Construct Your E-commerce Business Tier the Easy Way With XML, ASP, and Scripting
Dave Cohen

XML acts like the ASCII of the Internet, allowing interaction between hosts regardless of their operating systems, database managers, or data formats. Add ASP and standard data formating options such as CSS or XSL and you can write some impressive network apps.

This article assumes you're familiar with XML and ASP

Code for this article: XML.zip (294KB)

Dave Cohen is an independent software consultant based in Chapel Hill, NC. He recently created a Visual InterDev and ASP Web-based training course for Instruction Set, and is working on one for Apache Web server. He can be reached at dlcohen@alleycatsw.com.

With the emergence of XML, it is now feasible to construct intelligent Internet agents that can interact reliably with multiple, diverse hosts. XML acts like the ASCII of the Internet, allowing interaction between hosts regardless of their operating systems, database managers, or data formats. Add ASP and standard data formatting options such as Cascading Style Sheets (CSS) or Extensible Stylesheet Language (XSL) to the mix and impressive network applications can be written without even a passing nod to the Java language, C++, sockets, or other complex tools.

In this article, I'll walk through a multitier application for Computer Finder, a fictitious company that offers users a Web page for specifying desired computer features, such as RAM, processor, and price. When the form is submitted, Computer Finder steps through its list of vendors, sending a query to each one based on the user-specified parameters. It then assembles the results and formats them for display to the user.

Along the way, I'll demonstrate a number of important technologies, including:

Multitier application configuration

ASP employed as a business services tier application (most multitier applications use COM components in this middle tier)

Real-time query and retrieval from multiple sources, rather than maintaining a centralized, possibly out-of-date database

Database retrieval using ADO

Use of XML to standardize data from diverse sources

Manual conversion of XML to HTML

XML Document Object Model (DOM) manipulation with JScript®

Formatting XML using XSL at the client and server

The Architecture

Computer Finder's system architecture is illustrated in Figure 1 . The basic flow is as follows. First, the user accesses Computer Finder's search form using a Web browser. Figure 2 shows an example of the HTML form presented to the user. The user specifies the features she wants for a computer system, such as RAM and disk size, as well as a price limit. Then the user clicks on the Submit button, sending the form to Computer Finder, which plays the role of an intelligent agent.

Figure 2 Computer Finder's Search Form

ors, sending each one a
ses the specified
ML for return to Computer
ne vendors and sends
splays an example of the

Figure 3 Search Results

The user services tier is based on HTML, with some CSS for formatting. The Web page doesn't use any script, but script could be added to do some local work such as verifying that numbers are entered into numeric fields.
While I've seen many examples of multitier applications that utilize XML, they all used the Java language or COM on the business services tier. However, as I'll demonstrate here, fairly complex business logic can be realized using ASP for the business services tier. In the Computer Finder app the business services tier is written with JScript and includes XML DOM manipulation. The data services tier also employs ASP in JScript that uses ADO to access the vendor's ODBC database.
Most ASP applications use VBScript since it is processed at the Web server and does not care which Web browser is used. However, I prefer to use JScript because it allows me to construct utility functions that can be included in both client and server pages. In my opinion, JScript is more powerful than VBScript in many aspects, although I do miss the simplicity of VBScript functions like Trim. Of course, all of the popular server components used with VBScript, such as ADO, are available to JScript as well.

The computer Document Type Definition (DTD) for describing computer features is illustrated in Figure 4. While a real-life DTD for computers would have a lot more information, this one includes enough complexity to create powerful searches. Along with the expected elements with attributes, the DTD also has a text node for the description element, attribute value lists for several elements such as operatingsystem, required and implied values for attributes, and a multiple element level under inputoutput. These features will aid in demonstrating in-depth DOM manipulation. Figure 5 shows the code for the complete computer DTD.

The User Services Tier
The Computer Finder search form (compuser.htm) is a standard HTML form that uses HTML tables for layout and some CSS for display formatting. For instance, a style class for labels is defined so I can experiment with and quickly change the appearance of form field labels.
The most important part of compuser.htm is the HTML form's target, compbusi.asp:

```
<form name="ComputerOptionsForm" action="compbusi.asp">
  <input type="hidden" name="HTMLCaller"
   value="compuser.htm"><table>
    <tr>
      <td><table border="0">
        <tr>
          <td class="LabelStyle">Price</td>
          <td><input type="text" name="Price" size="10"
              value="2000"><span class="UnitsStyle">
          (maximum)</span></td>
        </tr>
.
```

```
.
.
  </table>
</form>
```

The compbusi.asp target is the business service tier application, which is the heart of the intelligent agent processing. A complete listing of the user services HTML page is shown in Figure 6.
In Figure 2, the Display selection at the bottom of the form lets the user select a method for displaying the result. This is clearly out-of-place; you would usually let the ASP target page check the user's browser capabilities and use the most appropriate method. However, in this application I will be demonstrating a number of different formatting techniques. I'll delve into these format options later.

The Data Services Tier
The data services tier application (compdata.asp) will necessarily vary for each participating vendor. This ASP page's job is to take parameters from Computer Finder's agent and query the vendor database, then format the result as XML according to the computer DTD. The process is straightforward:

Retrieve the search parameters from the calling page's query string (BuildSQL).

Step through the search parameters, creating a SQL statement that matches the vendor's product database structure (BuildSQL).

Execute the SQL statement using ADO (BuildXML).

Step through the result rows of the database query, creating the XML (BuildXML).

One interesting twist is that the BuildSQL function allows for some variation in the exact values for arguments. For instance, if the OperatingSystem argument's upper-cased value contains at least SERVER or NTS, then Windows NT Server is assumed. The argument names themselves must match exactly, however.
After constructing the SQL statement, ADO (via JScript) is used to query the database and create the XML response. First, the XML header and <computers> top-level tag are written to the response object. Then, the vendor's product database is opened:

```
  objConn = Server.CreateObject("ADODB.Connection");
  objConn.Open("XYZComputers");
```

The SQL statement is executed, and the application steps through the returned rows:

```
objResult = objConn.Execute(strSQL);
if (!objResult.EOF)
{
  objResult.MoveFirst();
  while (!objResult.EOF)
```

```
  {
.
.
.
//write the corresponding XML to the Response object
.
.
.
     objResult.MoveNext();
  }
}
objConn.Close();
```

The columns from each row are then written out according to the DTD:

```
Response.Write("  <computer model=\"" +
  objResult("Model").Value + "\" price=\"" +
  objResult("Price").Value + "\">\n");
```

An interesting deviation you should be aware of is that the vendor's Web site, stashed in the WEB_SITE global variable, will be added to the returned data as the website attribute of the vendor element:

```
Response.Write("<vendor name=\"XYZ Computers\" " +
            "website=\"" + WEB_SITE + "\"/>\n");
```

A complete listing of the data services ASP page is shown in Figure 7.
Computer Finder would likely offer vendors a skeleton, or at least a bundle of common functions, to make construction of this page easier. Regardless, it is not very difficult if the vendor can access their product database with ADO.
While constructing the data services tier, I thought it would be neat if a tool existed that allowed you to specify a mapping between XML and an ODBC database. The tool would accept this mapping along with the XML and database schema, then spit out conforming XML.

The Business Services Tier
The business services tier application (compbusi.asp) is the most interesting—and most complex—part of the example. This ASP program's job is to receive a query request from the user, send the request to all of its known vendors, retrieve and combine results from the vendors, and format the result for the user.
The main body of compbusi.asp starts off by determining the URL of the page containing the form so a Return To link can be added on the results page:

```
strReturnURL = Request.ServerVariables("HTTP_REFERER");
```

After determining the requested output format by looking at the DisplayType form field value (more on this later), compbusi.asp gets down to business by instantiating an XML parser object:

```
 objXMLDocument = new ActiveXObject("microsoft.xmldom");
```

It uses the FileSystem server component to read the list of vendor URLs from a file on the server, sending the user query to each one and accumulating the XML query results:

```
 objVendorFileSystem = new
    ActiveXObject("Scripting.FileSystemObject");
 objVendorFile =
    objVendorFileSystem.OpenTextFile(VENDOR_FILE);

 while (objVendorFile.AtEndOfStream == 0)
 {
   strURL = objVendorFile.ReadLine();
   if (strURL.length > 0)
   {
     objXMLDocument.async = false;
     strURL = BuildURL(strURL);

     objXMLDocument.load(strURL);
     if ((objXMLDocument.parseError.errorCode == 0))
     {
       AddXMLBody(
objXMLDocument.documentElement);
     }
 .
 .
 .
   }
 }
```

The vendor URL that was read in specifies the vendor's domain and data services tier ASP page, such as http://www.xyzcomputers.com/compdata.asp. The BuildURL function is used to construct the full URL for the vendor by concatenating the query arguments (which came in via compuser.htm) to the base URL. This complete URL is used in the load method of the parser object, creating a parse tree from the XML results sent back by the data services tier ASP program. Finally, AddXMLBody walks the parse tree and appends the XML text to the strXML global variable.
Why bother parsing the returned XML, and then just turn around and recreate the XML text? There are several good reasons. First, it verifies that the XML returned by the vendor is well formed. Second, it assures that there are no unexpected elements or attributes in the vendor XML. Third, it allows the vendor XML to be easily manipulated, if necessary.

