Stylish XML

Part two: using XSL to transform your XML documents

David Horton

Part one (`/articles/stylish_xml`) of this article looked at how Cascading Style Sheets (CSS) can be used to make XML documents look good in a web browser. In part two, I'll explore the more complex eXtensible Style sheet Language (XSL) and how it can be used to transform XML into HTML and PDF documents.

Unfortunately, one of the major problems with using XML/CSS is that it doesn't work for everyone

---

Limitations of XML/CSS

By the end of part one of this article I had a my tasty pico de gallo recipe marked up with XML tags and nicely styled using CSS. It looked great in my Firefox browser. Unfortunately, one of the major problems with using XML/CSS is that it doesn't work for everyone. Older, proprietary browsers and text-only browsers can only understand HTML and get terribly confused when trying to interpret XML/CSS. If I want to share my delicious recipes with someone using Netscape 4 or Lynx, I'm going to have to convert my XML into a format that their browser can handle. This means leaving behind CSS and constructing a new style sheet using the eXtensible Style-sheet Language (XSL).

As long as I'm in the mood for creating new things, I may as well create a new XML recipe to use for the examples in this article. You already have a recipe for a nice appetizer from part one, and now, for part two, you just need a drink to go with it. I can't think of anything better to have with pico de gallo and tortilla chips than a cold margarita so that's what I'll use in the examples. By the way, the margarita recipe and all of the style sheets in this article are available as a compressed download.

Transforming with XSL

An XSL Transformation, or XSLT, is the process of transforming an XML document into another type of XML document. Now why on earth would someone want to transform one XML document into another XML document? I'll give you a hint, Grasshopper: HTML is another type of XML document. I can build a custom XSL style sheet, apply it with a tool called an XSLT processor and presto, my XML is magically transformed into HTML.

In the examples, I will be using the XSLT processor called `xsltproc` to process the XSL style sheets. The `xsltproc` tool should be available as a package for most GNU/Linux distributions and Apple's OS X, if you want to follow along. The basic syntax for the command is `xsltproc -o [output-file] [style-sheet] [input-file]`. Or, in the case of this article's examples, `xsltproc -o output.html recipe-style.xsl margarita.xml`. If you don't have access to `xsltproc`, or if you're just feeling a little apathetic about typing all these commands, the output files from each of the examples are included alongside the other files in the compressed download.

An XSL Transformation is the process of transforming an XML document into another type of XML document

---

Simple beginnings

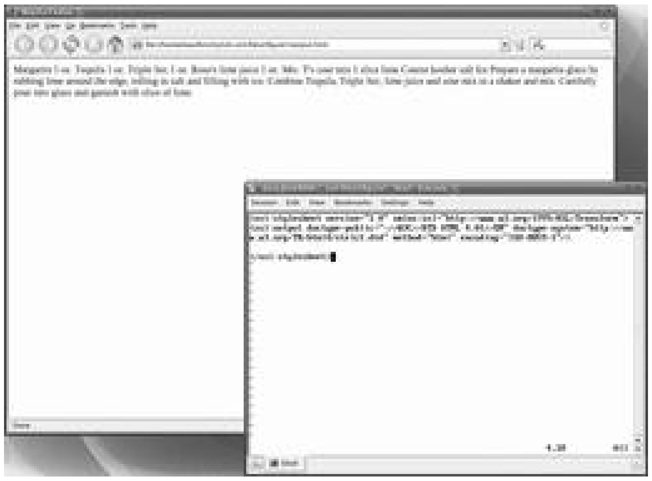
The most basic XSL style sheet is one that does nothing at all. Of course, there are a few headers that are required, but for the most part the style sheet is devoid of any processing instructions. When using a blank style sheet like this, it appears that `xsltproc` simply strips the tags from the XML recipe and dumps it as plain text to the output file. Close, but not quite. Take a look at the first line of the output and you will see a document-type declaration for HTML. This was added during processing, because the second line of the `recipe-style.xsl` specifies HTML as the output document. So I'm on the right track, but `output.html` displayed in a browser looks really bad. That's because the only output I have produced so far is a plain-text file masquerading itself as HTML.

Introducing the XSL template

A real HTML document should have tags and currently `output.html` has none. To make a valid HTML file I'll need to fix up the XSL style sheet to at least produce <html>, <head> and <body> tags in the appropriate places. This can be done by adding a single XSLT processing rule called a template. You can think of an XSLT template as being like a word processor's find-and-replace feature. You use the "match" attribute to tell the template what XML element it should find. Everything between the template's starting and ending tags is the replacement text. To convert my margarita recipe to HTML I need my style sheet to match the recipe element in the XML source file and replace it with appropriate HTML elements in the output document.

