



# The XML Strategist

Casting a critical eye on the Next Big Thing in technical publishing.

SARAH S. O'KEEFE, Column Editor

## Publishing XML Content with XSL

BY SARAH S. O'KEEFE, Senior Member

As you implement an XML-based publishing work flow, you need to make some critical decisions about how to render your XML content. That is, how do you convert your application-neutral, vendor-neutral, unformatted XML content into paginated content (such as PDF) or HTML?

This article introduces one solution: the Extensible Stylesheet Language (XSL), which is a programming language for processing XML. XSL, like XML, is a specification of the World Wide Web Consortium ([w3.org/Style/XSL](http://www.w3.org/Style/XSL)).

### Rendering Content for Browser Display

XSL offers you two ways to make XML content viewable in a Web browser. The first option is to transform your XML files into HTML files. Once you have created the HTML files, you can copy them onto your Web server. This approach is shown in Figure 1.

The second option is to use an XSL stylesheet to provide rendering information to the browser. In this approach, you put the XML files and the XSL stylesheet on your Web server, and the reader's Web browser interprets the stylesheet and formats the XML content. This approach is shown in Figure 2.

Table 1 compares the advantages and disadvantages of both approaches.

### The XSL Language

Note: This is a highly simplified overview of XSL.

XSL is a declarative language. When you apply an XSL transformation to an XML file, the structure and sequence of the XML file determines the processing sequence of the XSL transformation.

The sequencing of information in the XSL file is almost completely irrelevant. Here is a simple XSL transformation file:

```
<?xml version = "1.0">
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/
XSL/Transform">
  <xsl:template match="/">
    Hello world
  </xsl:template>
```

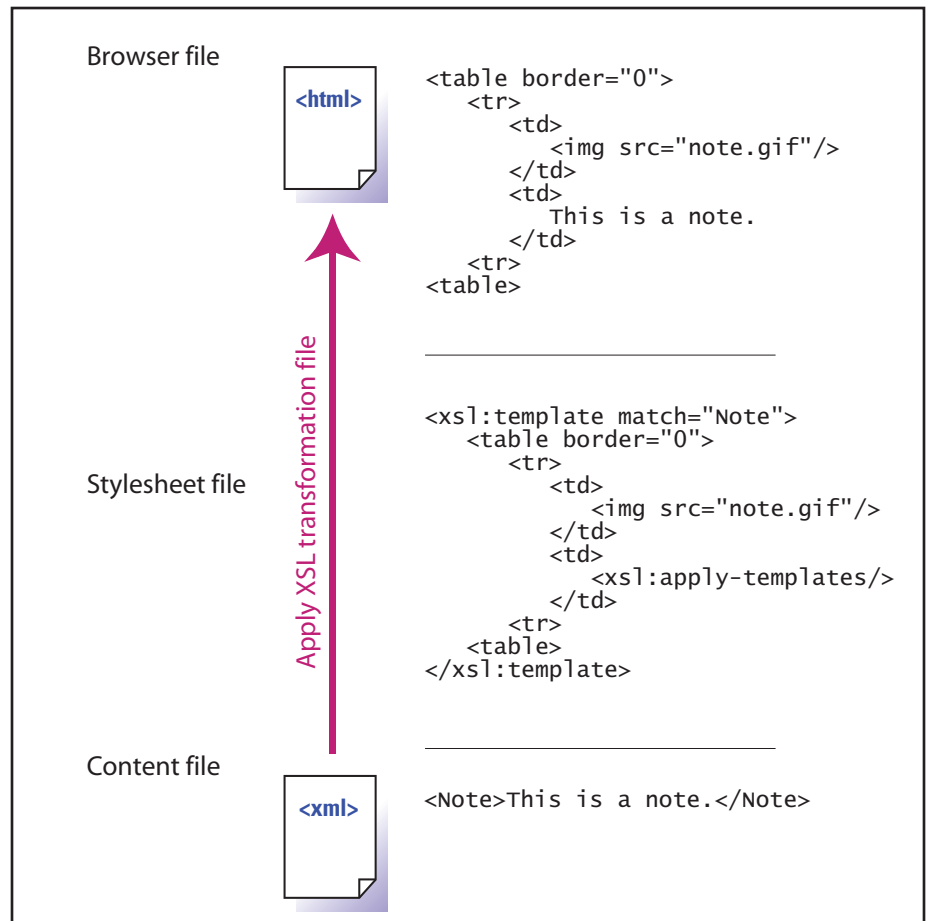
```
</xsl:stylesheet>
```

The `<xsl:template>` command matches the root of the document being processed. Since every well-formed XML document has a root, the result of the XSL transformation (for any XML file) would be:

```
Hello world
```

