Article Publication Date: March 4, 2003
Article URL: http://www.zend.com/zend/tut/tutorial-wong3.php
Author URL: n/a

Back to Article

Using PHP and XSL to Transform XML into Web Content

March 4, 2003

By Benson Wong

Intended Audience

This tutorial is for developers who have been looking for ways to use XML to manage web content, and XSL to style and format that content. You should be familiar with XML, and having some experience with XSL would be helpful.

Overview

Presenting content is one of the more time consuming and challenging tasks when building a web site. This tutorial looks at taking an XML document and dynamically transforming it into HTML using PHP and XSL. As a practical example, this tutorial is available as XML as well as the XSL stylesheet which transforms it into Zend.com's common tutorial layout.

The goal of this tutorial is to demonstrate how to use PHP and XSL to transform an XML document into HTML.

Learning Objectives

In this tutorial you will learn:

- An introduction to XML, XSL's technologies and the Simplified DocBook DTD.
- How to use PHP and XSL to transform a XML document into HTML

Background Information

When you're working with a lot of content coming from many different authors (like the Zend.com site itself) you need an efficient way of managing and organizing the content. On the Web, documents often need to be presented in a variety of different formats. XML has grown immensely in popularity for web publishing, because it allows data to be stored in a structured format that is easily transformed into other formats.

XSL was created as a tool for transforming XML documents. The core of XSL is made up of three technologies. XSL:T, XPath and XSL-FO. In this tutorial we will be looking only at using XSL:T (templates) and XPath for generating HTML.

Definitions

- **XML** is a meta language for defining markup languages. For more information on XML see "What is XML?" at XML.com
- **XSL** (Extensible Stylesheet Language) is used for expressing Stylesheets for XML documents. It is made up of three parts, XSLT (transformations), Xpath, and XML-FO. An XSL document is a well formed XML document, that instructs an XSL processor how to transform an XML document. See, What is XSL? for more information.
- **XPath** is a language for addressing and accessing parts of an XML document. It was designed to be used by XSL:T. The XPath specification can be found at: http://www.w3.org/TR/xpath, and tutorials can be found at: http://www.w3schools.com/xpath/

- **XSL:T -** XSL:T is one of the main technologies of XSL. It stands for XSL Transformations and is a language for transforming XML documents into other formats. The XSL:T specification can be found at http://www.w3.org/TR/xslt and tutorials can be found at http://www.w3schools.com/xsl/.

Simplified DocBook

Simplified DocBook is a small subset of the original DocBook XML DTD. While the full DocBook DTD is perfectly adequate to structure content for the web, it can be over kill for short web articles. Simplified DocBook addresses a more limited set of needs and provides a more specific set of elements.

Although Simplified DocBook is much smaller than the full DocBook DTD, it is still quite hefty in itself. Simplified DocBook is composed of 106 elements, 525 entities, and 26 notations.

For more information on Simplified DocBook see: http://www.oasis-open.org/docbook/xml/simple/.

Prerequisites

- PHP compiled with XML and XSL support
- A Simplified DocBook source file
- An XSL Document

How it works

Creating the XML File

Writing articles in a mark up language is time consuming and can be quite a painful process. Having the right tools can take some of the effort out of creating structured documents. For example: I wrote the content of this tutorial in Open Office's Writer, and then structured it into XML with XMLEditor by XMLmind.com, when it was ready to be published.

The advantage of using an XML tool to assist in the structuring is that the final document is less prone to human error. Most XML tools provide faculties for automatically formatting and spacing the XML tags, as well as checking the validity of the document.

XML Editor can be found at: http://www.xmlmind.com/

The Simplified DocBook Structure

Although the Simplifed DocBook is only a small part of DocBook, it can still be a little overwhelming to those unfamiliar with it. This section provides an introduction to the structure of Simplifed DocBook.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE article PUBLIC "-//OASIS//DTD Simplified DocBook XML
V4.1.2.5//EN"
"http://www.oasis-open.org/docbook/xml/simple/4.1.2.5/sdocbook.dtd">
<article>
  <title>A Short Example</title>