A lesser reason is that it serves to illustrate recursively walking the parse tree, as shown here in pseudocode:

```
AddXMLBody(node)
  loop through child nodes of node
    if child node is not an element node, then skip it
    get attributes of child node
    add the element opening tag to strXML
    loop through child nodes of this child node
      if child node is a text node,
        save its value and break
    end loop
    call AddXMLBody with child node (recursion)
    add any text node value to strXML
    add the closing element tag to strXML
  end loop
```

After closing the vendor URL file, the final task is to send the results to the user. As mentioned earlier, a field on the HTML form allows the user to specify the format option. While such an option would never really be presented to a user, I included it here so different formatting alternatives can be examined and tested. In real life, compbusi.asp would check the user's browser capabilities and determine the best method of formatting the result.
The complete source code for the business services ASP page is shown in Figure 8.

Formatting the Results
The business services tier application can format the vendor query results for presentation to the user via several methods. First, it can use plain text to display the XML as a text string. This is useful for debugging. The business services tier can also turn the XML into HTML for browsers that cannot handle XML. In this case the HTML specifies a CSS file in a <LINK> tag for displaying the results. It can also send the XML file back to the user. While the result will appear similar to using the plain text option, this method is significantly different in that it relies on the user's browser to format XML files as it sees fit.
Another option is using CSS to format the XML. This is not as powerful as using XSL, but is worth examining. The XML specifies the stylesheet processing command

```
<?xml-stylesheet type="text/css" href="compuser.csx"?>
```

so the XML and CSS files are sent to the user's computer for transformation. Unfortunately, a bug in either Microsoft Internet Explorer or ASP prevents this from working correctly. Using a static XML file that references a CSS file will work, but if the XML is created dynamically with ASP, the CSS file will not be sent back with the XML file.
You can use XSL to format the XML at the user's computer. The XML includes the stylesheet processing command

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

so the XML file and the XSL file are sent to the user's computer for transformation. Alternately, you can use XSL to format the XML at the server with the transformNode function. The result is that HTML is sent to the user. XML and XSL can therefore be used regardless of the browser's capabilities.
All of these formatting options are illustrated in Figure 9.

Figure 9 Result Formatting Options

Displaying the XML as HTML plain text requires the following steps:

Add the XML header to strXML.

Add the XML data string to strXML.

Add the XML footer to strXML.

Add the HTML header to strHTML.

Add strXML enclosed in <plaintext> tags to strHTML.

Add the HTML footer to strHTML.

Send the complete strHTML to the user.

Transforming the XML to HTML involves these steps:

Add the XML header to strXML.

Add the XML data string to strXML.

Add the XML footer to strXML.

Create a parse tree from the complete strXML result.

Add the HTML header with a CSS file reference to strHTML.

Use AddHTMLBody to convert the XML to HTML.

Add the HTML footer to strHTML.

Send the complete strHTML to the user.

The AddHTMLBody function walks the parse tree, creating an HTML table structure from the data. Unlike the AddXMLBody function shown earlier, AddHTMLBody does not recurse, but simply loops through the second-level elements of the tree. It's safe to do this since I created this XML from vendor-supplied XML.
Sending raw XML back to the client is simple:

Add the XML header to strXML.

Add the XML data string to strXML.

Add the XML footer to strXML.

Send the complete strXML to the user.

To return XML that is formatted with CSS:

Add the XML header to strXML.

Add the XML data string to strXML.

Add the XML footer to strXML.

Create a parse tree for the complete strXML result.

Use MassageXML to prepare the XML, modifying the parse tree.

Clear the existing strXML.

Add the XML header that references the CSS file to strXML.

Use AddXMLBody to create XML from the modified parse tree, adding it to strXML.

Add the XML footer to strXML.

Send the complete strXML to the user.

The MassageXML function serves to modify the XML data so CSS can operate on it. Unlike XSL, CSS cannot create tables, hyperlinks, and so on. This function turns attributes into text nodes enclosed in HTML table tags and adds an attribute for making the vendor Web site into a hyperlink. As with the AddHTMLBody function, I just loop through the second-level elements in the parse tree rather than using recursion.

This example illustrates that making XML get along with CSS may actually be more work than either translating the XML to HTML or using XSL. However, it does serve as good sample code for XML DOM manipulation, showing how to use methods such as appendChild and createTextNode in MassageXML.

These steps transform the XML with XSL at the client:

Add the XML header with an XSL file reference to strXML.

Add the XML data string to strXML.

Add the XML footer.

Send the XML to the user.

To transform the XML with XSL at the server and return HTML, do the following:

Create a parse tree from the XSL file.

Add the XML header to strXML.

Add the XML data string to strXML.

Add the XML footer to strXML.

Create a parse tree from the XML file.

Transform the XML parse tree using the XSL parse tree.

Send the resulting HTML to the user.

The transformNode method of the parser object takes care of the transformation from XML and XSL to HTML.

The last aspects of formatting are the stylesheets themselves. The compuser.css file (see Figure 10) is the CSS file used to format HTML in the method that directly converts XML to HTML. Also, the methods that transform XML with XSL add a reference to it in the resulting HTML. Finally, it is used in any HTML error pages returned to the user. In essence, it is the base file for colors, fonts, and other display basics.

The compuser.csx file (see Figure 11) is the CSS file used for the method that formats XML with CSS. I used a nonstandard file extension .csx to signify that it was CSS for XML.
The compuser.xsl file (see Figure 12) is the XSL file used for the methods that transform XML with XSL. The XSL file is fairly simple. It builds a table with three rows for each computer element: vendor name and URL link, computer description, and list of computer features. However, there are a couple of specifications in the XSL file that deserve further explanation.
First, I want the output data sorted by price, from lowest to highest. In the <xsl:template match="/"> section, I could add:

```
<xsl:apply-templates select="computers/computer"
order-by="+@price"/>
```

However, this would sort the prices alphabetically, so $500 would actually follow $1000. I need to sort numerically, but there's a dilemma: the DTD does not allow specification of data types. (One of the main features of the newer XML Schema approach is that elements can be typed.) XSL does offer a workaround via data type casting, so I cast price as a number:

```
<xsl:apply-templates select="computers/computer"
order-by="+number(@price)"/>
```

The second point of interest in compuser.xsl is how a hyperlink to the computer vendor's Web site is added to the HTML produced by the XSL. To do this, simply create the <A> tag with an HREF pointing to the desired URL:

```
<xsl:template match="vendor">
  <xsl:value-of select="@name"/>:
  <A>
    <xsl:attribute name="HREF">
      <xsl:value-of select="@website"/></xsl:attribute>
    <xsl:value-of select="@website"/>
  </A>
</xsl:template>
```

This XSL code results in this HTML:

```
<A HREF=http://www.xyzcomputers.com>XYZ Computers</A>
```

Setting Up the Application
To set up and run the application described in this article, follow these steps using the code that can be found at the link at the top of this article:

Copy the xyzcomp.mdb database to the PC serving as the vendor's server. Set up a System DSN with the name XYZComputers that points to xyzcomp.mdb.

Create a new Web site on the vendor's server, and copy the file compdata.asp to it.

Create a new Web site on the agent's Web server, and copy the compbusi.asp, vendurl.txt, compuser.htm, glass.gif, finder.gif, computer.dtd, compuser.css, compuser.csx, and compuser.xsl files to it. Set the URL in vendurl.txt to point to the URL for the data services ASP page (compdata.asp) on the vendor's Web site.

At the user's PC, use Internet Explorer to open the compuser.htm file on the agent's Web server.

