Processing RSS Files with XSLT

Dr. A J Trickett, atrickett@cpan.org

Abstract

Rich Site Summary format XML files are traditionally converted to HTML for display in a browser by a brute force approach. The XML transformation language XSLT can be used instead, while Perl deals with the logistics.

1 Introduction

A number of years ago I lived in Southern California. At the time the best place to get news was via National Public Radio, Public Broadcasting Service or the BBC World Service. Unfortunately, none of them had particularly good reception where I lived, and so I was forced to turn to the web for news of home. In those boom-time days, it seemed every web site was a portal, whether it made business sense or not. This helped me build a custom news page with British and European news, along with world business and technology news.

Returning to Europe I abandoned my academic training and rushed headlong into enterprise web content management working for one of the pioneering XML companies. We were so pioneering that we never made a profit and as the boom turned to bust the company imploded to nothing. My thoughts again turned to syndicated news and portals—How was it done? Was it easy? Could we use it to build a federated web site that was easy for a small company to build and maintain?

2 RSS Basics

Rich Site Summary (RSS) files allow people to syndicate a web site. RSS is an initialization of one of several possible phrases: Rich Site Summaries, Really Simple Syndication, or Resource Description Framework Site Summaries. As RSS evolved, the meaning of RSS has shifted to match it's evolving abilities.

An RSS file uses the Extensible Mark-up Language (XML) to give a summary of the content on that site. The XML specification is a descendent mark-up language of Standard General Mark-up Language (SGML) developed in the 1970s, but unlike SGML, which is complex to use, the designers wanted XML to be simple like HTML, which had proved popular as the basis of web sites. Typically, a site's content management system constructs RSS as stories and articles show up on the web site. Most sites place their RSS files on their web sites, and so the files are easy to download. Mirroring tools that only download the RSS files if they have changed are ideal for the task.

3 Extensible Style Language Transformations (XSLT)

The XML Style Sheet Language - Transformation (XSLT) is a World Wide Web Consortium (W3C) standard for converting XML documents to another format. A style sheet written in XML consists of a number of rules which the XSLT engine uses to convert the source XML to another format. A full introduction is beyond the scope of this paper, and I list many references at the end.

At the simplest level XSLT takes one XML document as a Document Object Model (DOM) tree, and coverts it to another format. The XSLT file is an list of transformation templates that apply to specific parts of the input DOM tree. Each individual template may operate in isolation of other rules, to give a result tree.

Code listing 1 shows a simple XML document with a `<statement>` tag and a `<footer>` tag. I want to apply a style sheet to it to produce the output in code listing 3.

In code listing 2, the first rule of my style sheet tells the engine to start at the root of the DOM tree `/`. It then outputs a `<div>` tag, and then the second rule tells the engine to look in the tree for a path than matches `root/statement` from the current context. If it finds a match the second template outputs a `<p>` tag, followed by the content of the current input tree node value `Hello World!`, then a `</p>`. Flow returns to the calling template, which outputs a `</div>`.

Code Listing 1: An example XML document

```
<?xml version="1.0"?>
<root>
<statement>Hello World!</statement>
<footer>Foo</footer>
</root>
```

Code Listing 2: An example style sheet

```
<xsl:template match="/">
    <div>
    <xsl:apply-templates select="root/statement"/>
    </div>
</xsl:template>

<xsl:template match="statement">
    <p>
    <xsl:value-of select="."/>
    </p>
</xsl:template>
```

Code Listing 3: Result of transformation

```
<div><p>Hello World!</p></div>
```

As with Perl, XSLT has more than one way to do it, which can intimidate new users. Like Perl, XSl-T is a very flexible language, so it is easy to write this style sheet in a totally different manner and get exactly the same result. I often find other people's XSL style-sheets very confusing, just as I did with other people's Perl scripts, but with time they do start to make sense.