To run XSL transformations, you need

Figure 1. Transforming XML content into HTML.



an XSL processor. Xalan and Saxon are two popular processors. Using Saxon, you could run something like the following example on the command line:

```
saxon document.xml simple.xml
```

Let's assume that document.xml looks like this:

```
<?xml version = "1.0">
<topic>
<title>My important title</title>
```

```
<para>Content goes here</para>
</topic>
```

You want to output the <title> element as an HTML <h1> and the <para> element as an HTML <p> tag.

The XSL file you need looks like this:

```
<?xml version = "1.0">
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/
XSL/Transform">
<xsl:template match="/">
```

```
<xsl:apply-templates/>
</xsl:template>
<xsl:template match="topic">
<xsl:apply-templates/>
</xsl:template>
<xsl:template match="title">
<h1><xsl:apply-templates/></h1>
</xsl:template>
<xsl:template match="para">
<p><xsl:apply-templates/></p>
</xsl:template>
</xsl:stylesheet>
```

Figure 2. Using XSL stylesheets to display formatted XML in a browser.

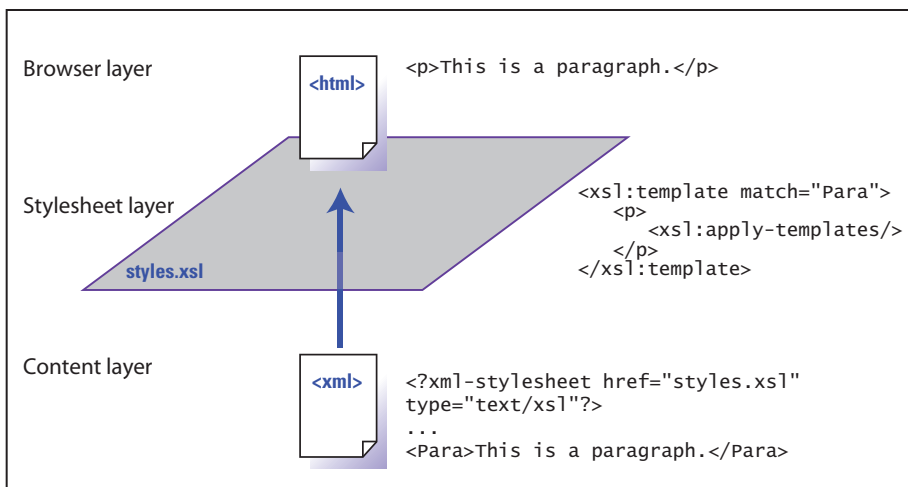


Table I. XSL Transformation vs. Embedded Stylesheet

Feature	Transformation	Embedded stylesheet
<b>Browser requirements</b>	You can produce HTML files that any browser can render.	All your readers must use browsers that support XSL ( <i>Internet Explorer 6</i> or higher and all versions of <i>Firefox</i> meet this requirement). If a significant number of your readers use older browsers, their browser may not be able to interpret your XSL stylesheet.
<b>File splitting</b>	You can split a large XML file into smaller HTML output files, which will improve browser performance for your end users.	The entire XML file is processed. If your XML files are large, this could cause delays in browser display.
<b>On-the-fly filtering</b>	Transformed HTML files are generally static, so there is no support for filtering content.	You can control what information is displayed based on a user's profile. For example, you could display a summarized version of a document for managers and a more detailed version for technical experts.

