Tailoring Content Using XML and XSL

Wei Meng Lee, Ngee Ann Polytechnic, Singapore

Back in the days of the Browser Wars, developers had to decide whether to produce a single set of documents compatible with all browsers, or develop multiple tailored sets. With the advent of WAP, the situation and the rules have changed: separating content and presentation is essential because of the increasing number of very different **user agents** (as browsers are now known). This paper will identify XML and XSLT as ideal tools for separating content and presentation, and show how these technologies can be used to overcome the inherent difficulties associated with the presentation of data to devices with limited capabilities.

Introduction

The advent of the Wireless Application Protocol (WAP) is a dream come true for consumers wanting information at their fingertips. However, it's a nightmare for web site operators. Not only must they maintain multiple sets of web pages for different web browsers, they now have to add a new set of documents to serve WAP users. To make matters worse, different WAP devices support different subsets of the standards.

One way for web site operators to approach the problem is to maintain two main sets of pages: HTML for the web users, and WML for the WAP users. However, this increases redundancy, and would mean maintaining multiple user interfaces for essentially the same content. A better option is to make use of the Extensible Markup Language (XML) and Extensible Stylesheet Language: Transformations (XSLT).

XML and XSLT

Let's briefly recap what XML is; XML is a meta-markup language. It is a set of rules for creating semantic tags used to describe data. While HTML is used to specify the layout of a web page, XML is used to describe data.

The following XML file (you'll find it in the accompanying material as `books.xml`) describes books in a library:

```
<?xml version="1.0" ?>
<Library>
   <Book Price="44">
      <ISBN>
         1861003129
      </ISBN>
      <Title>XSLT Programmer's Reference</Title>
      <Synopsis>
         XSL (eXtensible Stylesheet Language) is the styling
         language to match XML. At the most basic level it allows
         the programmer to manipulate XML on a template model:
         XSL provides the template into which to fit XML data for
         display on a web page
      </Synopsis>
      <URL>
         http://www.wrox.com/Consumer/Store/Details.asp?ISBN=1861003129
      </URL>
   </Book>
   <Book Price="54">
      <ISBN>
         1861003323
      </ISBN>
      <Title>Professional Visual Basic 6 XML</Title>
```

```
      <Synopsis>
         XML is the powerful new markup language that is rapidly
         changing the way data is handled, especially on the Web.
         Visual Basic is the ideal tool for creating applications
         to query and manipulate XML data
      </Synopsis>
      <URL>
         http://www.wrox.com/Consumer/Store/Details.asp?ISBN=1861003323
      </URL>
   </Book>
</Library>
```

XSL is an XML-based language that can be used to manipulate, sort, and filter XML data. The original XSL language has been further split into three parts:

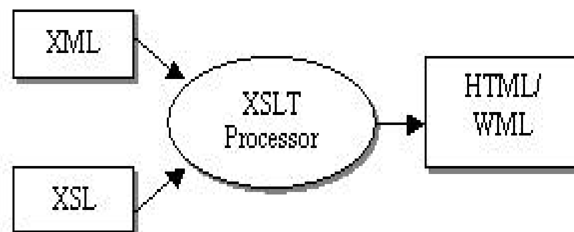
- Transformation (XSLT)
- Rendition (XSLF)
- XPath

XSLT enables you to define templates for your output, into which your XML data can be transformed.

Our Application

In this paper, I'm going to use Active Server Pages (ASP) to program the Microsoft MSXML component to transform a XML document to both HTML and WML, using XSLT. In a real application, the XML source can be generated from a database or otherwise dynamically created.

If you have installed Microsoft Internet Explorer 5.0, you already have a version of the MSXML component; however Microsoft has recently released an updated version of the XML parser, known as MSXML 3.0. The Microsoft XML Parser Technology Preview Release can be downloaded from http://msdn.microsoft.com/downloads/webtechnology/xml/msxml.asp



Using XSL provides a clear separation of content and display. Data can be stored in databases and retrieved as XML source, while the presentation information is kept inside an XSL stylesheet. The XSLT processor will transform the XML source to the required output, based on the XSL stylesheet.

