How can I use XML?

XML (eXtensible Markup Language) provides a method to store information about data or metadata. But the nature of XML and the current state of tools make it difficult to decide how to best utilize this new technology. Part of the reason for this confusion is that XML is not designed to stand alone as a solution to enterprise data needs. In order to use XML effectively, most applications will need to rely on a number of complementary technologies. For this example, I will use XSL (eXtensible Stylesheet Language) and XQL (eXtensible Query Language) to use an XML data file and provide some insight into the uses of XML.

XML is a cross-platform standard and can be executed in a number of environments. However, due to the immaturity of XML tools at the time of this writing, it is difficult to use XML anywhere except within a server-based solution. To keep these examples simple, this sample application will use the Microsoft XML parser and output the results within HTML files you can run on your machine. I chose this combination because the only requirements for these examples will be the Microsoft Internet Explorer 5 browser (which includes the XML parser). These samples can easily be placed on any server platform and output as standard HTML for delivery to any client. For instance, the code will require very little modification to use the Sun Java parser within a Java servlet.

The data

The basis of many applications is typically the data that the applications make available to the user. The database application in my example is no different. I've used this particular program over many years, and it's the one I always use to teach others (and myself) new languages and technologies. The original version of this application was written on an Apple IIC computer in 1984. It has been through many revisions and platforms over the years, and now I am going to make it an XML-enabled application.

The data is my collection of music albums. Originally 8-tracks and cassettes, the collection now consists mainly of compact discs (after a break-in that left me with only one album and no equipment to play it on!) stored in a jukebox-style player. Each album is identified by the jukebox number and contains additional information about the album, such as the title, artist, year recorded, recording company, price (for insurance purposes), music category, media type (CD, cassette, 8-track, vinyl), and track information. The XML file for this application is called mymusic.xml, shown in **Listing 1**.