Notice the repeated use of <xsl:apply-templates/>. This command essentially says, "Keep processing and looking for more elements and content." When <xsl:apply-templates/> encounters text, it outputs the text into your result file. So the result of the transformation above would be:

```
<h1>My important title</h1>
<p>Content goes here</p>
```

This is lovely, but we need to add some HTML goodies around it so that the browser knows what to do. Here is the updated version:

```
<?xml version = "1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/
XSL/Transform">
<xsl:template match="para">
<p><xsl:apply-templates/></p>
</xsl:template>
<xsl:template match="/">
<html>
<body>
<xsl:apply-templates/>
</body>
</html>
</xsl:template>
<xsl:template match="title">
<h1><xsl:apply-templates/></h1>
</xsl:template>
</xsl:stylesheet>
```

Three things about the updated code are notable:

- I removed the topic template. When you do not provide a template for an element, XSL assumes, by default, that you want to run <xsl:apply-tem

(Continued on page 40)

## Teaching Technical Writing

(continued from page 39)

- Am I always respectful of my colleagues, students, and other technical communicators?
- Am I a lifelong learner? Do I make my commitment to learning obvious to my students? Do I show a balanced understanding of my material, or do I talk solely from my own experience?
- Do my handouts exemplify best practices in technical communication?

For me, the rewards of teaching professionalism come from watching a graduate become an outstanding program manager of the local chapter, deliver a presentation to the chapter, or become chapter president. As an instructor, my ultimate reward is setting my fledglings free and proudly watching them fly. 🦋

### SUGGESTED READING

Dannels, Deanna P. "Learning to be Professional: Technical Classroom Discourse, Practice and Professional Identity Construction." *Journal of Business and Technical Communication* 14 (1) (2000).

*Alexa Campbell is the founding president of the Manitoba Chapter STC, the founding manager of STC's Canadian Issues community, and a fellow of STC. After teaching high school for many years, she switched to technical communication, and then returned to the classroom to develop and teach in a two-year diploma program in technical communication at Red River College in Winnipeg, Manitoba. She completed her master's degree in adult education at the University of Manitoba in 2006.*

Discuss this article  
online at [stcforum.org/  
viewtopic.php?id=1171](http://stcforum.org/viewtopic.php?id=1171)



## The XML Strategist

(continued from page 35)

- plates/> for that element's child elements. Therefore, even though I don't match the topic explicitly, the topic's child elements (<title> and <para>) are still processed in the output.
- I changed the order of the templates. You can list templates in any order. The XSL stylesheet is processed based on the sequence of the XML file, not the sequence of the XSL file.
  - Notice that the <html> and <body> tags surround the <xsl:apply-templates/> command. So, processing goes through the tree structure. When it encounters the root, it outputs the <html> and <body> opening tags. Then it does all the interior processing, and then it returns to create the </body> and </html> tags.

The result of these changes looks like this:

```
<html>
  <body>
    <h1>My important title</h1>
    <p>Content goes here</p>
  </body>
</html>
```

To add the document title into the <head> section of the HTML, you can modify the root (/) template:

```
<xsl:template match="/">
  <html>
    <head>
      <title><xsl:value-of select="/topic/title"/></title>
    </head>
    <body>
      <xsl:apply-templates/>
    </body>
  </html>
</xsl:template>
```

The result is the following:

```
<html>
  <head>
    <title>My important title</title>
  </head>
  ...
</html>
```

The <xsl:value-of> command lets you retrieve content from specific parts of the document. In this case, the select attribute points to *topic/title*, which means the topic child of the root and then the title child of the topic.

For larger documents, you can also generate a table of contents. For example, the following template generates a two-level table of contents for topics and any embedded subtopics. (This template assumes that the HTML file name is embedded in each topic as an attribute.)

```
<xsl:template name="toc">
  <xsl:for-each select="/topic">
    <a href="{@filename}"><xsl:value-of select="title"/></a>
    <xsl:for-each select="topic">
      <a href="{@filename}"><xsl:value-of select="title"/></a>
    </xsl:for-each>
  </xsl:for-each>
</xsl:template>
```

You can do far more with XSL—it provides all the tools you need to go from XML to fully formatted HTML. You can embed formatting information directly in the HTML files you generate, or you can produce HTML with class attributes that refer to an external cascading stylesheet (CSS) file.

In addition to processing elements, you can access all the nodes in an XML document. Nodes include elements, attributes, processing instructions (<?processing instruction ?>), comments (<!-- my comment -->), text, and white space.