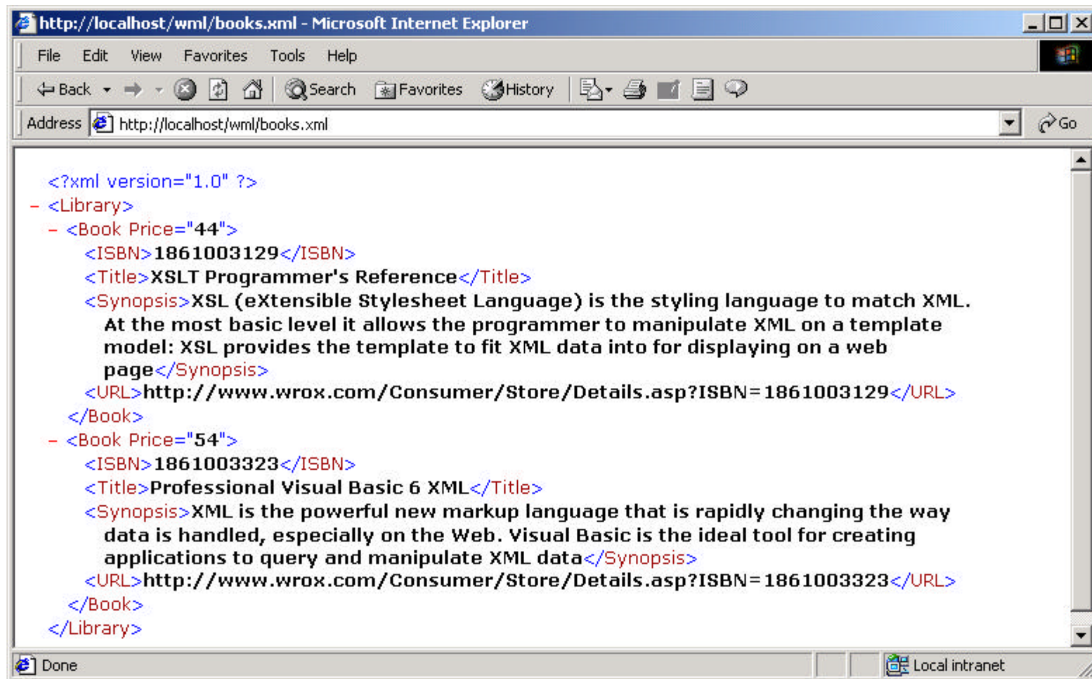
Transforming XML to HTML

Let's jump straight into an XSL stylesheet and see how it can transform an XML document into an HTML file. This is `books.xsl`:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
   <xsl:template match="/">
      <HTML>
         <BODY>
            <xsl:for-each select="Library/Book">
               <B><xsl:value-of select="Title" /></B><BR/>
               <I><xsl:value-of select="ISBN" /></I><BR/>
```

```
                <xsl:value-of select="Synopsis" /><BR/>
                <A>
                   <xsl:attribute name="href">
                      <xsl:value-of select="URL" />
                   </xsl:attribute>
                   <xsl:value-of select="URL" />
                </A>
                <BR/><BR/>
             </xsl:for-each>
          </BODY>
       </HTML>
    </xsl:template>
</xsl:stylesheet>
```

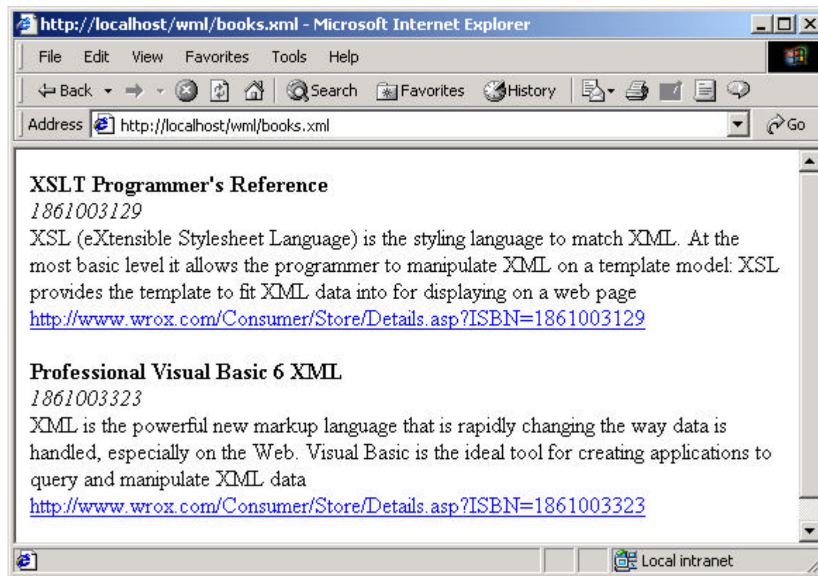
Now, load up the XML file we had earlier, using Microsoft Internet Explorer 5:



As you can see, if you don't specify a stylesheet in your XML file, IE5 will automatically use the default stylesheet and display the XML data in hierarchical format. Let's add a reference to the XSL stylesheet in the XML file:

