XSL Conditionals and Looping

Martin Holmes

XSL Conditionals: ifs, chooses, whens and otherwises

- Almost all programming languages have **conditional branching** structures.
- XSL has two: `<xsl:if>` and `<xsl:choose>`.
- XPath also has the `if-then-else` structure.

Using `<xsl:if>`

A simple example: using `<xsl:if>` to pluralize "author" if there are multiple authors.

```
Author<xsl:if test="count($docAuthors/author) gt 1">s</xsl:if>:
<xsl:for-each select="$docAuthors/author">
  <xsl:value-of select="forename" /> <xsl:value-of select="surname" /><br/>
</xsl:for-each>
```

- Let's look at this in detail:

```
<xsl:if  test="count($docAuthors/author) gt 1">s</xsl:if>
```

- The `@test` attribute contains an **XPath expression** which evaluates to `true()` or `false()`.
- What is inside the `<xsl:if>` tag is only implemented if it evaluates to `true()`.

Using `<xsl:choose>`

- Sometimes you need to handle two or more conditions. This is done with `<xsl:choose>`:

-
  ```
  <div>
    <head>DHSI Dress Code</head>
    <xsl:choose>
        <xsl:when  test="surname='Bauman'">
                  tie, no footwear
        </xsl:when>
        <xsl:when  test="surname='Holmes'">
                  footwear, no tie
        </xsl:when>
        <xsl:otherwise>
                  unpredictable
        </xsl:otherwise>
    </xsl:choose>
  </div>
  ```
- The processor looks at each `<xsl:when>` in turn; when it finds one whose `@test` evaluates to `true()`, it processed that one, and then exits the `<xsl:choose>`.
- If none are true, it processes `<xsl:otherwise>` (assuming there is one).

Using XPath `if-then-else`

- If your condition is very simple, and the processing you want to do as a result of it does not involve creating tags and attributes, then you can just use an `if-then-else` structure in XPath:

-
  ```
  <xsl:value-of
        select="
              if (count($docAuthors/author) gt 1) then
                  'Authors: '
              else
                  'Author: '
     " />
  ```

XSL Conditionals: Task

Here's a simple task you can try:

- Open the *Places* XML: http://web.uvic.ca/~mholmes/dhoxss2013/examples/places.xml
- Open the Conditionals XSLT file: http://web.uvic.ca/~mholmes/dhoxss2013/examples/conditionals.xsl
- Switch to the XSLT debugger in oXygen.
- Follow the instructions in the XSLT file. You'll need to write some conditional XSLT code to complete the task.

XSL Conditionals: Task

This is one possible solution:

```
<xsl:template match="/">
  <xsl:for-each select="//place">
    <xsl:value-of select="placeName" />
    <xsl:choose>
      <xsl:when test="count(location/geo) eq 1"> (point)</xsl:when>
      <xsl:when test="location[@type='path']"> (path)</xsl:when>
      <xsl:otherwise> (polygon)</xsl:otherwise>
    </xsl:choose>
    <xsl:text>
</xsl:text>
  </xsl:for-each>
</xsl:template>
```

Looping in XSLT

Most programming languages have a looping construct, like this:

```
for (i = 0; i < 9; i++){
   alert('i = ' + i);
}
```

Looping in XSLT

XSLT has something similar:

```
<xsl:for-each select="//author">
  <xsl:value-of select="surname"/>
  <xsl:text>, </xsl:text>
  <xsl:value-of select="forename"/>
</xsl:for-each>
```

Looping vs Templates

You might wonder how this:

```
<xsl:for-each select="//author">
  <xsl:value-of select="surname"/>
  <xsl:text>, </xsl:text>
  <xsl:value-of select="forename"/>
</xsl:for-each>
```

is different from this:

```
<xsl:template match="author">
  <xsl:value-of select="surname"/>
  <xsl:text>, </xsl:text>
  <xsl:value-of select="forename"/>
</xsl:template>
[...]
<xsl:apply-templates select="//author">
```

In most cases, what you can do with looping can be done equally well with templates. Which approach you prefer is often a matter of personal preference; programmers used to more traditional programming languages may tend to use loops a lot (I do), whereas those more focused on pure XSLT are more likely to let templates do the work for them.

A pure template approach is known as push, whereas in pull contexts it's more common to find xsl:for-each.

One reason for looping: sorting

-
  ```
  <xsl:for-each select="//author">
    <xsl:sort select="surname"/>
    <xsl:value-of select="surname"/>
    <xsl:text>, </xsl:text>
    <xsl:value-of select="forename"/>
  </xsl:for-each>
  ```
- You can also sort according to the sort rules of another language, using the `@lang` attribute:

-
  ```
  <xsl:for-each select="//author">
    <xsl:sort select="surname" lang="is"/>
    <xsl:value-of select="surname"/>
    <xsl:text>, </xsl:text>
    <xsl:value-of select="forename"/>
  </xsl:for-each>
  ```

- To see sorting according to a specific language collation, see: http://web.uvic.ca/~mholmes/dhoxss2013/examples/sorting_collation.xsl

XSL Looping: Task

Here's a simple task you can try:

- Open the *Hamlet* XML: http://web.uvic.ca/~mholmes/dhoxss2013/examples/hamlet.xml
- Open the Looping XSLT file: http://web.uvic.ca/~mholmes/dhoxss2013/examples/looping.xsl
- Switch to the XSLT debugger in oXygen.
- Run the transformation. Then try adding an `<xsl:sort>` instruction to the loop.