Listing 1: mymusic.xml

```
Listing 1 (mymusic.xml)

<?xml version="1.0"?>
<musiccollection>
 <album>
  <jukeboxnumber>1</jukeboxnumber>
  <mediatype>CD</mediatype>
  <musiccategory>Rock</musiccategory>
  <artist>Aerosmith</artist>
  <title>Permanent Vacation</title>
  <recordlabel>Geffen</recordlabel>
  <coverart>01permvac.jpg</coverart>
  <recordingyear>1987</recordingyear>
  <cost>0.00</cost>
  <track number="1">
   <title>Heart's Done Time</title>
   <length>4:42</length>
  </track>
  <track number="2">
   <title>Magic Touch</title>
   <length>4:40</length>
  </track>
  <track number="3">
   <title>Rag Doll</title>
   <length>4:21</length>
```

```
  </track>
  <track number="4">
   <title>Simoriah</title>
   <length>3:21</length>
  </track>
  <track number="5">
   <title>Dude (Looks Like a Lady)</title>
   <length>4:23</length>
  </track>
  <track number="6">
   <title>St. John</title>
   <length>4:12</length>
  </track>
  <track number="7">
   <title>Hangman Jury</title>
   <length>5:33</length>
  </track>
  <track number="8">
   <title>Girl Keeps Coming Apart</title>
   <length>4:12</length>
  </track>
  <track number="9">
   <title>Angel</title>
   <length>5:10</length>
  </track>
  <track number="10">
   <title>Permanent Vacation</title>
   <length>4:52</length>
  </track>
  <track number="11">
   <title>I'm Down</title>
   <length>2:20</length>
  </track>
  <track number="12">
   <title>The Movie</title>
   <length>4:00</length>
  </track>
 </album>
 <album>
  <jukeboxnumber>5</jukeboxnumber>
  <mediatype>CD</mediatype>
  <musiccategory>Rock</musiccategory>
  <artist>Dire Straits</artist>
  <title>Love Over Gold</title>
  <recordlabel>Warner Bros</recordlabel>
  <coverart>05direstraits.jpg</coverart>
  <recordingyear>1982</recordingyear>
  <cost>0.00</cost>
  <track number="1">
   <title>Telegraph Road</title>
   <length>14:15</length>
  </track>
  <track number="2">
   <title>Private Investigations</title>
   <length>6:45</length>
  </track>
  <track number="3">
   <title>Industrial Disease</title>
   <length>5:49</length>
  </track>
  <track number="4">
   <title>Love Over Gold</title>
   <length>6:16</length>
  </track>
  <track number="5">
   <title>It Never Rains</title>
   <length>7:54</length>
  </track>
```

```
</album>
<album>
  <jukeboxnumber>7</jukeboxnumber>
  <mediatype>CD</mediatype>
  <musiccategory>Rock</musiccategory>
  <artist>Dire Straits</artist>
  <title>Money For Nothing</title>
  <recordlabel>1988</recordlabel>
  <coverart>07direstraits.jpg</coverart>
  <recordingyear>1988</recordingyear>
  <cost>0.00</cost>
  <track number="1">
    <title>Sultans Of Swing</title>
    <length>5:46</length>
  </track>
  <track number="2">
    <title>Down To The Waterline</title>
    <length>4:00</length>
  </track>
  <track number="3">
    <title>Portobello Belle - Live</title>
    <length>4:33</length>
  </track>
  <track number="4">
    <title>Twisting By The Pool</title>
    <length>3:30</length>
  </track>
  <track number="5">
    <title>Tunnel Of Love</title>
    <length>8:09</length>
  </track>
  <track number="6">
    <title>Romeo And Juliet</title>
    <length>5:57</length>
  </track>
  <track number="7">
    <title>Where Do You Think You're Going</title>
    <length>3:31</length>
  </track>
  <track number="8">
    <title>Walk Of Life</title>
    <length>4:07</length>
  </track>
  <track number="9">
    <title>Private Investigations</title>
    <length>5:48</length>
  </track>
  <track number="10">
    <title>Telegraph Road - Live</title>
    <length>12:00</length>
  </track>
  <track number="11">
    <title>Money For Nothing</title>
    <length>4:05</length>
  </track>
  <track number="12">
    <title>Brothers In Arms</title>
    <length>4:48</length>
  </track>
</album>
<album>
  <jukeboxnumber>199</jukeboxnumber>
  <mediatype>CD</mediatype>
  <musiccategory>Classical</musiccategory>
  <artist>Various</artist>
  <title>Fantasia Disc 1</title>
  <recordlabel>Buena Vista</recordlabel>
  <coverart>199fantasia.jpg</coverart>
```

```
  <recordingyear>1990</recordingyear>
  <cost>0.00</cost>
  <track number="1">
   <title>Toccata and Fugue In D Minor (Bach)➥
   </title>
   <LENGTH/>
  </track>
  <track number="2">
   <title>Dance of the Sugar Plum Fairy ➥
   (Tchaikovsky)</title>
   <length>2:41</length>
  </track>
  <track number="3">
   <title>Chinese Dance (Tchaikovsky)</title>
   <length>1:02</length>
  </track>
  <track number="4">
   <title>Dance of the Reed Flutes</title>
   <length>1:48</length>
  </track>
  <track number="5">
   <title>Arabian Dance (Tchaikovsky)</title>
   <length>3:18</length>
  </track>
  <track number="6">
   <title>Russian Dance (Tchaikovsky)</title>
   <length>1:04</length>
  </track>
  <track number="7">
   <title>Waltz of the Flowers (Tchaikovsky)➥
   </title>
   <length>4:25</length>
  </track>
  <track number="8">
   <title>The Sorcerer's Apprentice (Dukas)➥
   </title>
   <length>9:17</length>
  </track>
  <track number="9">
   <title>Rite of Spring (Stravinsky)</title>
   <length>22:28</length>
  </track>
 </album>
 <album>
  <jukeboxnumber>200</jukeboxnumber>
  <mediatype>CD</mediatype>
  <musiccategory>Classical</musiccategory>
  <artist>Various</artist>
  <title>Fantasia Disc 2</title>
  <recordlabel>Buena Vista</recordlabel>
  <coverart>200fantasia.jpg</coverart>
  <recordingyear>1990</recordingyear>
  <cost>0.00</cost>
  <track number="1">
   <title>Symphony No. 6 I Allegro ma non ➥
   troppo (Beethoven)</title>
   <length>4:40</length>
  </track>
  <track number="2">
   <title>Symphony No. 6 II Andante molto ➥
   mosso (Beethoven)</title>
   <length>6:23</length>
  </track>
  <track number="3">
   <title>Symphony No. 6 III,IV,V Allegro ➥
   (Beethoven)</title>
   <length>10:57</length>
```

```
  </track>
  <track number="4">
   <title>Dance of the Hours (Ponchielli)➥
   </title>
   <length>12:13</length>
  </track>
  <track number="5">
   <title>A Night On Bald Mountain (Mussorgsky)➥
   </title>
   <length>7:25</length>
  </track>
  <track number="6">
   <title>Ave Maria, Op. 52 No. 6 (Schubert)➥
   </title>
   <length>6:27</length>
  </track>
 </album>
 <album>
  <jukeboxnumber>68</jukeboxnumber>
  <mediatype>CD</mediatype>
  <musiccategory>Hard Rock</musiccategory>
  <artist>AC/DC</artist>
  <title>Back In Black</title>
  <recordlabel>Atlantic</recordlabel>
  <coverart>68acdc.jpg</coverart>
  <recordingyear>1980</recordingyear>
  <cost>0.00</cost>
  <track number="1">
   <title>Hells Bells</title>
   <length>5:09</length>
  </track>
  <track number="2">
   <title>Shoot To Thrill</title>
   <length>5:14</length>
  </track>
  <track number="3">
   <title>What Do You Do For Money Honey</title>
   <length>3:33</length>
  </track>
  <track number="4">
   <title>Given The Dog A Bone</title>
   <length>3:30</length>
  </track>
  <track number="5">
   <title>Let Me Put My Love Into You</title>
   <length>4:12</length>
  </track>
  <track number="7">
   <title>You Shook Me All Night Long</title>
   <length>3:28</length>
  </track>
  <track number="8">
   <title>Have A Drink On Me</title>
   <length>3:57</length>
  </track>
  <track number="9">
   <title>Shake A Leg</title>
   <length>4:03</length>
  </track>
  <track number="10">
   <title>Rock and Roll Ain't Noise Pollution➥
   </title>
   <length>4:12</length>
  </track>
 </album>
</musiccollection>
```