```
<?xml version="1.0" ?>
<?xml:stylesheet type="text/xsl" href="books.xsl"?>
<Library>
```

Here, I've specified a processing instruction to indicate that this XML file should use the `books.xsl` stylesheet. Refresh the XML file loaded in IE5, and you'll see that the page is formatted nicely:



How It Works

Let's now take a closer look at the XSL stylesheet, and see how it transforms the XML source into an HTML file (XHTML, to be technically accurate). The first thing you should notice is that XSLT is an application of XML:

```
<?xml version="1.0" ?>
```

XSLT has elements that can be used to transform an XML file to the designated markup language. I'll introduce the XSLT elements that I use in this paper as I go along; they all have the following syntax:

```
<xsl:element>
```

An `<xsl:stylesheet>` element defines the set of templates to be applied to the input source tree to generate the output source tree. The `xmlns` attribute defines an XML namespace:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
```

An `<xsl:template>` element defines a template for the output:

```
   <xsl:template match="/">
```

The value of the `match` attribute is used to match against the XML source. In this instance, `"/"` matches against the root node of the XML source, and so the template is applied to that root node and everything inside it.

Since I'm generating a HTML file, we begin by specifying the beginning of the HTML document in the XSL stylesheet:

```
      <HTML>
         <BODY>
```

This will be inserted as is into the output document.

I need to extract *all* the information enclosed by the <Book> element in the XML file. The <xsl:for-each> element performs a loop, similar to that found in typical programming languages. The select attribute specifies the XSL pattern to be matched against the XML source:

```
            <xsl:for-each select="Library/Book">
```

The value of the select attribute "Library/Book" indicates that the processor should "look for any Book elements that are descendants of the Library element in the XML source". This element essentially loops through all the Book elements within the Library element.

Once the match is completed, I will use an <xsl:value-of> element to extract the value of the attribute defined in the XML source (which in this case is a <Book> element). The select attribute matches the element defined in the XML file. The value requested is then returned as text, which is inserted into the template:

```
               <B><xsl:value-of select="Title" /></B><BR/>
               <I><xsl:value-of select="ISBN" /></I><BR/>
               <xsl:value-of select="Synopsis" /><BR/>
```

To create a hyperlink in the target markup language (HTML in our case), I will use an <xsl:attribute> element. In our case, what I actually want is <A href="www.wrox.com/3129">www.wrox.com/3129</A>.

```
               <A>
                  <xsl:attribute name="href">
                     <xsl:value-of select="URL" />
                  </xsl:attribute>
                  <xsl:value-of select="URL" />
               </A>
```

This will insert an href HTML element into the code. As we saw previously, we could also do this by hard coding the value into the text.

Lastly, we have the closing tag for the <xsl:for-each> element:

```
            </xsl:for-each>
```

And the final closing tags for the HTML and XSLT tags:

```
         </BODY>
      </HTML>
   </xsl:template>
</xsl:stylesheet>
```

Dynamically Creating WML and HTML Using XML and XSLT

The above example has demonstrated the use of XSLT to transform XML into HTML, but it's quite possible to use the same technique to generate whatever markup language you desire.

However, the XSL stylesheet was embedded in the XML document, which does not help us to tailor our transformation to other platforms. What we need is the ability to transform the XML source into our desired target platform *dynamically*, during runtime.