To make things easier, the vendor and agent Web sites can be on the same PC. To really simplify things, you can install all elements on the same server and run Internet Explorer there as well.
To simulate multiple vendors, just replicate the vendor URL in vendurl.txt one or more times so that the database will be queried multiple times.

Conclusion
The flexibility of XML will certainly drive creation of agent applications like the one described here. One missing ingredient, however, is the creation and publication of standard XML schemas so businesses can openly interact with each other without going through the tedious process of setting up schemas for each relationship. The BizTalk™ initiative, spearheaded by Microsoft, is dedicated to publishing XML schemas for coordinating business-to-business electronic communications. See the sidebar "The Buzz about BizTalk" for more information.
The solution presented here shows how ASP and XML can be combined to create powerful multitier applications. Even better, perhaps, is the fact that server-side scripting with ASP further simplifies such solutions, since no additional language (such as the Java language) is needed at any point in the chain.
One caveat of this architecture is the use of XMLDOM.load, which uses WinInet. This is not necessarily the best fit for server execution since it is limited by default to only two concurrent connections to a given client (see Knowledge Base article 237906 for more information). To scale to the high load and concurrency requirements of the server, this should be rearchitected to use either a third-party or custom HTTP COM object using Winsock.
As the example stands, the performance of dependent Web servers would be tied to the performance of the vendor's server. Thus, the developer's app will block until the XMLDOM.load call returns. An even better suggestion would be to use a custom HTTP object in conjunction with MSMQ.

Figure 5 The Computer DTD

```
<!ELEMENT computers (computer*)>
<!ELEMENT computer (vendor?, description?, disk?,
                    monitor?, cpu, ram, operatingsystem,
                    inputoutput*, removabledisk?)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT disk EMPTY>
<!ELEMENT monitor EMPTY>
<!ELEMENT cpu EMPTY>
<!ELEMENT ram EMPTY>
<!ELEMENT operatingsystem EMPTY>
<!ELEMENT inputoutput (mouse?, joystick?, speakers?)>
<!ELEMENT mouse EMPTY>
<!ELEMENT joystick EMPTY>
<!ELEMENT speakers EMPTY>
<!ELEMENT removabledisk EMPTY>
<!ELEMENT vendor EMPTY>

<!ATTLIST computer model CDATA #REQUIRED
                   price CDATA #REQUIRED>
<!ATTLIST disk size CDATA #REQUIRED>
<!ATTLIST monitor size (15 | 17 | 19 | 21) "15">
<!ATTLIST cpu brand CDATA #REQUIRED
              speed CDATA #REQUIRED>
<!ATTLIST ram size CDATA #REQUIRED>
<!ATTLIST operatingsystem name (Win98 | WinNTW | WinNTS | Linux) "Win98">
```

```
<!ATTLIST mouse type (mouse | trackball | touchpad) "mouse">
<!ATTLIST removabledisk type (CDROM | CDRW | DVD) "CDROM">
<!ATTLIST vendor name CDATA #REQUIRED
            website CDATA #IMPLIED>
```

Figure 6 User Services HTML Page

```
<html>

<head>
<title>Computer Finder</title>
<meta name="GENERATOR" content="Microsoft FrontPage 3.0">
<style type="text/css">
  .LabelStyle   {font-size:18; font-face:Arial; color:black; font-weight:bold;}
  .UnitsStyle   {font-size:14; font-face:Arial; color:black; font-weight:bold;}
  .ButtonStyle  {font-size:18; font-face:Arial; color:black; font-weight:bold;
            height:40; width:150;}
  .InfoStyle    {font-size:18; font-face:Arial; color:green; font-weight:bold;}
</style>
</head>

<body background="glass.gif">
<!--
    User Services part of three-tier model.
    Form for user to select computer options.
    The server will use these specifications to query all of its
    known vendor sites.
-->

<p><img src="finder.gif" width="393" height="228" alt="finder.gif (5652
bytes)"><br>
</p>
<span class="InfoStyle">

<p>Enter your desired features and price,<br>
and we'll find the best deal on the Internet!</p>
</span>

<form name="ComputerOptionsForm" action="compbusi.asp">
  <table>
    <tr>
      <td><table border="0">
        <tr>
          <td class="LabelStyle">Price</td>
          <td><input type="text" name="Price" size="10" value="2000">
          <span class="UnitsStyle">
          (maximum)</span></td>
        </tr>
        <tr>
          <td class="LabelStyle">Disk</td>
```

```
  <td><input type="text" name="DiskSize" size="10" value="4">
  <span class="UnitsStyle"> GBs
  (minimum)</span></td>
</tr>
<tr>
  <td class="LabelStyle">Monitor</td>
  <td><input type="text" name="MonitorSize" size="10" value="15">
  <span class="UnitsStyle">
  inches (minimum)</span></td>
</tr>
<tr>
  <td class="LabelStyle">CPU Speed</td>
  <td><input type="text" name="CPUSpeed" size="10" value="300">
  <span class="UnitsStyle"> MHz
  (minimum)</span></td>
</tr>
<tr>
  <td class="LabelStyle">RAM</td>
  <td><input type="text" name="RAMAmount" size="10" value="32">
  <span class="UnitsStyle"> MBs
  (minimum)</span></td>
</tr>
<tr>
  <td class="LabelStyle">Operating<br>
  System</td>
  <td class="UnitsStyle"><select name="OperatingSystemType" size="1">
    <option selected>Windows 98</option>
    <option>Windows NT Workstation</option>
    <option>Windows NT Server</option></select></td>
</tr>
<tr>
  <td class="LabelStyle">Mouse</td>
  <td class="UnitsStyle"><select name="MouseType" size="1">
    <option selected>Standard</option>
    <option>Trackball</option>
    <option>Touchpad</option>
  </select></td>
</tr>
<tr>
  <td class="LabelStyle">CD Type</td>
  <td class="UnitsStyle"><select name="RemovableDiskType" size="1">
    <option selected>CDROM</option>
    <option>CD Read/Write</option>
    <option>DVD</option>
  </select></td>
</tr>
<tr>
  <td class="LabelStyle">Other Options</td>
  <td class="UnitsStyle"><input type="checkbox" name="JoystickOption"
  value="ON"> Joystick<input type="checkbox" name="SpeakersOption"
```

```
          value="ON">Speakers</td>
        </tr>
        <tr>
          <td class="LabelStyle">Display</td>
          <td class="UnitsStyle">
          <input type="radio" name="DisplayType" value="HTML" checked>HTML
          <input type="radio" name="DisplayType" value="TEXT">Text
          <input type="radio" name="DisplayType" value="CSS">XML/CSS
          <input type="radio" name="DisplayType" value="XSL">XML/XSL
          <input type="radio" name="DisplayType" value="XSL-S">
          XML/XSL (Server) </td>
        </tr>
        <tr>
          <td colspan="2"><input type="submit" name="SubmitButton"
value="Submit"
          class="ButtonStyle"> </td>
        </tr>
      </table>
      </td>
    </tr>
  </table>
</form></body></html>
```

