, declare the `xi` namespace. Add an attribute that defines that namespace to the root node of the type of document you are creating:

```
<?xml version='1.0'?>  
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.1.2//EN"
```

```
"http://www.oasis-open.org/docbook/xml/4.1.2/docbookx.dtd">  
<book xmlns:xi="http://www.w3.org/2003/XInclude"  
  id="product-usersguide">
```

After you add this declaration and one or more `xi:include` statements, you should be able to process your main document using your documentation tools and insert the specified files at the specified points.

Many documentation tools do not consider failure to include a document to be a fatal error, which can result in some very short documents. To provide a way to detect a failed `xi:include` statement, the XInclude specification provides the `fallback` statement. This statement is only valid within an `xi:include` statement and contains text that replaces the `xi:include` statement if the include fails. The following is an example of a simple `xi:include` statement with an `xi:fallback` statement:

```
<xi:include href="../../../common/tutorial1.xml">  
  <xi:fallback>FATAL ERROR - BROKEN INCLUDE!</xi:fallback>  
</xi:include>
```

Following this example embeds the string `FATAL ERROR - BROKEN INCLUDE!` in your XML output, which you can either spot directly or write a script that checks for such strings programmatically and halts document processing if one is found.

Use XPointer to include subsets of external information

As mentioned previously, XPointer is a subset of the XPath specification and is supported by the XInclude specification. Using an `xpointer` attribute in an `include` statement enables you to extract portions of an external XML resource and insert them at the current location in another document. Using XPointer within an XInclude statement is most common when you want to extract and reuse a nugget of information contained in another file. This provides an easy-to-use alternative to having to maintain all includable information as stand-alone external resources.

The basic syntax for an `include` statement that uses an `xpointer` attribute is:

```
<xi:include href="../../../common/glossary.xml" parse="xml"  
  xpointer="element(para)xpointer(//glossentry[@id='caching']/glossdef/*)"  
>
```

This example extracts the paragraph (`para`) element that the XPath statement `//glossentry[@id='caching']/glossdef/` points to, which is the contents of the `glossdef` element for a glossary entry with an ID of `caching`.

Glossaries are one of the most common examples of a case in which you want to maintain a number of different information units within a single file resource yet also be able to extract portions of that information for reuse elsewhere in your document and in other documents.

Organize XML documentation for reuse

The most significant point to consider when you include external XML resources in another document is that those resources must be syntactically valid at the point where you include them. This requires some planning when you create your XML data or refactor existing XML data.

One key to simplifying reuse through inclusion is to use the most generic markup possible in the document fragments you want to include. For example, DTDs and schemata such as DocBook offer section elements with explicit and derived hierarchy within the context of a document. Examples of explicit section elements include `<sect1>` and `<sect2>`. Using the equivalent generic `<section>`, which derives its hierarchy from the context in which it appears, provides a much more flexible model for information reuse, because it can be reused and is valid at multiple levels.

Reusing external resources that you want to both include and be able to generate as stand-alone documents can be somewhat more complex but is easy to automate. You can either maintain them as includable fragments that you place in either your book or a simple wrapper document that contains an XML article declaration and its basic framework, or you maintain them as stand-alone documents and convert them on the fly to an includable resource prior to actually including them.

For example, you might maintain a tutorial as a stand-alone DocBook document of type `article` so that you can print it stand-alone and have it automatically generate a table of contents. You might also want to be able to include that tutorial in a larger stand-alone document, such as a book, but an `article` is not valid in the book context. A DocBook article begins with a declaration of the form:

```
<article xmlns:xi="http://www.w3.org/2003/XInclude"
        id="tutorial1">
  <articleinfo>
    <title>
      Tutorial: Using Emacs
    </title>
  </articleinfo>
```

To convert this declaration into an includable document, you can write a small script or set of Makefile rules to transform it so that it looks like:

```
<section xmlns:xi="http://www.w3.org/2003/XInclude"
        id="tutorial1">
  <title>
    Tutorial: Using Emacs
  </title>
```

Here is a sample Makefile entry that does the same thing:

```
cat tutorial1.xml | grep -v articleinfo | \
  sed -e 's;<article;<section;' \
      -e 's;</article></section>';' > tutorial1-includable.xml
```

When you want to generate this document as a stand-alone article, you can process the file `tutorial1.xml`. When you want to include this document as a section in another document, you can invoke a Makefile rule like this example, then use an `<xi:include>` statement to include the file `tutorial1-includable.xml`.

Conclusion

In this article, you explored simple solutions to information reuse in XML documents, plus how to include external documents with XInclude and to include fragments from other documents with XPointer.

Today's corporate information needs and products require rich documentation sets that frequently use the same information in different contexts. Being able to use discrete chunks of information in multiple documents and document contexts minimizes the amount of writing you have to do and can eliminate the need to maintain the same information in multiple places. Dividing documentation content into multiple, uniquely identifiable units of information can increase the number of documentation files you have to manage and maintain. However, for most documentation groups, what you save in time and accuracy and gain in consistency can provide huge benefits, especially if you manage larger documentation sets.

Resources

Learn

- [Producing documentation and reusing information in XML, Part 1: Document publishing using XML](#) (William von Hagen, developerWorks, March 2009): Easily assemble an XML publishing system on any UNIX® or Linux system. Combine free, open source packages with XML for efficient re-use of information and produce documents in multiple formats from one set of source documents.
- [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.
- [XML Inclusions \(XInclude\) Version 1.0 \(Second Edition\)](#): See the W3C recommendation for definitive information about the XInclude specification.
- [XML Path Language \(XPath\)](#): See the W3C recommendation for definitive information about the XPath specification.
- [Entity Declarations, Attributes and Expansion"](#) (Norman Walsh, XML.com, August 1998): Read information about declaring and using XML entities.
- [DITA Infocenter](#): Find information about DITA, an attractive and popular XML documentation and publishing alternative to DocBook.
- [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

- [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**.)
- [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.