Although it's not always apparent, XSL really has three components:

- XSL transformations, as shown in the <xsl:template> sections in the preceding examples.
- XPath expressions, which let you identify specific parts of an XML document. The various match statements in my code examples use simple XPath expressions such as "para" or "title" or "/topic/title."
- XSL Formatting Objects (XSL-FO), which are not shown in my examples. XSL-FO provide a way to include detailed formatting specifications for

printed or PDF output. I highly recommend avoiding FO until you have mastered basic XSL and XPath.

A detailed discussion of XSL requires a book or two, such as Michael Kay's excellent *XSLT Programmer's Reference* (Wrox Press, 2003), an 800-page brick of a book.

### XSL Implementation Strategy

The availability of XSL significantly changes your single-sourcing options. Most single-sourcing work flows are built on proprietary authoring tools. For instance, you can create content in MadCap *Flare* and then export to various HTML formats and to print or PDF. From Adobe *FrameMaker*, you can use WebWorks *ePublisher Pro* or Omni Systems's *Mif2Go* to produce HTML output.

But with XML, you can separate the authoring process from the publishing process. Your authoring and publishing tools may not be the same.

### Print and PDF

Generating print and PDF from XML is difficult. The formatting requirements for print and PDF are much more complex than those for HTML. As a result, most solutions have commercial software components. You have two basic choices for print:

- An XSL-FO rendering engine. The three leading engines are *Antenna House*, *FOP*, and *RenderX*. *FOP* is an open-source application with some limitations; *Antenna House* and *RenderX* are more fully featured commercial software.
- A print publishing tool with support for XML. Options include *FrameMaker*, *InDesign*, and *QuarkXPress*. You can import and publish XML in any of these applications; you can also use *FrameMaker* to author valid, well-formed XML. *InDesign* and *QuarkXPress* are suitable for projects where highly designed, full-color printing is required; *FrameMaker* is better suited to long documents with a high level of consistency.

Should you choose XSL-FO or use

a print-publishing tool? XSL-FO will give you greater automation; a print-publishing tool will give you more attractive printed output. You will have to decide which of these considerations is more important—and your answer may vary for different documents.

### HTML and Other Markup Languages

If you are required to create HTML from your XML content, XSL is an excellent option. You can completely automate the HTML generation process and even set up work flows in which you generate several different forms of output at the same time. Once you try an XSL-based work flow, the traditional single-sourcing options (such as *FrameMaker* to WebWorks *Publisher*) seem dated.

XSL, like XML, is Unicode-based, so you can implement XSL-based publishing in a multilingual environment.

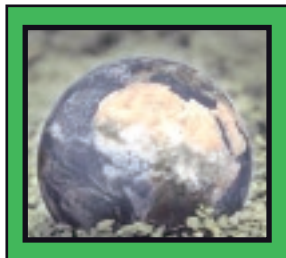

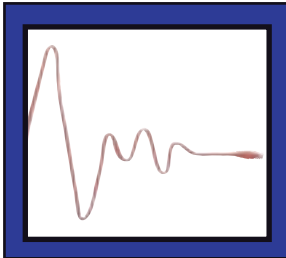
### Conclusion

For years I've been saying that good XSL-FO processing is coming soon. Unfortunately, it doesn't appear to have arrived yet. If you are willing to make some formatting compromises to achieve complete automation of your print production, you should take a look at XSL-FO. Otherwise, you'll need to look at the print-publishing tools, starting with *FrameMaker*.

For HTML production via XSL, no compromises are necessary. If you are moving to XML, XSL should be part of your implementation plan for HTML output. **❶**

*Sarah O'Keefe (xmlstrategist@scriptorium.com) is founder and president of Scriptorium Publishing Services, Inc. (www.scriptorium.com).*

*Discuss this article online at [stcforum.org/viewtopic.php?id=1169](http://stcforum.org/viewtopic.php?id=1169).*



## CONNECT.

CONNECT with other members, colleagues, and mentors. STC allows you to access other technical communication professionals through its 130 local chapters, its 21 special interest groups (SIGs), and the Technical Communication Summit.

## SHARE.

SHARE your accomplishments, ideas, and goals globally. STC offers the ability to collaborate with technical communication peers and friends through publications, online networks, and competitions.

## RENEW.

RENEW your career and expand your possibilities. STC invites you to renew your membership so that you can continue to enhance your profession, and increase your resources, while helping the Society grow and improve.

**To RENEW now, visit:  
[www.stc.org](http://www.stc.org)**