One of the goals of the original XML specification was to make XML as simple and readable as possible, so Listing 1 is easy to read and understand. For this music database, a single element encompasses the entire XML file. The <musiccollection> element contains any number of <album> tags that contain the necessary information about the album, which in turn contains any number of <track> tags that store the information for the tracks of the current album. The <track> tag uses an attribute called number to indicate the track number on that album. Of course, the entire set of data for this file is much larger; mymusic.xml includes just a few albums for demonstration purposes.

My goal with this set of data is to provide the user with a way to display the album information and search it for particular songs. We can do many other things with this sample application, but the main focus is to provide an example of XML in action. The first step is to display the XML file by parsing the data and outputting the results in a Web page. To display XML with a minimum amount of programming, you can use XSL to apply a template to the XML and output the results.

Adding style

Listing 2 contains the XSL template necessary to output the musiccollection data.

Listing 2: musicstyle1.xsl

```
Listing 2 (musicstyle1.xsl)

<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
 <xsl:template match="/">
 <xsl:for-each select="musiccollection/album">
  <TR valign="top">
  <TD><xsl:value-of select="jukeboxnumber"/></TD>
  <TD><xsl:value-of select="musiccategory"/></TD>
  <TD><xsl:value-of select="artist"/></TD>
  <TD><xsl:value-of select="title"/></TD>
  <TD>
   <TABLE BORDER="0" width="100%">
    <xsl:apply-templates/>
   </TABLE>
  </TD>
 </TR>
 </xsl:for-each>
 </xsl:template>

 <xsl:template match="track">
 <TR>
  <TD><xsl:value-of select="@number"/></TD>
  <TD><xsl:value-of select="title"/></TD>
  <TD align="right"><xsl:value-of select="length"/></TD>
 </TR>
 </xsl:template>
</xsl:stylesheet>
```

The XSL file is where the true power of XML starts to make itself obvious. XSL uses the concept of templates to output XML elements. The main template, identified by <xsl:template match="/">, matches all elements in the document node passed into the style sheet. This main template starts with standard HTML for a table row of album information. A special XSL method is used to loop through all the album elements within the musiccollection node. The <xsl:for-each.../> construct loops over the items matched against the select attribute which, in this case, is given the value “musiccollection/album”. For each album, the <xsl:value-of…/> construct outputs the desired data for the current album.

This XSL file uses multiple templates to break up the code and make it easier to perform modifications. A call is made within the album loop to apply the additional templates. The command that does this is <xsl:apply-templates/>. The second template, <xsl:template match="track">, is applied every time a

track tag is encountered within the current album. This template outputs the information regarding the current track using the <xsl:value-of.../> tags.

Putting it all together

Now that the template and data are prepared, the application needs an output mechanism. Since I'm using HTML, this is easy. A single HTML file called musicdisplay1.html, shown in **Listing 3**, contains all the code needed for outputting.