Using ASP to Manipulate the XML Source

I'm going to demonstrate how to use Microsoft ASP to manipulate the XML source at the server side. By doing this server-side, we include thin clients that need any savings in processing we can give them. It also means we do not need to have XSLT support in the client. The process should be transparent to the client when the transformation is done server side.

To try out my example, you'll need to have a web server installed. If you're running Windows 95/98/NT, you can use the Microsoft Personal Web Server (PWS), available for download from Microsoft in the NT 4.0 Option Pack. This installs IIS 4.0 for NT and, PWS for Windows 95/98. It is also supplied with Visual Studio. If you're using Windows 2000, you can use Microsoft Internet Information Server (IIS) 4/5.

In addition, you'll also need to have the MSXML component installed on your system. If you've installed IE5, you have it already. However, I recommend that you install the latest preview release.

Using the XMLDOM Object

The MSXML DOM object represents the top level of the XML source. It has methods and properties allowing us to obtain or create all other XML objects. I will illustrate the use of the MSXML DOM object using server-side VBScript. Here's `books.asp`:

```
<%
Set xml = Server.CreateObject("MSXML2.DOMDocument")

' Halt execution until the document has been loaded
xml.async = false

' Load books.xml
xml.load (Server.MapPath("books.xml"))

Set xsl = Server.CreateObject("MSXML2.DOMDocument")
xsl.async = false

If InStr(Request.ServerVariables("HTTP_USER_AGENT"), "Moz") Then
   xsl.load(Server.MapPath("books.xsl"))
Else
   xsl.load(Server.MapPath("bookswml.xsl"))
   Response.ContentType = "text/vnd.wap.wml"
End If
Response.write (xml.transformNode(xsl))
%>
```

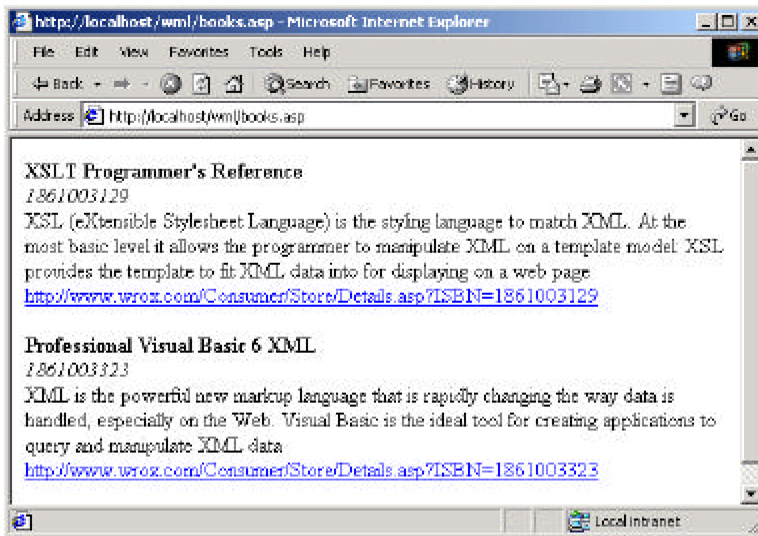
For the example that follows, the reference to the XSL stylesheet in the XML document must be removed. That is, the line,

```
<?xml:stylesheet type="text/xsl" href="books.xsl"?>
```

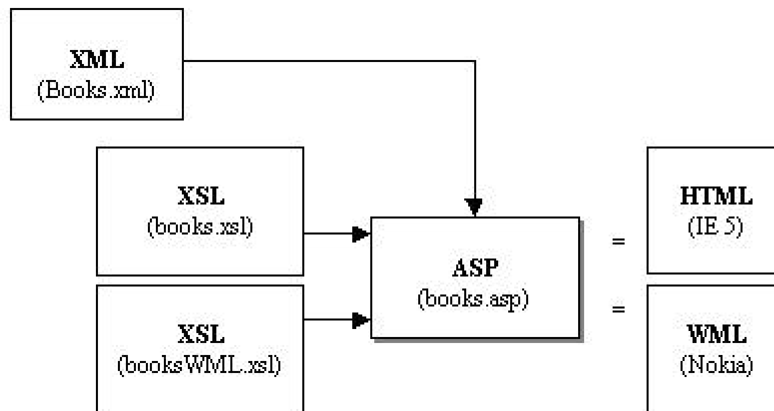
must be deleted.

