Tip: Generating internal HTML links with XSLT

Automate the building of a Web page table of contents

Uche Ogbuji

February 01, 2001

Uche Ogbuji walks you through how to use XSLT to populate HTML or XHTML output with anchors and internal links. Internal links help to organize long HTML content; XSLT provides facilities for generating those internal links, but some of the methods are somewhat obscure. This tip, with reusable sample code, clearly spells out two approaches for the process.

View more content in this series

In long Web pages with logical sections, it is a good usability pattern to provide an index of links at the top that link to subsections of the document, making it easy for the person behind the browser access the desired content rapidly. These are represented easily enough in HTML. The spot is marked with a named anchor:

```
<a name="section3"></a>
```

And the link uses a fragment referencing the anchor name:

```
<a href="#section3"></a>
```

XSLT provides many avenues for generating such links, and this tip takes a look at two techniques for handling this task under a variety of circumstances.

Using handy character data

One approach is to use a character data field in the data: attribute or element content. Say we have a document in the DocBook vocabulary. Just for fun, for Listing 1 I've picked a silly summary of periods in British drama. It's intentionally brief to save space here, but you should get the idea if you imagine that each section would actually go on at length rather than for one brief paragraph.

```
<?xml version='1.0'?>
<article>
  <sect1>
    <title>Elizabethan</title>
    <para>Shakespeare his histories, Jonson his charlatans.</para>
  </sect1>
</article>
```
The top-level sections (`sect1` elements) would be an ideal point at which to divide the document (assuming you'd want to because the document had a more typical amount of text in each section). You can translate this to HTML using the transform in Listing 2, which is marked with lettered comments where details need further explanation.

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

 <xsl:template match="/"
 <html>
 <head/>
 <body>
 <h2>Contents</h2>

 <!-- A -->
 <xsl:apply-templates mode="toc"/>
 <hr/>

 <!-- B -->
 <xsl:apply-templates/>
 </body>
 </html>
 </xsl:template>

 <!-- C -->
 <xsl:template match="sect1" mode="toc">
 <a href="#{title}"><xsl:value-of select="title"></a><br/>
 </xsl:template>

 <!-- D -->
 <xsl:template match="sect1">
 <div>
 <h2><a name="{title}"><xsl:value-of select="title"></a></h2>
 <xsl:apply-templates/>
 </div>
 </xsl:template>

 <xsl:template match="title"/>

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

 </xsl:transform>
```
Part A of Listing 2

First, you generate the list of links to the content. XSLT modes are ideal for this. There is a special set of templates (or really a single template) to handle the special mode of generating the content links, called toc. This line invokes that mode. The article element has no matching template so it follows the default and applies templates to its children, maintaining the mode. The sect1 elements do find a matching template.

Part B of Listing 2

Apply-templates is called without a mode in order to render the content itself.

Part C of Listing 2

This handles the section elements in the toc mode. It prints the title, using the title also as the URL fragment identifier for the link.

Part D of Listing 2

This template handles the sections while rendering the content itself. The title is used as the link anchor, and the title is also used as the anchor name.

The rest of the transform is pretty straightforward. A template is set up to suppress the display of the title element during the regular template processing, since it is already explicitly printed using xsl:value-of. Listing 3 demonstrates the transform.

```
[uogbuji@borgia tip-internal-links]$ 4xslt xml1 xslt1
<html>
<head>
  <META HTTP-EQUIV='Content-Type'
       content='text/html; charset=UTF-8'>
</head>
<body>
<h2>Contents</h2>
<a href='#Elizabethan'>Elizabethan</a>
<br>
<a href='#Jacobean'>Jacobean</a>
<br>
<a href='#Georgian'>Georgian</a>
<hr>
<div>
<h2><a name='Elizabethan'>Elizabethan</a></h2>
<p>Shakespeare his histories, Jonson his charlatans.</p>
</div>
<div>
<h2><a name='Jacobean'>Jacobean</a></h2>
<p>"Webster was much possessed by death..."</p>
</div>
<div>
<h2><a name='Georgian'>Georgian</a></h2>
```
Note that this method relies on you having a handy and unique string available in a character data field. Ideally, this would be an attribute of type ID so that the XML parser could ensure its uniqueness for you. Also, attributes of type ID have format limitations that make them suitable for use as a URL fragment identifier.

**Using generate-id()**

Unfortunately there isn’t always a handy field to grab to use for naming the anchor and internal link. Take as an example the document in Listing 4, which is also in DocBook form.