Figure 7 Data Services ASP Page

```
<%@ language="JScript" %>
<%
// Data Services part of three-tier model.

// Company Web site for hyperlink in final user display.
var WEB_SITE = "http://www.xyzcomputers.com";

var strSQL; // Global string to hold SQL text.

function BuildSQL()
// Build the SQL query string, based on the parameters in the URL.
{
  var objString;

  strSQL = "SELECT Computers.Model AS Model, " +
           "      Computers.Price AS Price, " +
           "      Computers.Description AS Description, " +
           "      Computers.DiskSize AS DiskSize, " +
           "      Computers.MonitorSize AS MonitorSize, " +
           "      CPUs.Speed AS CPUSpeed, " +
           "      CPUs.Manufacturer AS CPUManufacturer, " +
           "      Computers.RAMSize AS RAMSize, " +
           "      OperatingSystems.Id AS OperatingSystemId, " +
           "      PointingDevices.Id AS PointingDeviceId, " +
           "      Computers.Joystick AS IncludeJoystick, " +
```

```
        "       Computers.Speakers AS IncludeSpeakers, " +
        "       RemovableDisks.Id AS RemovableDiskId" +
        " FROM  Computers, " +
        "       CPUs, " +
        "       OperatingSystems, " +
        "       PointingDevices, " +
        "       RemovableDisks " +
        " WHERE Computers.CPUId = CPUs.Id " +
        "   AND Computers.OperatingSystemId = OperatingSystems.Id " +
        "   AND Computers.PointingDeviceId = PointingDevices.Id " +
        "   AND Computers.RemovableDiskId = RemovableDisks.Id ";

// Should verify the argument types passed in, such as for numerics.
AddToWhere(Request.QueryString("Price").Item, "Computers.Price <=");
AddToWhere(Request.QueryString("DiskSize").Item, "Computers.DiskSize >=");
AddToWhere(Request.QueryString("MonitorSize").Item,
        "Computers.MonitorSize >=");
AddToWhere(Request.QueryString("CPUSpeed").Item, "CPUs.Speed >=");
AddToWhere(Request.QueryString("RAMSize").Item, "Computers.RAMSize >=");
if (IsAString(Request.QueryString("OperatingSystem").Item))
{
  objString = new String(Request.QueryString("OperatingSystem").Item);
  objString = objString.toUpperCase();
  if (objString.indexOf("98") >= 0)  { // Assume Windows 98
    AddToWhere("1", "OperatingSystems.Id =");
  }
  else if ((objString.indexOf("WORKSTATION") >= 0) ||
        (objString.indexOf("NTW") >= 0)) {
    // Assume "Windows NT Workstation".
    AddToWhere("2", "OperatingSystems.Id =");
  }
  else if ((objString.indexOf("SERVER") >= 0) ||
        (objString.indexOf("NTS") >= 0)) {
    // Assume "Windows NT Server".
    AddToWhere("3", "OperatingSystems.Id =");
  }
  else if (objString.indexOf("LINUX") != null) {
    AddToWhere("4", "OperatingSystems.Id =");
  }
  delete objString;
}
if (IsAString(Request.QueryString("Mouse").Item))
{
  objString = new String(Request.QueryString("Mouse").Item);
  objString = objString.toUpperCase();
  if (objString.indexOf("MOUSE") >= 0) {
    AddToWhere("1", "PointingDevices.Id =");
  }
  else if (objString.indexOf("TRACKBALL") >= 0) {
    AddToWhere("2", "PointingDevices.Id =");
```

```
    }
    else if (objString.indexOf("TOUCHPAD") >= 0) {
      AddToWhere("3", "PointingDevices.Id =");
    }
    delete objString;
  }

  if (IsAString(Request.QueryString("RemovableDisk").Item)) {
    objString = Request.QueryString("RemovableDisk").Item;
    objString = objString.toUpperCase();
    if ((objString.indexOf("CDROM") >= 0) ||
        (objString.indexOf("CD-ROM") >= 0)) {
      // Could be CD-ROM, CDROM.
      AddToWhere("1", "RemovableDisks.Id =");
    }
    else if ((objString.indexOf("CD") >= 0) &&
        ((objString.indexOf("WRITE") >= 0)
          || (objString.indexOf("RW") >= 0))) {
      // Could be CD Write, CD ReadWrite, CDRW.
      AddToWhere("2", "RemovableDisks.Id =");
    }
    else if (objString.indexOf("DVD") >= 0) {
      AddToWhere("3", "RemovableDisks.Id =");
    }
    delete objString;
  }
  if (IsAString(Request.QueryString("Joystick").Item)) {
    AddToWhere("1", "Computers.Joystick =");
  }
  if (IsAString(Request.QueryString("Speakers").Item)) {
    AddToWhere("1", "Computers.Speakers =");
  }
}

function AddToWhere(strArg, strComp)
// Add a constraint to the SQL 'Where' clause.
{
  if (IsAString(strArg)) {
    strSQL = strSQL + " AND " + strComp + " " + strArg + " ";
  }
}

function IsAString(something)
// Is the argument a string?
{
  if (typeof(something) == "string") {
    return (true);
  }
  else {
    return (false);
```

```
  }
}

function BuildXML()
// Build the XML, based on the query results. This will go back to the business
// services page, which will combine it with results from other vendors.
{
  var objConn;
  var objResult;

  Response.Write("<?xml version=\"1.0\"?>\n");
  Response.Write("<!DOCTYPE computers SYSTEM \"computer.dtd\">\n")
  Response.Write("<computers>\n");

  objConn = Server.CreateObject("ADODB.Connection");
  objConn.Open("XYZComputers");
  objResult = objConn.Execute(strSQL);
  if (!objResult.EOF)
  {
    objResult.MoveFirst();
    while (!objResult.EOF)
    {
      Response.Write("  <computer model=\"" + objResult("Model").Value
                + "\" price=\"" + objResult("Price").Value + "\">\n");
      // Add vendor information, which is not from query.
      Response.Write("    <vendor name=\"XYZ Computers\" " +
                "website=\"" + WEB_SITE + "\"/>\n");
      Response.Write("    <description>\n      "
                + objResult("Description").Value +
                "\n    </description>\n");
      Response.Write("    <disk size=\""
                + objResult("DiskSize").Value + "\"/>\n");
      Response.Write("    <monitor size=\""
                + objResult("MonitorSize").Value + "\"/>\n");
      Response.Write("    <cpu brand=\""
                + objResult("CPUManufacturer").Value +
                "\" speed=\"" + objResult("CPUSpeed").Value + "\"/>\n");
      Response.Write("    <ram size=\""
                + objResult("RAMSize").Value + "\"/>\n");
      if (objResult("OperatingSystemId").Value == "2")
      {
        Response.Write("    <operatingsystem name=\"WinNTW\"/>\n");
      }
      else if (objResult("OperatingSystemId").Value == "3")
      {
        Response.Write("    <operatingsystem name=\"WinNTS\"/>\n");
      }
      else if (objResult("OperatingSystemId").Value == "4")
      {
        Response.Write("    <operatingsystem name=\"Linux\"/>\n");
```

```
      }
      else { // default
      Response.Write("    <operatingsystem name=\"Win98\"/>\n");
      }
      Response.Write("    <inputoutput>\n");
      if (objResult("PointingDeviceId").Value == "2"){
        Response.Write("      <mouse type=\"trackball\"/>\n");
      }
      else if (objResult("PointingDeviceId").Value == "3"){
        Response.Write("      <mouse type=\"touchpad\"/>\n");
      }
      else { // default
        Response.Write("      <mouse type=\"mouse\"/>\n");
      }
      if (objResult("IncludeJoystick").Value){
        Response.Write("      <joystick/>\n");
      }
      if (objResult("IncludeSpeakers").Value){
        Response.Write("      <speakers/>\n");
      }
      Response.Write("    </inputoutput>\n");
      if (objResult("RemovableDiskId").Value == "2"){
        Response.Write("    <removabledisk type=\"CDRW\"/>\n");
      }
      else if (objResult("RemovableDiskId").Value == "3"){
        Response.Write("    <removabledisk type=\"DVD\"/>\n");
      }
      else { // default
        Response.Write("    <removabledisk type=\"CDROM\"/>\n");
      }
      Response.Write("  </computer>\n");
      objResult.MoveNext();
    }
  }
  objConn.Close();
  Response.Write("</computers>");
}

BuildSQL();  // Build the SQL statement from the URL arguments.
BuildXML();  // Execute the SQL statement, and create the XML result.
             // XML is sent to the business services page that called us.

%>
```