Before I look into how the ASP document works, let's try to view the document using:

- Internet Explorer 5.0
- Nokia WAP emulator



And there you have it! You now have a single XML data source that is viewable on two radically different platforms. Let's recall and refine the diagram that you saw earlier:



In this case, we have two XSL stylesheets:

- `books.xsl` for generating HTML code
- `bookswml.xsl` for generating WML code

The `books.asp` document will load the XML document and transform it to the target markup language using the appropriate XSL stylesheet. I will discuss the `books.asp` document first, and after that we'll look at the `bookswml.xsl` stylesheet. To start with, then, we create an MSXML DOM object:

```
<%
Set xml = Server.CreateObject("MSXML2.DOMDocument")
```

Next, we set the `async` property to `false`, so that the XML document is fully loaded before control is transferred back to the script:

```
xml.async = false
```

After that, the XML source is loaded using the `load()` method of the MSXML DOM object:

```
xml.load(Server.MapPath("books.xml"))
```

The `Server.MapPath()` method is used to map the specified relative path to the physical directory on the server.

Once the XML source is loaded, we create another MSXML DOM object to load the XSL stylesheet:

```
Set xsl = Server.CreateObject("MSXML2.DOMDocument")
xsl.async = false
```

Parsing the XML Document

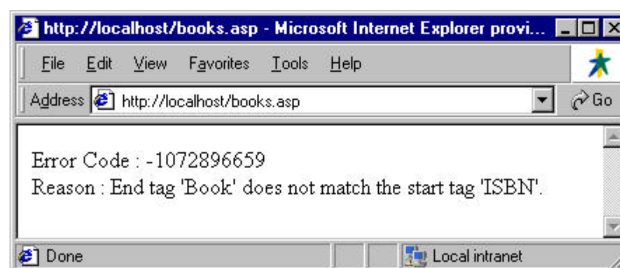
The `load()` method of the MSXML DOM object actually parses the XML source while it's loading. It will generate an error if the XML source is not well formed, and we can use the `parseError` property to view any error information. Suppose that our XML source has an error:

```
   <Book Price="44">
      <ISBN>
         1861003129
      <ISBN> <!--- missing "/" -->
      ...
```

We can modify the ASP file to trap the error generated by the XML parser:

```
xml.load(Server.MapPath("books.xml"))

If xml.parseError.errorcode <> 0 Then
   Response.write "Error Code : " & xml.parseError.ErrorCode & "<br/>"
   Response.write "Reason : " & xml.parseError.Reason
End If
```



Next, we check the client type. The `Request.ServerVariables` collection will contain the user agent string. If it contains the word `Mozilla`, it must be a web browser; if not, we assume the user is using a WAP device. This is a simplified way of differentiating a web browser from a WAP browser. (A more elaborate way would be to check the `USER_ACCEPT` string for the word `HTML` (for web browser) or `WML` (for WAP browser.)

```
If InStr(Request.ServerVariables("HTTP_USER_AGENT"), "Moz") Then
```

If the user agent is a web browser, we load the XSL stylesheet for HTML:

```
xsl.load(Server.MapPath("books.xsl"))
```

To transform the XML source to the designated HTML document, we use the `transformNode()` method of the MSXML DOM object, which processes the XML source using the supplied stylesheet. It returns a string containing the result of the transformation:

```
   Response.write(xml.transformNode(xsl))
```

If the user isn't using a web browser, we assume that they are using a WAP device, and therefore transform the XML document using the necessary XSLT elements:

```
Else
   xsl.load (Server.MapPath("bookswml.xsl"))
```

As a WAP device expects the WML MIME type, we use the `Response.ContentType` property to set the appropriate type:

```
   Response.ContentType = "text/vnd.wap.wml"

   Response.write(xml.transformNode(xsl))
```