Listing 3: musicdisplay1.html

```
Listing 3 (musicdisplay1.html)

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
 <HEAD>
  <SCRIPT LANGUAGE="JavaScript">
   var xmlParse = new ActiveXObject("Microsoft.xmldom");
   xmlParse.async = false;
   xmlParse.load("mymusic.xml");

   var xmlFormat = new ActiveXObject("Microsoft.xmldom");
   xmlFormat.async = false;
   xmlFormat.load("musicstyle1.xsl");

   function produceXML() {
   var tempHTML;
   tempHTML = "<TABLE BORDER=1><TR><TD>Juke Box</TD><TD>Category</TD>";
   tempHTML += "<TD>Artist</TD><TD>Album</TD><TD>Tracks</TD></TR>";
   tempHTML += xmlParse.transformNode(xmlFormat) + "</TABLE>";
   document.all.item("xmlData").innerHTML = tempHTML;
   }
  </SCRIPT>
 </HEAD>
 <BODY onload="produceXML();">
  <div id="xmlData"></div>
 </BODY>
</HTML>
```

The HTML document contains a single element in the body, "xmlData", that is used to receive the results of the parsed XML file. All the action on this page takes place with the JavaScript contained in the head section. The first six lines of code in the JavaScript block declare instances of the "Microsoft.xmldom" object and load the appropriate files (mymusic.xml and musicstyle1.xsl) into these objects. The async property is set to false so that each load is completed before continuing to the next line of code. In a high-performance application, you may prefer to leave this property true and select the readyState property, or use the ondataavailable event property to determine when the file is completely loaded. This would allow your code to continue performing other tasks while a large file was loading in the background.

The next few lines of code create the produceXML function, which does all the work in the document. The single line of code in this function is run once the HTML document is loaded, and it sets the innerHTML of the "xmlData" section to the results of the parsed XML. Calling the transformNode method of the xmldom object that contains the XML file and passing the xmldom object that represents the XSL file perform the transformation. The results are displayed in the browser using HTML tables to output each album and the corresponding tracks for that album. (I kept this very simple in this example.)

Searching

With the current code, we have successfully used XML to output the music collection in a browser window. One of the goals for this small sample application is to make it possible to search the XML file and return songs based on user-supplied search criteria. Searching is performed using XQL (eXtensible Query Language). XQL resembles SQL (Structured Query Language), which is used in popular database

products. To perform a search, the application will directly query the XML document and return a list of nodes that match the current query. This is a very straightforward way to query the data, but there is one problem with this method: The query returns a node list object that cannot be submitted to the XSL style sheet to be formatted. This is an issue with the current implementation that I hope future revisions of the standard will correct. Until then, I'll modify my approach to make this work properly. Revised XSL and HTML files are shown in **Listing 4** and **Listing 5**.

Listing 4: musicstyle2.xls

```
Listing 4 (musicstyle2.xls)

<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
 <xsl:template match="/">
  <xsl:apply-templates/>
 </xsl:template>
 <xsl:template match="album">
  <TR valign="top">
  <TD><xsl:value-of select="jukeboxnumber"/></TD>
  <TD><xsl:value-of select="musiccategory"/></TD>
  <TD><xsl:value-of select="artist"/></TD>
  <TD><xsl:value-of select="title"/></TD>
  <TD>
   <TABLE BORDER="0" width="100%">
   <xsl:apply-templates/>
   </TABLE>
  </TD>
  </TR>
 </xsl:template>

 <xsl:template match="track">
  <TR>
  <TD><xsl:value-of select="@number"/></TD>
  <TD><xsl:value-of select="title"/></TD>
  <TD align="right"><xsl:value-of select="length"/></TD>
  </TR>
 </xsl:template>
</xsl:stylesheet>
```

The XSL template has been changed to reflect the new type of data that is received for transformation. The first output generated by the application used the entire musiccollection or document node. A search cannot return the entire musiccollection node, as that would negate the search and return all the albums. Instead, the results of the search will return the album nodes of the XML file that contain a song title with the text indicated in the search. Since the XSL file will receive these album nodes, a template has been added to the XSL file to match the album node. This template contains the same code as the original template designed to match the root of the document, except that it doesn't perform a loop since it will receive one album node at a time.