Figure 8 Business Services ASP Page

```
<%@ language="JScript" %>
<%

// Business Services part of three-tier model.
```

```
// Process user query by contacting all known vendors.
// The code also illustrates multiple output options.

// XML DOM node types.
var ELEMENT_NODE              = 1;
var ATTRIBUTE_NODE            = 2;
var TEXT_NODE                 = 3;
var CDATA_SECTION_NODE        = 4;
var ENTITY_REFERENCE_NODE     = 5;
var ENTITY_NODE               = 6;
var PROCESSING_INSTRUCTION_NODE = 7;
var COMMENT_NODE              = 8;
var DOCUMENT_NODE             = 9;
var DOCUMENT_TYPE_NODE        = 10;
var DOCUMENT_FRAGMENT_NODE    = 11;
var NOTATION_NODE             = 12;

var HTML_CSS_FILE             = "compuser.css";
var XML_CSS_FILE              = "compuser.csx";
var XML_XSL_FILE              = "compuser.xsl";

// Vendor file contains URLs of vendors to check.
var VENDOR_FILE = "e:\\inetpub\\wwwroot\\compxml\\vendurl.txt";

var intElemLevel = 0; // Global to count XML element level
                      // for recursive function.
var strReturnURL;     // Global for HTML query page.
var intNumResults;    // Number of query results
var strXML;           // Global XML string.
var strHTML;          // Global HTML string.

function AddXMLBody(objNode)
{
  // Gets XML query results from all vendors, combining results.

  var intNumElem;       // Number of elements in the parsed XML tree.
  var intNumAttr;       // Number of attributes for specific element.
  var intElemIndex;     // Element loop index.
  var intAttrIndex;     // Attribute loop index.
  var intNodeIndex;     // Element node loop index.
  var objThisElem;      // Current element.
  var objThisAttr;      // Current attribute.
  var objThisNode;      // Current node.
  var strNodeText;      // Node text string.
  var blnAnySub;        // Boolean flag.

  intElemLevel++;
  intNumElem = objNode.childNodes.length;
  for (intElemIndex = 0; intElemIndex < intNumElem; intElemIndex++)
  // Loop through child nodes for this element.
```

```
  {
    objThisElem = objNode.childNodes(intElemIndex);
    if (objThisElem.nodeType != ELEMENT_NODE) {
      // We know we are only dealing with element nodes at this level.
      continue;
    }
    strXML = strXML + indent(intElemLevel) + "<" + objThisElem.nodeName;
    intNumAttr = objThisElem.attributes.length;
    for (intAttrIndex = 0; intAttrIndex < intNumAttr; intAttrIndex++) {
      // Loop through attributes for this element.
      objThisAttr = objThisElem.attributes(intAttrIndex);
      strXML = strXML + " " + objThisAttr.nodeName + "=\""
              + objThisAttr.nodeValue + "\"";
    }
    blnAnySub = false;
    strNodeText = "";
    for (intNodeIndex = 0; intNodeIndex < objThisElem.childNodes.length;
         intNodeIndex++) {
    // See if this element has any element child nodes.
    // Also, look for text nodes.
      objThisNode = objThisElem.childNodes(intNodeIndex);
      if (objThisNode.nodeType == TEXT_NODE) {
        strNodeText = indent(intElemLevel+1) + trimws(objThisNode.nodeValue)
                    + "\n";
        blnAnySub = true;
        break; // Assume at most one text node per element.
      }
      else if (objThisNode.nodeType == ELEMENT_NODE) {
        blnAnySub = true;
      }
    }
    if (!blnAnySub) {
      strXML = strXML + "/>\n";
      continue;   // This element is complete.
    }
    // Recurse through children of this element node.
    strXML = strXML + ">\n" + strNodeText;
    AddXMLBody(objThisElem);
    strXML = strXML + indent(intElemLevel) + "</"
            + objThisElem.nodeName + ">\n";
  }
  intElemLevel--;
}

function indent(intNum)
// Indents XML, so it looks nicer.
{
  var strString = "";
  for (var intIndex = 0; intIndex < intNum; intIndex++) {
    strString = strString + "  ";
```

```
  }
  return(strString);
}

function trimws(strString)
// Trims white space from beginning and ending of a string.
// Could have cheated and used VBScript's Trim() function.
{
  var intBegin, intEnd;
  var strResult;
  var chrChar;
  var flg;

  intEnd = strString.length - 1;
  intBegin = 0;
  flg = true;
  chrChar = strString.charAt(intBegin);
  while ((intBegin < intEnd) && ((chrChar == "\n") || (chrChar == "\r") ||
                          (chrChar == "\t") || (chrChar == " "))) {
    // Find first non-whitespace.
    chrChar = strString.charAt(++intBegin);
  }
  chrChar = strString.charAt(intEnd);
  while ((intEnd >= 0) && ((chrChar == "\n") || (chrChar == "\r") ||
                      (chrChar == "\t") || (chrChar == " "))) {
    // Find last non-whitespace.
    chrChar = strString.charAt(--intEnd);
  }
  if (intBegin < intEnd) {
    strResult = strString.substr(intBegin, intEnd - intBegin + 1);
  }
  else {
    strResult = "";
  }
  return(strResult);
}

function BuildURL(strThisURL)
// Build the URL to send to the vendors, based on the user's query.
{
  var strNewURL;
  var strItem;
  var strString;

  strNewURL = trimws(strThisURL);
  if (strNewURL.length = 0) {
    return(strNewURL);
  }
  strNewURL = strNewURL + "?"; // Adding URL arguments.
  if (IsAString(strItem = Request.QueryString("Price").Item)) {
```

```
  strNewURL = strNewURL + "Price=" + strItem + "&";
}
if (IsAString(strItem = Request.QueryString("DiskSize").Item)) {
  strNewURL = strNewURL + "DiskSize=" + strItem + "&";
}

if (IsAString(strItem = Request.QueryString("MonitorSize").Item))
{
  strNewURL = strNewURL + "MonitorSize=" + strItem + "&";
}
if (IsAString(strItem = Request.QueryString("CPUSpeed").Item)) {
  strNewURL = strNewURL + "CPUSpeed=" + strItem + "&";
}
if (IsAString(strItem = Request.QueryString("RAMSize").Item)) {
  strNewURL = strNewURL + "RAMSize=" + strItem + "&";
}
// Turn Operating System query value into one suitable for vendor query.
if (IsAString(strItem = Request.QueryString("OperatingSystem").Item)) {
  strString = new String(strItem);
  strString = strString.toUpperCase();
  if (strString.indexOf("98") >= 0)  {
  // Assume "Windows 98".
    strNewURL = strNewURL + "OperatingSystem=Windows98";
  }
  else if ((strString.indexOf("WORKSTATION") >= 0) ||
          (strString.indexOf("NTW") >= 0)) {
  // Assume "Windows NT Workstation".
    strNewURL = strNewURL + "OperatingSystem=WindowsNTWorkstation";
  }
  else if ((strString.indexOf("SERVER") >= 0) ||
          (strString.indexOf("NTS") >= 0)) {
  // Assume "Windows NT Server".
    strNewURL = strNewURL + "OperatingSystem=WindowsNTServer";
  }
  else if (strString.indexOf("LINUX") != null)  {
    strNewURL = strNewURL + "OperatingSystem=Linux";
  }
  delete strString;
}
if (IsAString(strItem = Request.QueryString("MouseType").Item)) {
  strNewURL = strNewURL + "Mouse=" + strItem + "&";
}
if (IsAString(strItem = Request.QueryString("RemovableDisk").Item)) {
  strNewURL = strNewURL + "RemovableDisk=" + strItem + "&";
}
if (IsAString(strItem = Request.QueryString("JoystickOption").Item)) {
  strNewURL = strNewURL + "Joystick=Yes&";
}
if (IsAString(strItem = Request.QueryString("SpeakersOption").Item)) {
  strNewURL = strNewURL + "Speakers=Yes&";
```

```
  }

  if ((strNewURL.charAt(strNewURL.length-1) == "?") ||
      (strNewURL.charAt(strNewURL.length-1) == "&")) {
    strNewURL = strNewURL.substr(0,strNewURL.length-1);
  }
  return(strNewURL);
}

function IsAString(something)
// Check if the data item is a string.
{
  if (typeof(something) == "string") {
    return (true);
  }
  else {
    return (false);
  }
}

function AddHTMLBody(objNode)
// Build HTML query results from all vendors, combining results.
{
  var strVendorName;
  var strWebSite;
  var strModel;
  var strDescription;
  var strPrice;
  var strDiskSize;
  var strMonitorSize;
  var strCPU;
  var strRAM;
  var strOperatingSystem;
  var strMouse;
  var strSpeakers;
  var strJoystick;
  var strRemovableDisk;
  var intComputers;
  var objThisComputer;
  var intElemIndex;
  var intSubElemIndex;
  var intNumElem;
  var objThisElem;
  var intIOs;
  var objThisIO;
  var intIOIndex;
  var strThisVendor;

  // Start an HTML table.
  strHTML = strHTML + "<TABLE CLASS=\"DataTable\">\n";
```

```
strThisVendor = "";
intComputers = objNode.childNodes.length;
for (intElemIndex = 0; intElemIndex < intComputers; intElemIndex++)
// Loop through second level Computer elements.
{
  // Initialize query values.
  strVendorName        = "";
  strWebSite           = "";
  strModel             = "";
  strDescription       = "";
  strPrice             = "";
  strDiskSize          = "";
  strMonitorSize       = "";
  strCPU               = "";
  strRAM               = "";
  strOperatingSystem   = "";
  strMouse             = "";
  strSpeakers          = "";
  strJoystick          = "";
  strRemovableDisk     = "";

  objThisComputer = objNode.childNodes(intElemIndex);
  if (objThisComputer.nodeType != ELEMENT_NODE) {
    continue;
  }
  strModel = GetAttributeValue(objThisComputer,"model");
  strPrice = "$" + GetAttributeValue(objThisComputer,"price");
  intNumElem = objThisComputer.childNodes.length;
  for (intSubElemIndex = 0; intSubElemIndex < intNumElem;
      intSubElemIndex++) {
  // Assume all child elements of Computer element are one level down.
    objThisElem = objThisComputer.childNodes(intSubElemIndex);
    switch (objThisElem.nodeName) {
    case "vendor":
      strVendorName = GetAttributeValue(objThisElem,"name");
      strWebSite = GetAttributeValue(objThisElem,"website");
      break;
    case "description":
      if (objThisElem.firstChild.nodeType == TEXT_NODE) {
        // Assume text node is first child.
        strDescription = objThisElem.firstChild.nodeValue;
      }
      break;
    case "disk":
      strDiskSize = GetAttributeValue(objThisElem,"size") + " GB Disk";
      break;
    case "monitor":
      strMonitorSize = GetAttributeValue(objThisElem,"size")
                    + " in Monitor";
      break;
```

```
      case "cpu":
        strCPU = GetAttributeValue(objThisElem,"brand") + ": " +
                GetAttributeValue(objThisElem,"speed") + " Mhz";
        break;
      case "ram":
        strRAM = GetAttributeValue(objThisElem,"size") + " MB RAM";
        break;
      case "operatingsystem":
        strOperatingSystem = GetAttributeValue(objThisElem,"name");
        break;
      case "inputoutput":
        intIOs = objThisElem.childNodes.length;
        for (intIOIndex = 0; intIOIndex < intIOs; intIOIndex++) {
        // Loop through child nodes of InputOutput node.
          objThisIO = objThisElem.childNodes(intIOIndex);
          switch (objThisIO.nodeName) {
              case "mouse":
                  strMouse = "Mouse";
                  break;
          case "speakers":
            strSpeakers = "Speakers";
            break;
          case "joystick":
            strJoystick = "Joystick";
            break;
          }
        }
        break;
      case "removabledisk":
        strRemovableDisk = GetAttributeValue(objThisElem,"type");
        break;
      }
    }

    if (strVendorName != strThisVendor)
    // Add a Vendor row if this Computer is from a different vendor.
    {
      if (strThisVendor.length > 0)
      {
        strHTML = strHTML + "  <TR><TD
COLSPAN=\"10\"><HR></TD></TR>\n";
      }
      strHTML = strHTML + "  <TR>\n";
      strHTML = strHTML + "    <TD COLSPAN=\"10\"
CLASS=\"VendorLine\">\n";
      strHTML = strHTML + strVendorName + ": <A HREF=\"" +
                    strWebSite + "\">" + strWebSite + "</A><BR>\n";
      strHTML = strHTML + "    </TD>\n";
      strHTML = strHTML + "  </TR>\n";
      strThisVendor = strVendorName;
```

```
    }
    // Add row of data.
    strHTML = strHTML + "  <TR>\n";
    strHTML = strHTML + "    <TD COLSPAN=\"10\" CLASS=\"ModelLine\">\n";
    strHTML = strHTML + strModel + ": " + strDescription + "\n";
    strHTML = strHTML + "    </TD>\n";
    strHTML = strHTML + "  </TR>\n";

    strHTML = strHTML + "  <TR>\n";
    strHTML = strHTML + "    <TD CLASS=\"DataLine\">"
                  + strPrice + "</TD>\n";
    strHTML = strHTML + "    <TD CLASS=\"DataLine\">"
                  + strDiskSize + "</TD>\n";
    strHTML = strHTML + "    <TD CLASS=\"DataLine\">"
                  + strMonitorSize + "</TD>\n";
    strHTML = strHTML + "    <TD CLASS=\"DataLine\">"
                  + strCPU + "</TD>\n";
    strHTML = strHTML + "    <TD CLASS=\"DataLine\">"
                  + strRAM + "</TD>\n";
    strHTML = strHTML + "    <TD CLASS=\"DataLine\">"
                  + strOperatingSystem + "</TD>\n";
    strHTML = strHTML + "    <TD CLASS=\"DataLine\">"
                  + strRemovableDisk + "</TD>\n";
    strHTML = strHTML + "    <TD CLASS=\"DataLine\">"
                  + strMouse + "</TD>\n";
    strHTML = strHTML + "    <TD CLASS=\"DataLine\">"
                  + strSpeakers + "</TD>\n";
    strHTML = strHTML + "    <TD CLASS=\"DataLine\">"
                  + strJoystick + "</TD>\n";
    strHTML = strHTML + "  </TR>\n";
  }
  strHTML = strHTML + "  <TR><TD COLSPAN=\"10\"><HR></TD></TR>\n";
  strHTML = strHTML + "</TABLE>\n";
}

function GetAttributeValue(objNode,strAttr)
// Get a named attribute for an element.
{
  return objNode.attributes.getNamedItem(strAttr).nodeValue;
}

function GetParserError(objDocument)
// Get any errors from XML parser.
{
  var errstr;