Applying the Stylesheet

With that, we're ready to take a look at the XSL stylesheet for generating the WML content. Here's `bookswml.xsl`:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
   <xsl:template match="/">
      <xsl:eval no-entities="true">
         '<![CDATA[
            <?xml version="1.0"?>
         ]]>'
      </xsl:eval>
      <xsl:eval no-entities="true">
         '<![CDATA[
            <!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
               "http://www.wapforum.org/DTD/wml_1.1.xml">
         ]]>'
      </xsl:eval>
      <wml>
         <card id="card1" title="Books">
            <p>
               <xsl:for-each select="Library/Book">
                  <b><xsl:value-of select="Title" /></b><br/>
                  <i><xsl:value-of select="ISBN" /></i><br/>
                  <xsl:value-of select="Synopsis" /><br/>
                  <a>
                     <xsl:attribute name="href">
                        <xsl:value-of select="URL" />
                     </xsl:attribute>
                     <xsl:attribute name="title">
                        <xsl:value-of select="URL" />
                     </xsl:attribute>
                  </a><br/><br/>
               </xsl:for-each>
            </p>
         </card>
      </wml>
   </xsl:template>
</xsl:stylesheet>
```

To some extent, this is similar to the XSL stylesheet for generating HTML. For WML content, however, you need to include the following headers:

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
                     "http://www.wapforum.org/DTD/wml_1.1.xml">
```

The problem is that if I simply include these two lines into the XSL stylesheet, the XSLT processor will attempt to parse them, resulting in errors. What I need to do is preserve the two lines and not allow the parser to parse them.

I can use the `<![CDATA[string]]>` syntax to preserve the content of the string, and an `<xsl:eval>` element to evaluate the string:

```
<xsl:eval no-entities="true">
   '<![CDATA[
      <?xml version="1.0"?>
   ]]>'
</xsl:eval>
<xsl:eval no-entities="true">
   '<![CDATA[
      <!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
         "http://www.wapforum.org/DTD/wml_1.1.xml"
   >]]>'
</xsl:eval>
```

The rest of the stylesheet is then very similar to the one we had for HTML.

Alternatives

Instead of using the `<xsl:eval>` element to include the headers, I could also have sent them in the ASP document:

```
Response.ContentType = "text/vnd.wap.wml"
Response.Write "<?xml version=""1.0""?>"
Response.Write "<!DOCTYPE wml PUBLIC ""-//WAPFORUM//DTD " & _
       "WML 1.1//EN"" ""http://www.wapforum.org/DTD/wml_1.1.xml"">"
```

In this case, you have to be careful with the double quotes. If you want to output double quotes in your string, use a double quote character (which is simply two double quotes) where you wish one to appear. The code download includes this alternative.

Customizing WML for Different WAP Browsers

In the last section, we looked at how to generate content for display on both web browsers and WAP browsers. In this section, we will apply what we have learned to the art of tailoring WML content for display on *different* WAP browsers.

For this example, we will consider the Nokia browser and Phone.com's browser. I will use the Nokia WAP toolkit and the UP.Simulator. The source code for all the files involved is given below.

> *The browser-specific features mentioned in this section are based on the emulators. In real life, it is important for developers to test an application on each different platform before deploying it.*

Modules.xml

```
<?xml version="1.0" ?>
<Modules>
   <Module name="OS">
      <Title>Operating Systems</Title>
      <Description>
         This module teaches the fundamentals
         of Operating Systems
      </Description>
      <Year>1</Year>
      <URL>www.np.edu.sg/~OS</URL>
   </Module>
   <Module name="CSA">
      <Title>Computer Systems and Architecture</Title>
```

```
      <Description>
         This module teaches the fundamentals
         of Computer Systems
      </Description>
      <Year>2</Year>
      <URL>www.np.edu.sg/~CSA</URL>
   </Module>
   <Module name="WAD">
      <Title>Web Application Development</Title>
      <Description>
         This module teaches the basics
         of developing web applications
      </Description>
      <Year>3</Year>
      <URL>www.np.edu.sg/~WAD</URL>
   </Module>
</Modules>
```

Modules.asp

```
<%
Set xml = Server.CreateObject("MSXML2.DOMDocument")
xml.async = false
xml.load (Server.MapPath("modules.xml"))

Set xsl = Server.CreateObject("MSXML2.DOMDocument")
xsl.async = false

If InStr(Request.ServerVariables("HTTP_USER_AGENT"), "Nok") Then
   xsl.load(Server.MapPath("NokWML.xsl"))
Elseif InStr(Request.ServerVariables("HTTP_USER_AGENT"), "UP") Then
   xsl.load(Server.MapPath("UPWML.xsl"))
End If

Response.ContentType = "text/vnd.wap.wml"
Response.write (xml.transformNode(xsl))
%>
```