Listing 5: musicdisplay2.html

```
Listing 5 (musicdisplay2.html)

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
 <HEAD>
  <SCRIPT LANGUAGE="JavaScript">
   var xmlParse = new ActiveXObject("Microsoft.xmldom");
   xmlParse.async = false;
   xmlParse.load("mymusic.xml");

   var xmlFormat = new ActiveXObject("Microsoft.xmldom");
   xmlFormat.async = false;
   xmlFormat.load("musicstyle2.xsl");

   function produceXML() {
```

```
    var tempHTML;
    var currentNode;
    var qryText;

    if (document.searchForm.searchText.value.length > 0) {
      qryText = "//album[track/title $ieq$ '" + document.searchForm.searchText.value + "']";
    }
    else {
      qryText = "//album"
    }
    var qry = xmlParse.selectNodes(qryText);

    tempHTML = "<TABLE BORDER=1><TR><TD>Juke Box</TD><TD>Category</TD>";
    tempHTML += "<TD>Artist</TD><TD>Album</TD><TD>Tracks</TD></TR>";

    for ( currentNode = qry.nextNode(); currentNode != null; currentNode = qry.nextNode() ) {
      tempHTML += currentNode.transformNode(xmlFormat);
    }

    tempHTML += "</TABLE>";
    document.all.item("xmlData").innerHTML = tempHTML;

    return false;
    }

    function resetMe() {
    document.searchForm.searchText.value = "";
    produceXML();
    }
  </SCRIPT>
 </HEAD>
 <BODY onload="produceXML();">
  <form name="searchForm" method="post" onSubmit="return produceXML();"
onReset="resetMe()">
    <input name="searchText" type="text" size="30">
    <input type="submit" value="Query Song Titles">
    <input type="reset" value="Clear Search">
  </form>
  <hr />
  <div id="xmlData"></div>
 </BODY>
</HTML>
```

There are a number of changes to the HTML file that reflect the addition of the search capability. The method used for searching is selectNode(<query string>). The selectNode function returns a collection of nodes indicated by the search pattern. Ignoring the syntax of the search pattern for a moment, take a look at the code in the produceXML() function. The function has been modified to return a value of false. This facilitates the assignment of the function to an HTML form. That form can then be used to submit search strings. By returning false, the form is never actually submitted, but we can capture the text the user types into the form.

Since the results of the selectNode function are valid XML nodes, it would seem that we could simply pass the results to the XSL file for transformation. Unfortunately, this is where I think the XML standard still needs to mature. XML is a well-formed language, and one of the requirements is that everything be properly contained. In HTML, it's possible to start font and bold tags but close them in a different order. In XML, this is not allowed. Since the album nodes returned from the search are no longer contained within the musiccollection node, the function must manually loop through the search result collection and transform each node individually. This is accomplished using the FOR loop within the productXML() function:

```
for ( currentNode = qry.nextNode(); currentNode != null; currentNode = qry.nextNode() )
```

Notice that the qry variable (which is a collection of nodes or a nodeList) has a method called nextNode(), which retrieves the nodes from the result set just like retrieving records from a recordset or resultset in ADO or JDBC. The FOR loop checks the current node to make sure it is not equal to null. Each node then calls the transformNode() method and stores results temporarily in a string variable. After the loop, the DIV element in the body of the HTML is set to the string variable, and the results are displayed to the user.

Search patterns

Now that the application has been modified to support the search capability, it's important to understand the search syntax or patterns. Search patterns are not difficult to understand once you have a grasp of a few basic syntax items. For the search implemented in this example, the pattern "//album" is used if the text on the form is blank (which is always the case when the page is first requested). The slashes tell the XSL processor the context in which to perform the search and, therefore, which nodes to return for the results. If only track information were desired instead of the entire album, the search could be modified to "//album/track". The double slashes are a shortcut for the document root and can be modified to "/musiccollection/album" instead of "//album".

Following the context of the search, the criteria must be specified. Without the criteria (such as when the search box is blank), all the album nodes would be returned. The search attempts to match the song titles that are in the track nodes. The entire search is enclosed by brackets. The search "[track/title ieq 'song title']" specifies that XSL should match track/title nodes and perform a case-insensitive equality. This functionality is very similar to the WHEN clause in SQL. **Table 1** summarizes a few of the more common comparison and Boolean logic operators.

Table 1: Comparison and Boolean logic operators

Equality	eq or =
Inequality	ne or =
Case-insensitive equality	ieq
Case-insensitive inequality	ine
Less than	lt or <
Less than or equal to	le or <=
Greater than	gt or >
Greater than or equal to	ge or >=
Logical and	and
Logical or	or
Logical not	not

By using these expressions, you can assemble many complex queries. In the example, ieq performs a case-insensitive match against the track titles and returns the album nodes that contain the track nodes that match the criteria. All data in XML is text, but when using these operators, the appropriate conversions will be made for numeric comparisons. The conversion is based on the value on the right side of the

comparison. If that value is a string, then string comparison will be used. If the value is an integer or real number, the XML values will be cast to long integer or doubles.

Conclusion

This small sample application demonstrates the power of XML as a data definition language. Through all the changes to the code, no changes were made to the XML data file. This is the intention of XML and one of the reasons for the rapid adoption of this technology. This example only scratches the surface of XML, XSL, and XQL. The power and capabilities of these technologies go much further. The XSL and XQL extensions are quite complex and require some time to master. However, XML is going to be more and more dominant as distributed Web applications take foot throughout the enterprise.