



December 1999

[Download source files for this article.](#)

# Simplify Web Site Navigation with Server-Side XML

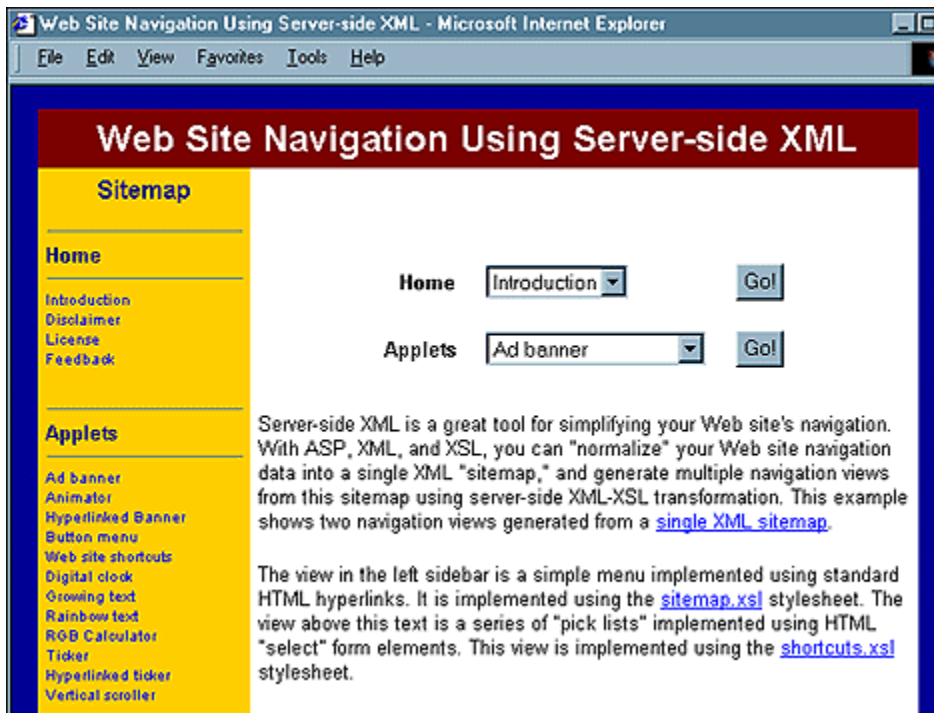
by **Andy Hoskinson**

In the area of Web development, the normalization of navigation data has been a perpetual problem for many Webmasters and developers. For instance, suppose you want to display a site map, organized into logical categories, as a menu of standard HTML hyperlinks. You also want the site map as a series of pick lists using HTML select form elements and some client-side JavaScript code. When given this task, you may have struggled to determine the easiest way to implement the navigation options. On one hand, you could use static HTML for both navigation views. However, in this scenario, you'd have to make site map changes in two places--obviously no good. And when a URL does change, you might forget to make the change in one spot, resulting in the dreaded HTTP 404 error for your visitors.

Another alternative might be to store the site map in a relational database, such as Microsoft SQL Server, and write components to retrieve the navigation data and display it in several different views. This works well and achieves the data normalization objective. Of course, by bringing a database server into the picture, you've cranked up the application's complexity level a couple of notches, making your job even harder. Moreover, the ASP application will incur the additional speed hits as it retrieves the navigation elements from a database server.

Fortunately, server-side XML provides a third alternative. As we've discussed in previous articles, XML enhances an ASP/IIS Web site's functionality and maintainability in a browser-independent manner. This month, we'll show you how to simplify your Web site's navigation using server-side XML technologies, as seen in Figure A.

**Figure A:** Our ASP page shows an XML site map as both a simple HTML menu in the left sidebar, and a pair of dropdown boxes in the main section



## A brief discussion of normalization

Database developers frequently talk about a concept called normalization. Simply put, *normalization* ensures that a specific piece of information exists only once in a given data store. That way, if the information changes, you only need to update it in one spot. By eliminating unnecessary duplication, normalization simplifies maintenance and ensures data integrity.

## Enter XML

XML is a great tool for simplifying a Web site's navigation. With ASP, XML, and XSL, you can normalize the site's navigation data into a single XML site map. What's more, you can encapsulate multiple navigation views from this site map using server-side XML-XSL transformation. This way, if your site map changes, you only need to alter one file. The runtime XML-XSL transformation propagates (in a virtual sense) the alteration throughout the Web site. This solution is robust, easy to implement, and extremely low-maintenance.

## The basic technique

For our example, we'll create two XSL navigation views from a single XML site map, as shown in Figure A. The left view will use standard HTML hyperlinks, while the second view in the page's main section will use a pair of dropdown boxes. We'll write a VBScript function that uses the Microsoft XML DOM to transform the XML into HTML, using the appropriate XSL stylesheet as a template. Next, we'll integrate this solution into an existing ASP application. Since the XML-XSL transformation occurs on the server, this page works in virtually any browser on the market. To begin, let's create the XML document.