  <section>
    <title>Section #1</title>

    <para>A short example of a Simplified DocBook file.</para>
  </section>
</article>
```

In Simplified DocBook there are really only two main structuring elements, they are `<article>` and `<section>`. Other elements, such as `<para>`, `<example>`, `<itemizedlist>`, etc. describe different things within the document.

In Simplified DocBook the root element is always `<article>`. Articles can be separated into sections, and each sections can contain an unlimited number of sub-sections. The most common types of element are titles, paragraphs, links, and lists. For a full listing of elements see Simplified DocBook: The Definitive Guide.

Creating the XSL Template

The purpose of the XSL template we will be creating in this tutorial is to transform the Simplified DocBook structured data into Zend.com's standard tutorial format. The standard layout for their tutorials are:

- Title of tutorial
- Date of article
- Name of author
- Table of Contents
- Intended Audience
- OverviewLearning
- Objectives
- Definitions
- Background
- Information
- Prerequisites
- How it works
- The Example Script

Creating an XSL Style sheet to transform XML into HTML is a little more difficult than creating the XML data file, because it requires knowledge of both the template language and XPath. Anything more than introductory is outside the scope of this tutorial, but I'll try to give you a good idea of the role each of these technologies in the transformation.

Here is an example of an XSL Template to transform the minimal example above:

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

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

    <xsl:template match="article/title">
      <h1><xsl:value-of select="."/></h1>
    </xsl:template>

    <xsl:template match="section">
        <xsl:apply-templates/>
    </xsl:template>

        <!-- Formatting for JUST section titles -->
        <xsl:template match="section/title">
          <h2><xsl:value-of select="."/></h2>
        </xsl:template>

    <xsl:template match="para">
      <P><xsl:apply-templates/></P>
    </xsl:template>
</xsl:stylesheet>
```

As you can see, a XSL template is simply a well formed XML document. The root element is `<xsl:stylesheet>`, and it is usually made up of `<xsl:output>`, `<xsl:template>, <xsl:apply-templates>` and `<xsl:value-of>`. There are a few more useful elements but those four are essential to providing the core functionality.

Fundamental XSL Elements

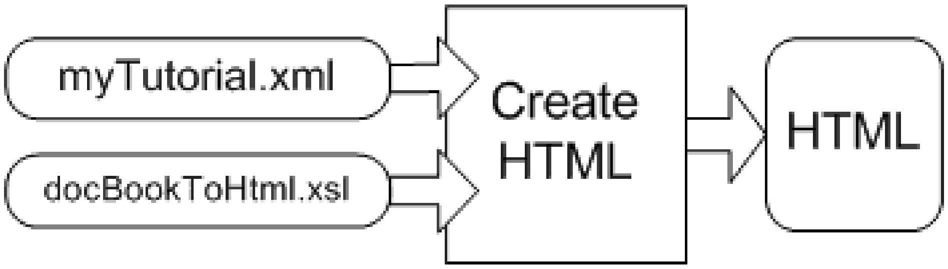
<xsl:output>	This element defines the format of the output document. The allowed methods are xml, html, text and name.
<xsl:template>	An XSL Stylesheet consists of a set of templates. These templates contain rules that are applied when a specific matching node is found. Template element structure is quite simple. At the most basic level they simply replace the element they match with whatever is in the template. From the above example when `<xsl:template match=”article/title”>` is matched, the XML `<title>A Short Example</title>` turns into HTML as`, <h1>A Short Example</h1>`. To correctly match a template to a single node or a set of nodes, XPath is used. In the above example, `<xsl:template match=”para”>` matches any `<para>` element within the document. If the template element read, `<xsl:template match=”article/para”>` then only `<para>` elements directly beneath `<article>` will be matched.
<xsl:apply-templates>	This element instructs the XSL processor to apply matching templates to the current element, or the current element's child nodes.
<xsl:value-of>	This element extracts the value of a selected node.

There are a few other common and useful elements, such as `<xsl:if>` and `<xsl:choose>` which allows testing of conditions before applying rules, `<xsl:for-each>` which allows looping through elements, and `<xsl:sort>` which allows sorting of output.

Putting it all together

Performing XSL transformations in PHP is very simple. The hardest part of using XSL in PHP is creating the actual transformation file.

With XML and XSL documents, transformation using PHP consists of two steps: The first step is to create the XSL:T processor with `xlst_create(); the second is to` use `xlst_process()` to transform the XML into HTML.



The `xslt_process()` function handles all the transformation work. There are three different ways of using `xslt_process()`. The examples below are from the PHP Manual.

