10 XML, XSL and XSLT: Transforming XML

LEARNING OBJECTIVES

At the end of this chapter, you will be able to:

- Define XSLT and describe its use for transforming XML documents.
- Write an XSL style sheet to transform the XML document to XHTML.
- Use XPATH to access the parts of the XML documents.
- Describe and use XSL elements: value-of, for-each, sort, if and choices.
- Use the templates for transformation.
- Use JavaScript to transform XML on client side.

The aim of this chapter is to learn about and explore the possibilities of using XML as the starting point for data to be transformed into other target formats using XSLT. Style sheets are used and linked to documents. It is shown here that it is possible to process XML with a browser or a programing language on the client side.

XML allows the capture of meaning, or semantics, but it also has the ability to store data in a format that can be transformed into other types of document. This could be simply another XML document but with a different structure, or a totally different kind, such as XHTML, text or even PDF. By doing this you are keeping the meaning of the data away from forms it can be represented in and therefore increasing the flexibility. The XML document becomes the central store for information, which can then be output in whatever form is required for a particular context.

For example, if you choose to output your data in XHTML form you can harness the power of the markup to display that information in tables, frames or forms and highlight any information of particular interest with colors and emphasis.

In this chapter we explore the basics of how this is done so it is possible to transfer XML documents onto the Web and increase the usefulness of data stored in this way.

10.1 Introducing XSL

Where CSS is used as a style sheet language for HTML, XSL could be considered as the equivalent for XML. This is correct but it's not just a style sheet language. XSL is in fact really a name that covers a family of different languages:

1. XSLT This is XSL Transformations and will be the main area covered in this chapter. It concerns the transforming of XML documents from one syntax to another.

2. XSL:FO This is XSL Formatting Objects and is a way to visually transform XML to several formats such as PDF, PostScript and text files.
3. XPath This is used by XSLT to access, or to refer to, parts of XML documents.

Each of these is described in the W3C recommendations.

10.2 XML Transformed

XSLT could be considered one of the most important parts of XSL and became a W3C recommendation on 16 November 1999. It can be used to render XML into another kind of document. This could be another XML document or some other type, such as HTML or XHTML. Several browsers support an actual implementation of XSLT; these include Internet Explorer (using MSXML), Mozilla, Firefox and Netscape (using TransforMiiX). As usual, though, there is some variation in how well they do implement XSLT so it is wise to check with the specific browser and version. XSLT is used to change, and rearrange, elements, add and sort to build a new document. In effect it transforms a source XML tree into a resulting XML tree.

To do this, XSLT also uses another language that has already been mentioned in this chapter, XPath. This allows elements and attributes in the tree to be navigated. Using XPath, XSLT defines parts of an XML document that match a template. Once found and matched to the template the elements can then be transformed.

Checkpoint!

- XSL is the style sheet language for XML.
- The name covers several other language components such as XSLT, XSL:FO and XPath.
- XSL is concerned with transforming a source document to another type or format as a resulting document.
- XSLT will transform a document using XPath to specify and address parts of the source document's XML tree.
- Most browsers support an implementation of XSLT.

10.3 A Simple Example

The best way to begin is with a simple example. The following code is part of a simple XML document:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<library>
  <book>
  <title>A Scanner Darkly</title>
  <author>Philip K. Dick</author>
  <publisher>Gollancz</publisher>
  <price>6.99</price>
  <year>1985</year>
  <isbn>1-85798-847-7</isbn>
```

```
  </book>
  .
  .
  .
</library>
```

The next step is to create an XSL style sheet that will transform the XML document to the format we want, which in this case will be XHTML:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
 <html>
 <body>
   <h2>My Library</h2>
   <table border="1">
   <tr bgcolor="#9acd32">
     <th align="left">Title</th>
     <th align="left">Author</th>
   </tr>
   <xsl:for-each select="library/book">
   <tr>
     <td><xsl:value-of select="title"/></td>
     <td><xsl:value-of select="author"/></td>
   </tr>
   </xsl:for-each>
   </table>
 </body>
 </html>
</xsl:template>
</xsl:stylesheet>
```

Just like connecting the CSS style sheet to XHTML or HTML there needs to be a link between the two. This is done in the XML document:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="libstyle.xsl"?>
<library>
 <book>
   <title>A Scanner Darkly</title>
   <author>Philip K. Dick</author>
   <publisher>Gollancz</publisher>
   <price>6.99</price>
   <year>2002</year>
```