3.1 Using Perl

Code listing 4 uses the LWP::Simple module to retrieve files, and the GNOME libxslt-based XML::LibXSLT to transform them. The first command line argument to the script specifies the file to fetch, and the second specifies the XSLT template to use. Once LWP::Simple fetches the XML file, XML::LibXML and XML::LibXSLT convert it to HTML via XSLT. Perl provides the framework for the download and conversion, and the XSL stylesheet provides the rules of the conversion—which separates code and appearance.

Code Listing 4: Fetch and transform XML file

```
#!/usr/bin/perl

use strict;
use LWP::Simple;
use XML::LibXML;
use XML::LibXSLT;

my $site = shift;
my $xslt = shift;

my $rss  = get($site);

my $xslt   = XML::LibXSLT->new;
my $parser = XML::LibXML->new;

my $source_xml  = $parser->parse_string($rss);
my $style_xsl   = $parser->parse_file($xsl);

my $stylesheet  = $xslt->parse_stylesheet($style_xsl);
my $transformed = $stylesheet->transform($source_xml);

print $stylesheet->output_string($transformed);
```

4 Problems with RSS

There are two RSS families, and they are different so that I cannot use the same XSL style sheet on all of them. In theory, I should be able to convert one RSS file into another one, but it is not that simple.

Netscape Communications developed the original RSS format, version 0.9, and UserLand later simplified it to create version 0.91. Independently, the RSS-DEV Working Group developed version 1.0, a new and incompatible format based on the W3C Resource Description Format (RDF) core. UserLand, unhappy with the RDF-based RSS, continued to extend and expand RSS up to its current version, 2.0.

The RDF-based RSS format uses XML namespaces, which has its advantages, but makes the document much more verbose and more diffficult to transform with XSLT. A number of XSLT proprietary extensions make this much easier, although they are not universally supported.

As many RSS files are automatically generated from badly written HTML by content managements systems, the resultant RSS content is often poorly formed or invalid. Some site editors correct their RSS feeds, but all too often there is nothing to be done but to accept that the incoming feed will be wrong.

The W3C requires an XML parser to abort processing if it that encounters a badly formed or invalid document. The major Perl XML parsers comply and will die in those cases. If a document format is invalid, the parser cannot convert it to a DOM tree, so Perl cannot transform the document. This is deliberate feature of XML to prevent potential ambiguity of on-the-fly second-guessing that HTML parsers perform. Most web browsers will read and display almost any form of HTML no matter how badly formatted it is.

4.1 The XML::RSS module

The XML::RSS module converts betweeen Perl structures and RSS formats. I can use it to programmatically convert one RSS version into another one; however, the module has a number or problems and limitations, plus one fatal flaw as of version 0.97—it does not output properly escaped XML, so any '&' is incorrectly outputted as '&', a special character in XML since it signals the start of an entity encoding. The module should encode any literal '&' as an '&'.

As a result of an email I sent to brian d foy regarding his recent article in this journal using the XML::RSS, he took it upon himself to fix the module, and another project on SourceForge was born. The project developers have not released a version that fixes this problem, although they have been working on it.

4.2 XML::RSS::Tools

The XML::RSS::Tools module attempts to circumvent source and XML::RSS escaping problems, while using XML::RSS to normalise the source. It incorporates HTTP tools and the GNOME-based XSLT engine to provide giving a complete a tool-kit. Code listing 5 uses the module to download the file, then transforms and outputs the result in one step. It has the same command line arguments as the earlier example—an RSS file location and an XSL template.

I create an XML::RSS::Tools object by initialising the module to its default configuration. Inside an eval, I use the object to load the source file and the xsl file, transform the source, and output the result as a string. I use an eval block in case an invalid RSS file causes the XML parser to die.

Code Listing 5: Using XML::RSS::Tools

```
#!/usr/bin/perl
use strict;
use XML::RSS::Tools;

my $rss = XML::RSS::Tools->new;
eval {
    print $rss->rss_file(shift)->xsl_file(shift)->transform->as_string;
    };

print $rss->as_string('error') if ($@);
```

This XSL style sheet in code listing 6 converts a single RSS feed into a XHTML fragment. It starts with the standard XML and XSLT header details. I told the process to turn off the XML declaration to make the fragment easier to directly incorporate in a XHTML document. I selected XML output and turned on indents to give a neater document.