  errstr = "<P CLASS=\"ErrorMessage\">" +
           "Invalid XML<BR>" +
           "URL: " + objDocument.parseError.url + "<BR>" +
           "Line: " + objDocument.parseError.line + "<BR>" +
```

```
        "Character: " + objDocument.parseError.linepos + "<BR>" +
        "Source Text: " + objDocument.parseError.srcText + "<BR>" +
        "Error Code: " + objDocument.parseError.errorCode + "<BR>" +
        "Description: " + objDocument.parseError.reason + "<BR>" +
        "</P>";
  return errstr;
}

function GetCurrPath()
// Get the current path of this ASP file.
{
  var strPath;
  var intPosition;

  strPath = Request.ServerVariables("PATH_TRANSLATED").Item;
  // Find the last backslash.
  for (intPosition = strPath.length - 1; intPosition >= 0; intPosition--)
  {
    if (strPath.charAt(intPosition) == "\\")
    {
      break;
    }
  }
  if (intPosition >= 0)
  {
    return strPath.substr(0, intPosition+1)
  }
  else
  {
    return "\\";
  }
}

function MassageXML(objDocument)
// CSS can't add labels, create tables, create hyperlinks, etc.
// Also, we have to assume the data is ordered correctly.
// So, we will massage the XML DOM in preparation for CSS.
// Actually, rather than go through this step, it's better to
// just create the HTML or use XSL - but this provides a chance
// to illustrate XML DOM manipulation.
//
// The massage process basically turns attributes into text nodes,
// complete with any preceding or trailing labels. It also
// adds an attribute to turn the vendor's Web site into a hyperlink.
//
// Like in AddHTMLBody, we are assuming that all Computer elements
// are one level down, and all child elements of Computer are one
// level below it.
{
  var intComputers;
```

```
var objThisComputer;
var intElemIndex;
var intSubElemIndex;
var intNumElem;
var objThisElem;
var strIO;
var intIOs;
var objThisIO;
var intIOIndex;
var objNewNode;
var objNode;

objNode = objDocument.documentElement;
intComputers = objNode.childNodes.length;
for (intElemIndex = 0; intElemIndex < intComputers; intElemIndex++)
// Loop through top-level Computer elements.
{
  objThisComputer = objNode.childNodes(intElemIndex);
  if (objThisComputer.nodeType != ELEMENT_NODE)
  {
    continue;
  }
  intNumElem = objThisComputer.childNodes.length;
  for (intSubElemIndex = 0; intSubElemIndex < intNumElem;
      intSubElemIndex++)
  // Loop through child elements of Computer element.
  {
    objThisElem = objThisComputer.childNodes(intSubElemIndex);
    switch (objThisElem.nodeName) {
    // Create text nodes for elements, as we want their values to appear.
    case "vendor":
      // Create an attribute for the Web Site attribute to
      // create hyperlink text.
      objNewNode = objDocument.createTextNode("<BR/>" +
        GetAttributeValue(objThisElem,"name") + ": " +
        "<A HREF=\"" + GetAttributeValue(objThisElem,"website") +
        "\">" + GetAttributeValue(objThisElem,"website") + "</A>");
      objThisElem.appendChild(objNewNode);
      break;
    case "description":
      if (objThisElem.firstChild.nodeType == TEXT_NODE) {
        objThisElem.firstChild.nodeValue = "<BR/>" +
          GetAttributeValue(objThisComputer,"model") + ": " +
          objThisElem.firstChild.nodeValue + "<BR/>";
      }
      break;
    case "disk":
      objNewNode = objDocument.createTextNode(
        GetAttributeValue(objThisElem,"size") + "GB Disk");
      objThisElem.appendChild(objNewNode);
```

```
      break;
    case "monitor":
      objNewNode = objDocument.createTextNode(
        GetAttributeValue(objThisElem,"size") + "in Monitor");
      objThisElem.appendChild(objNewNode);
      break;
    case "cpu":
      objNewNode = objDocument.createTextNode(
        GetAttributeValue(objThisElem,"brand") + ":" +
        GetAttributeValue(objThisElem,"speed") + "Mhz");
      objThisElem.appendChild(objNewNode);
      break;
    case "ram":
      objNewNode = objDocument.createTextNode(
        GetAttributeValue(objThisElem,"size") + "MB RAM");
      objThisElem.appendChild(objNewNode);
      break;
    case "operatingsystem":
      objNewNode = objDocument.createTextNode(
        GetAttributeValue(objThisElem,"name"));
      objThisElem.appendChild(objNewNode);
      break;
    case "inputoutput":
      strIO = "";
      intIOs = objThisElem.childNodes.length;
      for (intIOIndex = 0; intIOIndex < intIOs; intIOIndex++) {
        objThisIO = objThisElem.childNodes(intIOIndex);
        switch (objThisIO.nodeName) {
        case "mouse":
          strIO = strIO + " Mouse";
          break;
        case "speakers":
          strIO = strIO + " Speakers";
          break;
        case "joystick":
          strIO = strIO + " Joystick";
          break;
        }
      }
      objNewNode = objDocument.createTextNode(strIO);
      objThisElem.appendChild(objNewNode);
      break;
    case "removabledisk":
      objNewNode = objDocument.createTextNode(
        GetAttributeValue(objThisElem,"type"));
      objThisElem.appendChild(objNewNode);
      break;
    }
  }
}
```

```
}

function AddHTMLHeader(strCSSFile)
// Add HTML header stuff for HTML output.
{
  strHTML = "<HTML>\n<HEAD>\n";
  if (strCSSFile.length > 0) {
    // Add reference to CSS file.
    strHTML = strHTML
              + "<LINK REL=\"STYLESHEET\" TYPE=\"text/css\" HREF=\""
              + strCSSFile + "\">\n";
  }
  strHTML = strHTML + "</HEAD><BODY>\n";
  // Add a 'Return To' label.
  strHTML = strHTML + "<A HREF=\"" + strReturnURL
            + "\">Return to Computer Finder</A>\n" + "<HR>\n";
}

function AddHTMLFooter()
// Add HTML footer stuff for HTML output.
{
//  if (blnAddXML) { // This option is mainly for debugging.
//    strHTML = strHTML + "<plaintext>" + strXML + "</plaintext>";
//  }
  strHTML = strHTML + "</BODY></HTML>\n";
}

function AddXMLHeader(strXSLFile)
// Add XML header stuff for XML output.
{
  var strType;

  strXML = "<?xml version= \"1.0\"?>\n";
  strType = "";

  // See if XSL or CSS file passed in.
  if (strXSLFile.search(/\.xsl/i) > 0) {
    strType = "text/xsl";
  }
  else if ((strXSLFile.search(/\.csx/i) > 0) ||
           (strXSLFile.search(/\.css/i) > 0)) {
  // CSS = CSS file for HTML, CSX = CSS file for XML.
    strType = "text/css";
  }
  if (strType.length > 0) {
    // Add processing instruction for XSL or CSS file.
    strXML = strXML + "<?xml-stylesheet " + "type=\"" + strType + "\" "
             + "href=\"" + strXSLFile + "\"?>\n";
  }
```

```
  // If strXML subsequently parsed, then the full DTD path must be
  // supplied, thus the need for GetCurrPath. If parse from URL,
  // this is not needed since parser looks in the same directory.
  // strXML= strXML + "<!DOCTYPE computers SYSTEM \""
  // + GetCurrPath() + "computer.dtd\">\n";
  strXML = strXML + "<computers>\n";
}

function AddXMLFooter()
// Add XML footer stuff for XML output.
{
  strXML = strXML + "</computers>\n";
}

function OutputParserError(objDocument)
// Output an HTML page with XML parser error.
{
  AddHTMLHeader(HTML_CSS_FILE);
  strHTML = strHTML + GetParserError(objDocument);
  AddHTMLFooter();
  Response.Write(strHTML);
}

function OutputNoResultsError()
// Output an HTML page with no results error.
{
  AddHTMLHeader(HTML_CSS_FILE);
  strHTML = strHTML +
      "<P CLASS=\"ErrorMessage\">No matches found for query.</P>";
  AddHTMLFooter();
  Response.Write(strHTML);
}

var objVendorFileSystem;
var objVendorFile;
var strURL;
var objXMLDocument;
var strXMLBody;
var intDisplayType;
var DISPLAY_XML         = 0;
var DISPLAY_HTML        = 1;
var DISPLAY_CSS         = 2;
var DISPLAY_XSL         = 3;
var DISPLAY_XSL_SERVER  = 4;
var DISPLAY_TEXT        = 5;
var PARSER_ERROR        = 100;
var NO_RESULTS_ERROR    = 101;

intDisplayType = DISPLAY_XML;
```

```
// Set the 'Return To' hyperlink URL.
strReturnURL = Request.ServerVariables("HTTP_REFERER");

// See what kind of display was requested.
// In real life, we would look at the browser capabilities
// to determine this.
switch (Request.QueryString("DisplayType").Item) {
case "HTML":
// Format for HTML output.
  intDisplayType = DISPLAY_HTML;
  break;
case "CSS":
// Format for XML/CSS output.
  intDisplayType = DISPLAY_CSS;
  break;
case "XSL":
// Format for XML/XSL output.
  intDisplayType = DISPLAY_XSL;
  break;
case "XSL-S":
// Format for XML/XSL output, but do transformation here, not at client.
  intDisplayType = DISPLAY_XSL_SERVER;
  break;
case "TEXT":
// Format as plain text HTML.
  intDisplayType = DISPLAY_TEXT;
  break;
default:
// Format for XML/XSL output.
  intDisplayType = DISPLAY_XML;
  break;
}

objXMLDocument = new ActiveXObject("microsoft.xmldom");

// Open the file of vendor URLs.
objVendorFileSystem = new ActiveXObject("Scripting.FileSystemObject");
objVendorFile = objVendorFileSystem.OpenTextFile(VENDOR_FILE);

// Construct XML string by:
//   Loop through all the vendors:
//     Execute the vendor URL to retrieve database information.
//     Parse the retrieved XML.
//     Add to the XML string.
//
// There are more efficient ways of doing this, but this process assures:
//   Vendor XML is well formed.
//   There are no 'surprises' in the resulting XML sent to the user.
// Plus, it allows further manipulation of the XML from vendor to user,
// if XSL is not being used. XSL could handle the transformations.
```

```
strXML = "";
while (objVendorFile.AtEndOfStream == 0)
{
  strURL = objVendorFile.ReadLine(); // Get URL for next vendor.
  if (strURL.length > 0)
  {
    if (strURL.charAt(0) == "#") // comment
    {
      continue;
    }

    objXMLDocument.async = false;
    strURL = BuildURL(strURL);

    // Execute the vendor ASP page, then parse the result.
    // Note: load method loads a specified file. XML processor is smart
    //       enough to recognize the ASP extension, and process the page.
    objXMLDocument.load(strURL);
    if ((objXMLDocument.parseError.errorCode == 0))
    {
      AddXMLBody(objXMLDocument.documentElement); // Builds XML string.
    }
    else
    {
      intDisplayType = PARSER_ERROR;
      break;
    }
  }
}
strXMLBody = strXML; // Add headers and footers below.

objVendorFile.Close();

// Now see how many results we got.
intNumResults = 0;
AddXMLHeader("");
strXML = strXML + strXMLBody;
AddXMLFooter();
objXMLDocument.loadXML(strXML); // Parse the XML.
if ((objXMLDocument.parseError.errorCode == 0))
{
  intNumResults = objXMLDocument.documentElement.childNodes.length;
}
if (intNumResults == 0)
{
  intDisplayType = NO_RESULTS_ERROR;
}

switch (intDisplayType)
```

```
{
case PARSER_ERROR: // Parser error
  OutputParserError(objXMLDocument);
  break;

case NO_RESULTS_ERROR: // No results for query.
  OutputNoResultsError();
  break;

case DISPLAY_HTML: // Formatted HTML.
  // Construct HTML string by:
  //   Write the header.
  //   Process the parsed XML.
  //   Write the footer.

  AddXMLHeader("");
  strXML = strXML + strXMLBody;
  AddXMLFooter();
  objXMLDocument.loadXML(strXML); // Parse the XML.
  if ((objXMLDocument.parseError.errorCode == 0))
  {
    AddHTMLHeader(HTML_CSS_FILE);
    AddHTMLBody(objXMLDocument.documentElement); // Builds HTML string.
    AddHTMLFooter();
    Response.Write(strHTML);
  }
  else
  {
    OutputParserError(objXMLDocument);
  }
  break;

case DISPLAY_CSS:  // XML with CSS formatting
// This is supposed to work - but there is a problem.
// Whereas CSS in XML works (for the most part) in a static XML
// document, CSS is ignored if it is used in XML document
// dynamically generated by ASP.
  AddXMLHeader("");
  strXML = strXML + strXMLBody;
  AddXMLFooter();
  objXMLDocument.loadXML(strXML); // Parse the XML.
  if ((objXMLDocument.parseError.errorCode == 0))
  {
    MassageXML(objXMLDocument); // Massage XML for CSS.
    AddXMLHeader(XML_CSS_FILE);
    AddXMLBody(objXMLDocument.documentElement); // Builds XML string.
    AddXMLFooter();
    Response.Write(strXML);
  }
  else
```

```
  {
    OutputParserError(objXMLDocument);
  }
  break;

case DISPLAY_TEXT:
// Plain text HTML.
  AddXMLHeader("");
  strXML = strXML + strXMLBody;
  AddXMLFooter();
  AddHTMLHeader("");
  strHTML = strHTML + "<plaintext>" + strXML + "</plaintext>";
  AddHTMLFooter();
  Response.Write(strHTML);
  break;

case DISPLAY_XSL:
// XML formatted with XSL.
  AddXMLHeader(XML_XSL_FILE);
  strXML = strXML + strXMLBody;
  AddXMLFooter();
  Response.Write(strXML);
  break;

case DISPLAY_XSL_SERVER:
// XML formatted with XSL, transformed at server.
  strXSLPath = GetCurrPath() + XML_XSL_FILE;
  objXSLDocument = new ActiveXObject("microsoft.xmldom");
  objXSLDocument.load(strXSLPath);  // Parse the XSL file.
  if ((objXSLDocument.parseError.errorCode == 0) {
    AddXMLHeader("");
    strXML = strXML + strXMLBody;
    AddXMLFooter();
    objXMLDocument.loadXML(strXML); // Parse the XML string.
    if (objXMLDocument.parseError.errorCode == 0) {
      // Transform XML with XSL, creating HTML.
      Response.Write(objXMLDocument.transformNode(
                  objXSLDocument.documentElement));
    }
    else {
      OutputParserError(objXMLDocument);
    }
  }
  else {
    OutputParserError(objXSLDocument);
  }
  break;

default: // Raw XML
  AddXMLHeader("");
```

```
  strXML = strXML + strXMLBody;
  AddXMLFooter();
  Response.Write(strXML);
  break;
 }

 %>
```