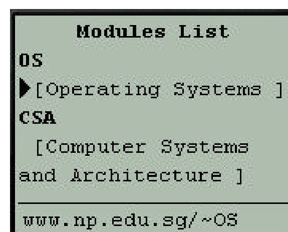
UPWML.xsl

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
   <xsl:template match="/">
      <xsl:eval no-entities="true">
         '<![CDATA[
            <?xml version="1.0"?>]]>
         '</xsl:eval>
      <xsl:eval no-entities="true">
         '<![CDATA[
            <!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
               "http://www.wapforum.org/DTD/wml_1.1.xml">
         ]]>'
      </xsl:eval>
      <wml>
         <card id="card1" title="Modules">
            <p align="center"><b>Modules List</b></p>
            <p>
               <xsl:for-each select="Modules/Module">
                  <b><xsl:value-of select="@name" /></b><br/>
                  <a>
                     <xsl:attribute name="href">
                        <xsl:value-of select="URL" />
                     </xsl:attribute>
                     <xsl:attribute name="title">
                        <xsl:value-of select="URL" />
                     </xsl:attribute>
                     <xsl:value-of select="Title" />
                  </a><br/>
               </xsl:for-each>
            </p>
         </card>
      </wml>
   </xsl:template>
</xsl:stylesheet>
```

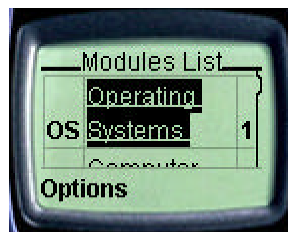
NokWML.xsl

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
   <xsl:template match="/">
      <xsl:eval no-entities="true">
         '<![CDATA[
            <?xml version="1.0"?>
         ]]>'
      </xsl:eval>
      <xsl:eval no-entities="true">
         '<![CDATA[
            <!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
               "http://www.wapforum.org/DTD/wml_1.1.xml">
         ]]>'
      </xsl:eval>
      <wml>
         <card id="card1" title="Modules List">
            <p>
               <table columns="3">
                  <xsl:for-each select="Modules/Module">
                     <tr>
                        <td>
                           <b><xsl:value-of select="@name"/></b>
                           <br/>
                        </td>
                        <td>
                           <a>
                              <xsl:attribute name="href">
                                 <xsl:value-of select="URL" />
                              </xsl:attribute>
                              <xsl:value-of select="Title" />
                           </a>
                        </td>
                        <td>
                           <b><xsl:value-of select="Year"/></b>
                           <br/>
                        </td>
                     </tr>
                  </xsl:for-each>
               </table>
            </p>
         </card>
      </wml>
   </xsl:template>
</xsl:stylesheet>
```

When a UP.Browser is used to view the page (`Modules.asp`), this is what is shown:



When a Nokia browser is used, the following will be seen:



The Modules.asp document detects the type of WAP device requesting the page, and loads the correct XSL stylesheet. In the case of the UP.Browser, the WML content generated does not contain a table, while the WML content for the Nokia browser contains a table with three columns. The reason for this is that the UP.Browser does not display tables correctly when there is an <a> element within a table cell. This is a good example of the different look-and-feel of current WAP 1.1 devices.

Retrieving Attribute Values

Finally, notice that the XML source has a <Module> element with a name attribute:

```
<?xml version="1.0" ?>
<Modules>
   <Module name="OS">
      <Title>Operating Systems</Title>
```

To extract the value of name in our XSL stylesheet, we use the @ character, followed by the attribute name:

```
                  <b><xsl:value-of select="@name" /></b><br/>
```

Summary

In this paper, we have seen the use of XML and XSLT to separate content from display. The combination of XML and XSLT provides a formidable pairing to help us tailor web content to suit different platforms. What I've demonstrated is just one of the many different possibilities that you can achieve with XML and XSLT.

I have also given an introduction to the various XSLT elements, and illustrated their usage with examples. As WAP gains in popularity, we should see XML and XSLT gaining wide acceptance among developers as a way of separating data from content.