```
    <isbn>1-85798-847-7</isbn>
  </book>

  .
  .
  .
</library>
```

As you can see the second line has been added, which links to your XSL style sheet libstyle.xsl.

My Library

Title	Author
A Scanner Darkly	Philip K. Dick
Dr BloodMoney	Philip K. Dick
Ubik	Philip K. Dick

Figure 1 Output from Listing 1.

To actually see the result you will have to load the XML back into your XSLT-compliant brows
Figure 1 shows the output after adding a few more records (Listing 1).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="libstyle.xsl"?>
<library>
  <book>
    <title>A Scanner Darkly</title>
    <author>Philip K. Dick</author>
    <publisher>Gollancz</publisher>
    <price>6.99</price>
    <year>2002</year>
    <isbn>1-85798-847-7</isbn>
  </book>
  <book>
    <title>Dr BloodMoney</title>
    <author>Philip K. Dick</author>
    <publisher>Gollancz</publisher>
    <price>6.99</price>
    <year>2001</year>
    <isbn>1-85798-952-X</isbn>
  </book>
  <book>
    <title>Ubik</title>
    <author>Philip K. Dick</author>
    <publisher>Gollancz</publisher>
```

```
    <price>6.99</price>
    <year>2002</year>
    <isbn>1-85798-853-1</isbn>
  </book>
</library>
```

Listing 1 A simple XML document.

Checkpoint!

- XML can have an associated style sheet to transform its output.
- This is linked in to the XML document using a special style sheet reference.
- Apart from adding the reference there is no difference to the XML document.
- The style sheet must be available to perform the transformation.

The XML

As you can see from the XML used in this example, Listing 1 is not that different to what has been used before; there is a root element, `library`, followed by several book elements. Other items could belong to the library too, such as DVDs or music CDs.

The Style Sheet

The style sheet provides the template that transforms the document from one structure to another. In this case the `<xsl:template>` starts the definition of the actual template, as the name suggests. The `match="/"` attribute makes sure this begins applying the template to the root of the source XML document. Inside the template is the formatting for the output document, which is HTML and in this case a table. Probably the most interesting part of this is how it manages to visit each record of data stored in the XML. Here we can see a structure after the initial HTML that begins `<xsl:for-each select="library/book">`; this makes sure each book element is visited in turn and the code inside the `for-each` structure repeated in turn. Note where this structure ends and the code in-between. The data in each record is accessed by using:

```
<xsl:value-of select="title" />
```

and:

```
<xsl:value-of select="author" />
```

This is wrapped in the formatting commands for the HTML table that separate each item.

Linking

The style sheet is linked into the XML by adding the connecting statement to the XML document:

```
<?xml-stylesheet type="text/xsl" href="libstyle.xsl"?>
```

Concept Check

- Experiment with the above XML and XSL documents.
- Alter the output format so it simply lists all the data in the XML document.
- Alter the output again to create a table that contains all the stored data, complete with headings at the top of the table.

10.4 XSL Elements

XSL contains many elements that can be used to manipulate, iterate and select XML for output. Some of the more important ones are discussed here.

value-of

`Value-of` allows information to be extracted and added to the resulting document. The actual `select` attribute consists of an XPath expression that is similar in addressing parts to an operating system's file hierarchy with the forward slash selecting sub-directories.

for-each

The `for-each` element can be used to iterate and select through every XML element; again the `select` attribute contains an XPath.

sort

Information can also be sorted for output, simply by adding an `<xsl:sort>` element inside a `for-each`.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
  <body>
    <h2>My Library</h2>
    <table border="1">
    <tr bgcolor="yellow">
      <th align="left">Title</th>
      <th align="left">Author</th>
    </tr>
    <xsl:for-each select="library/book">
    <xsl:sort select="year"/>
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="author"/></td>
    </tr>
    </xsl:for-each>
    </table>
  </body>
```

```
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Listing 2 Using <xsl:sort>.

In Listing 2 sort is used to sort the information by year, although this could be used on any of the fields.

The <xsl:if> element allows the specifying of a query to extract information you may want. For example, Listing 3 has the <xsl:if> element:

```
<xsl:if test="year &gt; 2001">
```

This simply checks whether the year is greater than 2001 and in this case will only output those items that qualify, which is two of them.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
  <body>
    <h2>My Library</h2>
    <table border="1">
    <tr bgcolor="yellow">
       <th align="left">Title</th>
       <th align="left">Author</th>
    </tr>
    <xsl:for-each select="library/book">
    <xsl:if test="year &gt; 2001">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="author"/></td>
    </tr>
    </xsl:if>
    </xsl:for-each>
    </table>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

Listing 3 Using a condition.

Choices

Sometimes a multiple condition is required, or a default, in case the main condition is not met. For this it is possible to use the <xsl:choose> element, which is used with the <xsl:when> and <xsl:otherwise> elements. Using this structure it is possible to change output according to whether an item is found or highlight specific information. Listing 4 shows the structure and gives an example of use.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
  <body>
    <h2>My Library</h2>
    <table border="1">
    <tr bgcolor="yellow">
       <th align="left">Title</th>
       <th align="left">Author</th>
    </tr>
    <xsl:for-each select="library/book">
    <xsl:choose>
    <xsl:when test="year = 2001">
    <tr>
      <td bgcolor="red"><xsl:value-of
                    select="title"/></td>
      <td><xsl:value-of select="author"/></td>
    </tr>
    </xsl:when>
    <xsl:otherwise>
    <tr>
      <td bgcolor="green"><xsl:value-of
                    select="title"/></td>
      <td><xsl:value-of select="author"/></td>
    </tr>
    </xsl:otherwise>
    </xsl:choose>
    </xsl:for-each>
    </table>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

Listing 4 Multiple choices.