Figure 10 CSS File for HTML Display Formatting

```
/*
   Cascading Style Sheet for computer data.
   This file is used with the HTML-formatted result.
*/

.VendorLine          {font-family:Arial; font-size:28; color:blue;}

.ModelLine           {font-family:Arial; font-size:16; font-style:italic;
                    font-weight:bold; color:navy;}

.DataTable           {}

.DataLine            {font-family:Arial; font-size:14;
                    color:black; border:outset;}

.ErrorMessage        {font-family:Arial; font-size:16; color:maroon;}
```

Figure 11 CSS File for XML Display Formatting

```
/*
   Cascading Style Sheet for computer data.
   This file is used for the XML result.
*/

computer       {display:block; margin:15px;}
vendor         {display:block; font-family:Arial; font-size:28; color:blue;}
description    {display:block; font-family:Arial; font-size:16;
                font-style:italic; font-weight:bold; color:navy;}
price          {display:inline; font-family:Arial; font-size:14; color:black;}
disk           {display:inline; font-family:Arial; font-size:14;
                color:black;}
monitor        {display:inline; font-family:Arial; font-size:14; color:black;}
cpu            {display:inline; font-family:Arial; font-size:14; color:black;}
ram            {display:inline; font-family:Arial; font-size:14; color:black;}
operatingsystem {display:inline; font-family:Arial; font-size:14; color:black;}
removabledisk   {display:inline; font-family:Arial; font-size:14; color:black;}
inputoutput     {display:inline; font-family:Arial; font-size:14; color:black;}
.ErrorMessage   {font-family:Arial; font-size:16; color:maroon;}
```