It wasn't hard to add the template rule to my `recipe-style.xsl` style sheet and get the HTML tags I wanted to see. Unfortunately, although I am one step closer to a valid HMTL document, it seems that all of the recipe's content has now disappeared. This is because I have started using processing rules in my style sheet, but I have not followed through by specifying where to place the recipe's content. Adding a simple <xsl:apply-templates /> element between the HTML <body> tags will fix things up and get my recipe content back.

Output produced by an "empty" style sheet

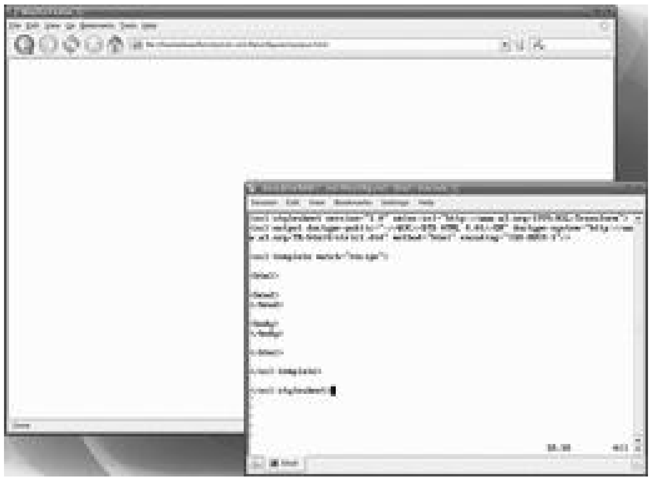


```
Margarita 1 oz. Tequila 1 oz. Triple Sec 1 oz. Rose's lime juice 1 oz. Mix [illegible]
[illegible]
```

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output doctype-public="-//W3C//DTD HTML 4.01//EN" doctype-system="http://www
w.w3.org/TR/html4/strict.dtd" method="html" encoding="ISO-8859-1"/>

</xsl:stylesheet>
```

HTML tags are here, but the recipe's gone!



```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output doctype-public="-//W3C//DTD HTML 4.01//EN" doctype-system="http://www
w.w3.org/TR/html4/strict.dtd" method="html" encoding="ISO-8859-1"/>