```xml
<?xml version='1.0'?><article>
  <sect1>
    <title>The Burial of the Dead</title>
    <para>Summer surprised us that day, coming over the Starnbergensee.</para>
  </sect1>

  <sect1>
    <title>A Game of Chess</title>
    <para>But 0 0 0 0 that Shakespeherian rag. It's so elegant, so intelligent.</para>
  </sect1>

  <sect1>
    <title>Death by Water</title>
    <para>He passed the stages of his age and youth / Entering the whirlpool.</para>
  </sect1>
</article>
```

This second DocBook example in Listing 4 has section titles, but this time they are not suitable for making the link names. For one thing, they contain spaces, which would be converted by the XSLT processor to rather ugly URLs, which might not be so bad except that they are very long. This length might also cause problems with user agents. In this case, we have no handy field without such potential pitfalls. One could patch together an identifier using some heavy-hitting XSLT, but luckily there is another approach.

When you need to generate internal links and there is no convenient field that is unique, consistently available, and of suitable form for a URL fragment, you can use XSLT’s ability to generate a unique ID for you for a given node. The `generate-id()` function, defined in the XSLT spec (see Related topics), takes a node set and returns a string representing an identifier.
generated from the first node in the set, by document order. This identifier has two important properties: it is guaranteed to be different for different argument nodes, and it is guaranteed to be the same for identical argument nodes.

Armed with this knowledge, you write a transform like the one in Listing 5.

```xml
<?xml version="1.0"?>
<xsl:transform
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 version="1.0"
>
  <xsl:template match="/">
    <html>
      <head/>
      <body>
        <h2>Contents</h2>
        <xsl:apply-templates mode="toc"/>
        <hr/>
        <xsl:apply-templates/>
      </body>
    </html>
  </xsl:template>
</xsl:transform>

<!-- A -->
<xsl:template match="sect1" mode="toc">
  <a href="#{generate-id()}"><xsl:value-of select="title"/></a><br/>
</xsl:template>

<!-- B -->
<xsl:template match="sect1">
  <div>
    <h2><a name="{generate-id()}">\n      <xsl:value-of select="title"/>
    </a></h2>
    <xsl:apply-templates/>
  </div>
</xsl:template>
<xsl:template match="title"/>
<xsl:template match="para">
  <p><xsl:apply-templates/></p>
</xsl:template>
</xsl:transform>

**Part A of Listing 5**

Now rather than specify the link using the contents of an element, the `generate-id()` function is used. No node set is passed in so the current node is used for the node set generation. The current node is the `sect1` element for each section of the document.

**Part B of Listing 5**

Again, the `sect1` node is used to generate an ID, which for each node is identical to the ID generated in the corresponding link at the top of the page.

Listing 6 shows the output of the transformation.
Tip: Generating internal HTML links with XSLT

Listing 6. The output of the transformation in the previous listing

[uogbuji@borgia tip-internal-links]$ 4xslt xml2 xslt2
<html>
<head>
  <META HTTP-EQUIV='Content-Type'
  content='text/html; charset=UTF-8'></head>
<body>
<h2>Contents</h2>
<a href='#id137055808'>The Burial of the Dead</a>
<br>
<a href='#id137023880'>A Game of Chess</a>
<br>
<a href='#id137262848'>Death by Water</a>
<br>
<hr>
<div>
<h2><a name='id137055808'>The Burial of the Dead</a></h2>
<p>Summer surprised us that day, coming over the Starnbergensee.</p> </div>
<div>
<h2><a name='id137023880'>A Game of Chess</a></h2>
<p>But O O O O that Shakespeherian rag. It's so elegant, so intelligent.</p> </div>
<div>
<h2><a name='id137262848'>Death by Water</a></h2>
<p>He passed the stages of his age and youth / Entering the whirlpool.</p> </div>
</body>
</html>

[uogbuji@borgia tip-internal-links]$

Conclusion

There are many ways you could apply this technique for automating internal links; long documents are only one example. How about using it to index an FAQ page? In fact, the FAQ page I helped put together at 4Suite.org uses the generate-id() approach to provide a FAQ summary that links internally to the individual FAQ entries. You can use it in dynamic Web pages to create internal links consistently for each page view. I'm sure you'll find other times when these approaches come in handy.
Related topics

- The W3C's XSL page with many useful links to XSLT-related resources, including the specifications themselves, tutorials, articles, and implementations.
- The style sheet processor I used in the examples is our open-source tool 4XSLT, part of 4Suite.
- IBM trial software: Build your next development project with trial software available for download directly from developerWorks.
- Find more XML resources on the developerWorks XML zone. For a complete list of XML tips to date, check out the tips summary page.