The structure follows this basic pattern:

```
<xsl:choose>
  <xsl:when test "test expression">
    output if true
  </xsl:when>
  <xsl:otherwise>
    output in other cases
  </xsl:otherwise>
<xsl:choose>
```

Applying Templates

The next example shows how to use `<xsl:apply-templates>`, which allows control over how templates can be applied, either to the current element or, if using `select`, to a specific element's nodes. Listing 5 initially applies a template to the root and then three other templates to control output to the screen for title, author and pricing information.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
      <h2>Book Library</h2>
      <xsl:apply-templates/>
  </body>
  </html>
</xsl:template>

<xsl:template match="book">
          <p>
          <xsl:apply-templates select="title"/>
          <xsl:apply-templates select="author"/>
          <xsl:apply-templates select="price"/>
          </p>
</xsl:template>

<xsl:template match="title">
Title: <span style="color:red">
<xsl:value-of select="."/></span>
<br />
</xsl:template>
<xsl:template match="author">
Artist: <span style="color:blue">
<xsl:value-of select="."/></span>
<br />
</xsl:template>
<xsl:template match="price">
Price: <span style="color:green">
<xsl:value-of select="."/></span>
<br />
</xsl:template>
</xsl:stylesheet>
```

Listing 5 Applying multiple templates.

The initial template match is with the root '/'. Within this there is a further `<xsl:apply-templates>` that activates the other templates, starting with the book-matching template. This then calls the others in sequence to give the desired output. Again, it is possible to use the usual HTML formatting to color and highlight the information.

Checkpoint!

- XSL uses templates, targeted by XPath, to change the resulting output document.
- XSL contains many elements to process XML; these include sorting, iteration and conditional selection.
- Templates can be applied in specific orders an output formatted as required.

Concept Check

- Expand the library XML document to include DVDs and CDs.
- Experiment with how you can look for specific information in an XML document, for example just the books or just the DVDS.
- Output each as a separate table.

10.5 Transforming with XSLT

As has been shown, it is possible to convert an XML document to XHTML using the browser's own parser. However, this is not always possible:

1. The browser at the client end may not be suitable or equipped to do the transformation.
2. It may not be a good idea to include the reference to the style sheet or even have the style sheet available!

In the case of the browser, it may be that it lacks the features required; for example, the actual transformation may be aimed at Braille or aural presentation. In the second case, including a reference within an XML document actually limits the XML to only that particular transformation and therefore gains flexibility.

The answer to this is to process the document and style sheet outside of the browser's own mechanisms for doing this task. This can be done either on the client side or the server side (which we will look at later).

Using JavaScript

One way to process and transform XML on the client side is using JavaScript, which has several features for doing the task very well.

```
<html>
<body>
<script type="text/javascript">
// Load the XML document
var xml = new ActiveXObject("Microsoft.XMLDOM")
xml.async = false
xml.load("lib.xml")
// Load the XSL document
var xsl = new ActiveXObject("Microsoft.XMLDOM")
```

```
xsl.async = false
xsl.load("libstyle.xsl")
// Do the actual transform
document.write(xml.transformNode(xsl))
</script>
</body>
</html>
```

Listing 6 Processing XML with JavaScript.

Listing 6 shows one ' ' to transform with JavaScript using Microsoft's proprietary Application Programing Interface (API) for the Internet Explorer browser. As you can see this is very simple indeed. It is also possible to process XML using the DOM, although as usual you have to be careful with compatibility between browsers. Using both of these mechanisms it is possible to also traverse an XML document and process either according to a style sheet or simply using the JavaScript to make the stylistic decisions.

Apart from JavaScript, it is also possible to use other programing languages (such as Java and.NET) to process and then output a transformed document.

Checkpoint!

- Client side transforming of XML does not have to be limited to the browser's implementation of XSLT.
- It is also possible to process XML using XSLT through languages such as JavaScript utilizing DOM or proprietary language features.
- It's also possible to process XML with languages without a style sheet at all.
- Separate processing allows a certain degree of flexibility as there is no need to reference a set XSL document within the XML.

Summary

1. XSL is a special style sheet language for XML.
2. XSL is composed of several languages, XSLT, XSL:FO and XPath.
3. XSL allows the transformation of XML from an initial source document to another kind of document, which can be XML, XHTML, text or some other type.
4. XPath is used to refer to specific parts of the source XML document.
5. Most browsers implement a version of XSLT and so can be used to transform documents from one format to another, given the XML document with its referenced style sheet.
6. Templates are used to match and then change part of the resulting document.
7. XSL contains many useful elements to sort, iterate and conditionally select parts of the XML document.
8. It is also possible to process XML outside of the browser, using client side languages. This can utilize the style sheet, which does not have to be referenced in the original XML, leading to more flexibility.
9. It is also possible to process XML without a style sheet with languages such as Java and JavaScript that contain special language features to parse the XML source tree.