<xsl:template match="recipe">

<html>

<head>
</head>

<body>
</body>

</html>

</xsl:template>

</xsl:stylesheet>
```

Templates galore

Now that I have created some basic HTML output, my recipe just needs a little aesthetic improvement. OK, so it needs a lot of improvement, especially in terms of appropriate line breaks and whitespace, but this is something I can easily do by employing more XSL templates. So far I've used a single XSL template in my `recipe-style.xsl` file. There's no reason I should stop at just one. In fact, I can create a template for each XML element in my `margarita.xml` source file to give the HTML output a nice look.

You can think of an XSLT template as being like a word processor's find-and-replace feature

Since I am to catering to people with older, proprietary browsers and text-only browsers, I'll use standard HTML tags to achieve the basic style I want. A big improvement to my HTML output can be made by just by adding <p> and <br> tags to get line-breaks in the correct places. I can also change the font size for the title and section heading by using <h1> and <h2> tags, respectively. Four more simple templates added to `recipe-style.xsl` give the HTML output a more appealing look.

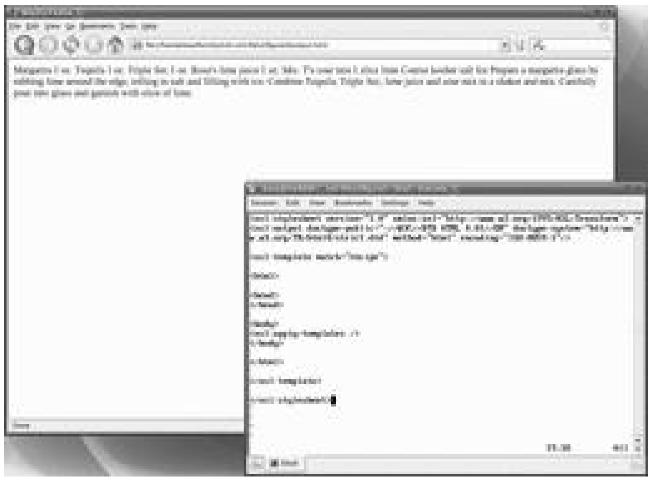
The value of a title

My recipe already has the title displayed as a nice big level-one heading, but it really should have a title between the HTML <head> tags as well. It's not a big deal in terms of the look of the document, but it is required for my output to be considered valid HTML.

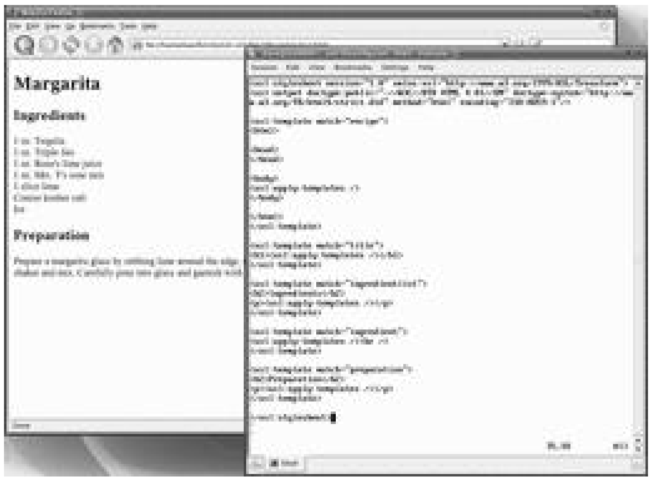
I already have a template for the XML title element that displays it between <h1> tags in the HTML output and now I need to use the title again in another place, this time between <title> tags. While it is possible to write two templates for the title element and apply them differently in different situations, it is far easier to use an XSL <value-of /> element.

Remember when I said that templates were like using find-and-replace in a word processor? Well, <value-of /> is like using just the find without the replace, and it uses a select attribute to specify what to find. For example, if I want to know what text is between the <title> tags in my XML document I would simply use <value-of select="title" /> in my XSL style sheet to find it. Strategically placing this value-of element in my XSL style sheet will result in the recipe title appearing in the HTML header of my output as well as in the body. The result doesn't look that much different in a browser, but by adding this title the HTML will now pass W3C's markup validation (`http://validator.w3.org`).

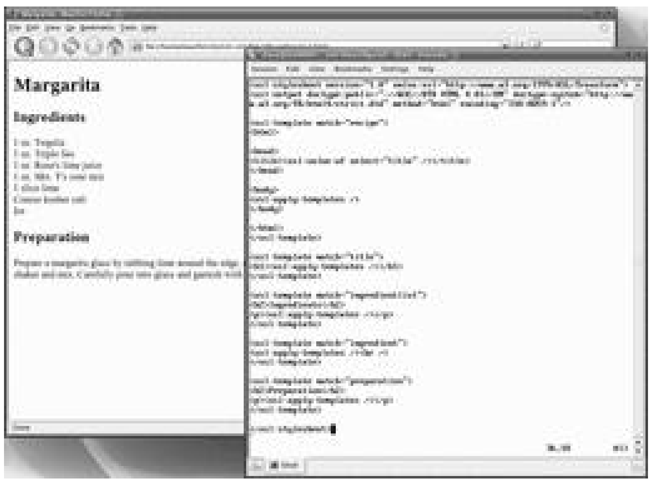
¡apply-templates /> brings the recipe's content back



More templates for a better look



HTML that validates



Margarita

Ingredients

[illegible]

Preparation

[illegible]

Transforming to PDF and more

With just a few simple templates, XSL does a great job of transforming my XML-based margarita recipe into HTML, but that's not all it can do. With a little more work I can also produce documents for Adobe Acrobat or OpenOffice all from the same XML source file. I'll need to construct a new XSL style sheet and install a new tool called a Formatting Objects processor, but otherwise the procedure is very similar to what we've covered so far.

The reason for the new style sheet and the new tool is that transforming to PDFs is a two-step process. First, I will use XSL templates to transform my XML-based margarita recipe into another type of XML document. Sounds familiar right? Only this time I'm not transforming into HTML, but rather into a markup language called Formatting Objects (FO). The second step is to take the FO markup and run it through the FO processor to get a PDF document.

Formatting Objects markup may look scary at first, but it's not bad after you play around with it a bit. To help you get comfortable I have included a sample XSL style sheet and Formatting Objects output as part of the files in the compressed download.

Further exploration

This article barely scratches the surface of the everything that can be done with XSL and FO. For those of you who want to go further, I have listed some helpful references below.

The W3 Schools website has some good tutorials covering XSL transforms (`http://www.w3schools.com/xsl/`) and XSL formatting objects (`http://www.w3schools.com/xslfo/`).

There are some XSL style sheets that you can download and examine both from my web site (`http://www.happy-monkey.net/recipebook/`) and from the DocBook XSL repository (`http://docbook.sourceforge.net/`).

You can also learn a lot from using the various tools. Available XSLT processors include xsltproc (`http://xmlsoft.org/XSLT/xsltproc2.html`) and Saxon (`http://saxon.sourceforge.net/`). A couple of my favorite Formatting Objects processors are Apache FOP (`http://xml.apache.org/fop/`) and XMLMind's FO Converter (`http://www.xmlmind.com/foconverter/`). All but `xsltproc` are Java-based and may be used on a variety of platforms.

Copyright information

About the author

David Horton got started with GNU/Linux in 1996 when he needed a way to share a single dial-up internet connection with his college room-mates. He found the solution he needed with an early version of Slackware and a copy of the PPP-HOWTO from The Linux Documentation Project. Ten years later he is older and wiser and still hooked on GNU/Linux. Many of Dave's interests and hobbies can be explored on his website (`http://www.happy-monkey.net`).