Figure 12 Transforming XML with XSL

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl"
xmlns="http://www.w3.org/TR/REC-html40" result-ns="">

<!--
   Extended Style Sheet for computer data.
   This file is used with the XML result.
-->

<xsl:template match="/">
  <HTML>
    <HEAD>
      <!-- We will be setting up the table and format in XSL,
           but use CSS for other stuff like colors, font, etc. -->
      <LINK REL="STYLESHEET" TYPE="text/css" HREF="compuser.css"/>
    </HEAD>
    <BODY>
      <A HREF="compuser.htm">Return to Computer Finder</A>
      <HR/>
      <TABLE CLASS="DataTable">
        <!-- Cast price element as number, so sorted numerically. -->
        <xsl:apply-templates select="computers/computer" order-
        by="+number(@price)"/>
      </TABLE>
    </BODY>
  </HTML>
</xsl:template>

<xsl:template match="computers/computer">
  <TR>
    <TD COLSPAN="10" CLASS="VendorLine">
      <xsl:apply-templates select="vendor"/>
    </TD>
  </TR>
  <TR>
    <TD COLSPAN="10" CLASS="ModelLine">
      <xsl:value-of select="@model"/>:
      <xsl:value-of select="description"/>
    </TD>
  </TR>
  <TR>
    <TD CLASS="DataLine">$<xsl:value-of select="@price"/></TD>
    <TD CLASS="DataLine"><xsl:apply-templates select="disk"/></TD>
    <TD CLASS="DataLine"><xsl:apply-templates select="monitor"/></TD>
    <TD CLASS="DataLine"><xsl:apply-templates select="cpu"/></TD>
    <TD CLASS="DataLine"><xsl:apply-templates select="ram"/></TD>
    <TD CLASS="DataLine"><xsl:apply-templates select="operatingsystem"/></TD>
    <TD CLASS="DataLine"><xsl:apply-templates select="removabledisk"/></TD>
    <xsl:apply-templates select="inputoutput"/>
```

```
  </TR>
</xsl:template>

<xsl:template match="vendor">
  <xsl:value-of select="@name"/>:
  <A>
    <xsl:attribute name="HREF"><xsl:value-of select="@website"/></xsl:attribute>
    <xsl:value-of select="@website"/>
  </A>
</xsl:template>

<xsl:template match="disk">
  <xsl:value-of select="@size"/> GB Disk
</xsl:template>

<xsl:template match="monitor">
  <xsl:value-of select="@size"/> in Monitor
</xsl:template>

<xsl:template match="cpu">
  <xsl:value-of select="@brand"/>:
  <xsl:value-of select="@speed"/> Mhz
</xsl:template>

<xsl:template match="ram">
  <xsl:value-of select="@size"/> MB RAM
</xsl:template>

<xsl:template match="operatingsystem">
  <xsl:value-of select="@name"/>
</xsl:template>

<xsl:template match="removabledisk">
  <xsl:value-of select="@type"/>
</xsl:template>

<xsl:template match="inputoutput">
  <TD CLASS="DataLine"><xsl:apply-templates select="mouse"/></TD>
  <TD CLASS="DataLine"><xsl:apply-templates select="speakers"/></TD>
  <TD CLASS="DataLine"><xsl:apply-templates select="joystick"/></TD>
</xsl:template>

<xsl:template match="mouse">
  Mouse
</xsl:template>

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

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

</xsl:stylesheet>
```

For related information see:
XML Developer Center at: http://msdn.microsoft.com/xml/default.asp.
Also check http://msdn.microsoft.com for daily updates on developer programs, resources and events.

From the February 2000 issue of Microsoft Systems Journal.
Get it at your local newsstand, or better yet, subscribe