The first template rule selects the XML root of the document, outputs a literal `<div>` tag, then applies the `rss/channel` rule, and outputs a `</div>` tag.

The `rss/channel` rule is where I process the details of the channel. I start by creating a number of variables, and populating them with the details to create an image link and the heading. I use an `xsl:if` to check if there is a image to link to, and if so populate an `<img>` tag with it. I create `<h3>` and `<a>` tags which link to the originating site. I output an `<hr/>` tag to separate the title, and then create an un-ordered list to put the individual story titles in. Inside the `<ul></ul>` tags I place an `xsl:apply-templates` command which inserts the contents of each item. One of the many nice things about the XSL language is that I do not need to know how many items there are in a given story—the simple rule will find them all.

The `item` rule creates a pair of variables for the link, outputs an `<li>` tag, constructs an `<a>` tag, and close with a literal `</li>` since this is XML.

Combining the style sheet with a RSS format will give the XHTML fragment in code listing 7.

Code Listing 6: Style sheet to transform RSS to XHTML

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
     exclude-result-prefixes="xsl">
<xsl:output method="xml" omit-xml-declaration="yes" indent="yes"/>

<xsl:template match="/">
     <div>
     <xsl:apply-templates select="rss/channel"/>
     </div>
</xsl:template>

<xsl:template match="rss/channel">
     <xsl:variable name="link" select="link"/>
     <xsl:variable name="description" select="description"/>
     <xsl:variable name="image" select="image/url"/>
     <xsl:if test="$image">
            <img src="{$image}" style="float: right; margin: 2px;" />
     </xsl:if>
     <h3>
        <a href="{$link}" title="{$description}"><xsl:value-of select="title" /></a>
     </h3>
     <hr/>
     <ul><xsl:apply-templates select="item"/></ul>
</xsl:template>

<xsl:template match="item">
     <xsl:variable name="item_link" select="link"/>
     <xsl:variable name="item_title" select="description"/>
     <li>
        <a href="{$item_link}" title="{$item_title}"><xsl:value-of select="title"/></a>
     </li>
</xsl:template>

</xsl:stylesheet>
```

Code Listing 7: XHTML result of RSS transformation

```
          <div>
            <img alt="Image"
src="http://use.perl.org/images/topics/useperl.gif" style="float: right;
margin: 2px;" />
            <h3>
              <a href="http://use.perl.org/" title="All the Perl that's
Practical to Extract and Report">use Perl</a>
            </h3>
            <hr />
            <ul>
              <li><a
href="http://use.perl.org/article.pl?sid=02/12/20/220252" title="">Parrot
v0.0.9 "Nazgul" released</a></li>
              <li><a
href="http://use.perl.org/article.pl?sid=02/12/18/2350216" title="">Happy
Birthday Perl</a></li>
              <li><a
href="http://use.perl.org/article.pl?sid=02/12/16/1648246" title="">POE
0.24 Released</a></li>
            </ul>
          </div>
```

5 Conclusion

Perl is a powerful language for collecting, downloading and manipulating data. The XML::RSS::Tools module works around some problems in XML::RSS and incorporates the XSLT processing. This way, the code is separate from the presentation details.

6 References

XML In A Nutshell, 2nd edition by Harold & Means, O'Reilly and Associates.

XSLT Quickly by Bob Ducharme, Manning Publications.

XSLT by Doug Tidwell, O'Reilly and Associates.

Beginning XSLT by Jeni Tennison, Wrox Press Ltd.

XSLT Programmer's Reference 2nd Edition by Michael Kay, Wrox Press Ltd.

XSLT Cookbook by Sal Mangano, O'Rielly and Associates.

Content Syndication with RSS by Ben Hammersley, O'Reilly and Associates.

"What is RSS?' by Mark Pilgrim, http://www.xml.com/pub/a/2002/12/18/dive-into-xml.html