Method 1: Automatically save the results to a file

```
<?php

// Allocate a new XSLT processor
$xh = xslt_create();

// Process the document
if (xslt_process($xh, 'sample.xml', 'sample.xsl', 'result.xml')) {
    print "SUCCESS, sample.xml was transformed by sample.xsl into
result.xml";
    print ", result.xml has the following contents\n<br>\n";
    print "<pre>\n";
    readfile('result.xml');
    print "</pre>\n";
}
else {
    print "Sorry, sample.xml could not be transformed by sample.xsl
into";
    print "  result.xml the reason is that " . xslt_error($xh) . "
and the ";
    print "error code is " . xslt_errno($xh);
}

xslt_free($xh);

?>
```

Method 2: Returning the results as a string

(In most cases this method is more useful.)

```
<?php
```

```
// Allocate a new XSLT processor
$xh = xslt_create();

// Process the document, returning the result into the $result
variable
$result = xslt_process($xh, 'sample.xml', 'sample.xsl');
if ($result) {
    print "SUCCESS, sample.xml was transformed by sample.xsl into the
\$result";
    print " variable, the \$result variable has the following contents
\n<br>\n";
    print "<pre>\n";
    print $result;
    print "</pre>\n";
}
else {
    print "Sorry, sample.xml could not be transformed by sample.xsl
into";
    print "  the \$result variable the reason is that " . xslt_error
($xh) .
    print " and the error code is " . xslt_errno($xh);
}

xslt_free($xh);

?>
```

Method 3: Providing and returning strings

The third way of using `xslt_process()` is to provide the XML and XSL documents as string values, and have the function return a string value. This method is the most flexible as you can do some pre-processing and post-processing to the XML or XSL documents before passing them to the processor.

```
<?php
// $xml and $xsl contain the XML and XSL data

$arguments = array(
     '/_xml' => $xml,
     '/_xsl' => $xsl
);

// Allocate a new XSLT processor
$xh = xslt_create();
```

```
// Process the document
$result = xslt_process($xh, 'arg:/_xml', 'arg:/
_xsl', NULL, $arguments);
if ($result) {
    print "SUCCESS, sample.xml was transformed by sample.xsl into the
\$result";
    print " variable, the \$result variable has the following contents
\n<br>\n";
    print "<pre>\n";
    print $result;
    print "</pre>\n";
}
else {
    print "Sorry, sample.xml could not be transformed by sample.xsl
into";
    print "  the \$result variable the reason is that " . xslt_error
($xh) .
    print " and the error code is " . xslt_errno($xh);
}
xslt_free($xh);
?>
```

The Script

The following script demonstrates how post-processing can be applied after the XML document has been transformed. After the document has been transformed, PHP looks for special delimiters in the HTML, which designate different languages (ie. php), and run them through syntax highlighting.

```
<?php

function unhtmlentities ($string)
{ // FROM PHP Manual
    $trans_tbl = get_html_translation_table (HTML_ENTITIES);
    $trans_tbl = array_flip ($trans_tbl);
    return strtr($string, $trans_tbl);
}


// $xml and $xsl contain the XML and XSL data...
$xml = implode('',file('document.xml'));
$xsl = implode('',file('sdocbook.xsl'));
```

```
$xh = xslt_create();
$output = xslt_process($xh, 'arg:xml', 'arg:xsl', NULL,
             array(
              'xml' => $xml,
              'xsl' => $xsl));

/* Find all the little parts that need to be highlighted.
   .. the XSL transformation process puts in tags like
   <parse_php>, <parse_xsl>, <parse_xml>... etc.

*/
$types = array('php');  // add more to this array as we find better
highlighters

foreach ($types as $type) {
    $oKey = "<parse_$type>";
    $cKey = "</parse_$type>";

    $outputArray = explode($cKey,$output);
    unset($outputArray[count($outputArray)-1]); // drop the data at
the end.
    foreach ($outputArray as $data) {
        $data = substr($data,strpos($data,$oKey)+strlen($oKey));
        switch ($type) { // do the highlighting...
            case 'php':
                $dataReplace = unhtmlentities($data);
                $dataReplace = highlight_string($dataReplace,true);
                break;
        }

        // replace the data in output..
        $output = str_replace("$oKey$data$cKey",$dataReplace,
$output);
    }
}

echo $output; // print output
?>
```

About the Author

Benson Wong is a programmer and a system administrator. As the head of Tummy Tech's technology department, Benson often has to be a Swiss army knife of technical knowledge. While most of his time is spent working with his programming team, Benson occasionally works with his network and system administration team to develop new procedures for streamlining the technical requirements of their clients. When not at the office, Benson is building the electronic music community beatmatch.com and working on his martial arts training.

Benson can be contacted at phpauthor@tummytech.com.