## Create the XML site map

Listing A shows our well-formed XML site map. Just like any other XML document, Sitemap.xml

creates a tree hierarchy of nodes and their children. The root node, `SITEMAP`, contains a collection of one or more `CATEGORY` nodes. Each `CATEGORY` node consists of a `TITLE` attribute and one or more `PAGE` nodes. Each `PAGE` node encapsulates an individual page's title and URL.

### Listing A: Sitemap.xml code listing

```
<?xml version="1.0"?>
<SITEMAP>
<CATEGORY TITLE="Home">
  <PAGE URL="/java/default.asp" TITLE="Introduction"/>
  <PAGE URL="/java/disclaimer.asp" TITLE="Disclaimer"/>
  <PAGE URL="/java/license.asp" TITLE="License"/>
  <PAGE URL="/java/feedback.asp" TITLE="Feedback"/>
</CATEGORY>
<CATEGORY TITLE="Applets">
  <PAGE URL="/java/adbanner/default.asp" TITLE="Ad banner"/>
  <PAGE URL="/java/animator/default.asp" TITLE="Animator"/>
  <PAGE URL="/java/banner_links/default.asp" TITLE="Hyperlinked Banner"/>
  <PAGE URL="/java/buttonmenu/default.asp" TITLE="Button menu"/>
  <PAGE URL="/java/choiceMenu/default.asp" TITLE="Web site shortcuts"/>
  <PAGE URL="/java/clock/default.asp" TITLE="Digital clock"/>
  <PAGE URL="/java/growing_text/default.asp" TITLE="Growing text"/>
  <PAGE URL="/java/rainbow_text/default.asp" TITLE="Rainbow text"/>
  <PAGE URL="/java/rgb/default.asp" TITLE="RGB Calculator"/>
  <PAGE URL="/java/ticker/default.asp" TITLE="Ticker"/>
  <PAGE URL="/java/ticker_link/default.asp" TITLE="Hyperlinked ticker"/>
  <PAGE URL="/java/verticalScroll/default.asp" TITLE="Vertical scroller"/>
</CATEGORY>
</SITEMAP>
```

To author this document, you can use any ASCII text editor, such as Notepad or WordPad. You can also use one of the many free XML editors available for download from various Internet sites, such as Microsoft's XML Notepad. When you're through, save it as *Sitemap.xml*.

### Create XSL stylesheets for each navigation view

As our next step, we'll write an XSL stylesheet for each navigation view. We'll name the HTML menu stylesheet *sitemap.xsl*, and the dropdown box stylesheet as *shortcuts.xsl*. Listing B contains *sitemap.xsl*'s completed code, while Listing C shows *shortcuts.xsl*.

### Listing B: Sitemap.xsl code listing

```
<?xml version="1.0"?><DIV xmlns:xsl="http://www.w3.org/TR/WD-xsl"><xsl:for-each sele
</FONT><BR/></xsl:for-each></P></xsl:for-each></DIV>
```

### Listing C: Shortcuts.xsl code listing

```
<?xml version="1.0"?>
<DIV xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<CENTER>
<TABLE BORDER="0" CELLPADDING="10" CELLSPACING="0">
<xsl:for-each select="SITEMAP/CATEGORY">
<FORM METHOD="POST" ACTION="redir.asp" TARGET="_top">
<TR>
```

```

        <TD ALIGN="right">
            <FONT FACE="Arial" SIZE="2">
                <B><xsl:value-of select="@TITLE"/></B>
            </FONT>
        </TD>
        <TD><SELECT NAME="NewURL" SIZE="1"
ONCHANGE="top.location.href=this.form.NewURL
=>.options[form.NewURL.selectedIndex].value;">
            <xsl:for-each select="PAGE">
                <OPTION>
                    <xsl:attribute name="value">
                        http://www.hoskinson.net
                    <xsl:value-of select="@URL"/>
                </xsl:attribute>
                <xsl:value-of select="@TITLE"/>
            </OPTION>
        </xsl:for-each>
    </SELECT></TD>
    <TD><INPUT TYPE="submit" VALUE="Go!" /></TD>
</TR>
</FORM>
</xsl:for-each>
</TABLE>
</CENTER>
</DIV>

```

Both of our XSL stylesheets contain the standard HTML markup to define each view's presentation. In addition, interspersed among the HTML tags are special XSL tags that define how and where the XML data will be added to the resulting HTML output.

**Note:** Since an XSL stylesheet is an XML document itself, its markup must conform to XML grammar rules. Most notably, you must ensure that all tags are closed. Since HTML is considerably more forgiving in this aspect, you must ensure that all of the markup in your XSL sheets will successfully parse. For example, it's insufficient to use <BR> to express a line break--you must use <BR/> instead. To read more about the W3's recommendation for XHTML, which addresses this issue, see [www.w3.org/TR/xhtml1/](http://www.w3.org/TR/xhtml1/).

## VBScript function to perform XML-XSL transformations

Now that we've created the XML document for the XML site map and defined the navigation views using XSL, we must write a VBScript function that uses the Microsoft XML parser to perform the server-side XML-XSL transformation. This function, shown in Listing D, will be included in the ASP scripts.

### Listing D: TransformXML() VBScript function

```

function TransformXML(strXMLDoc, strXSLDoc)
on error resume next
dim objXMLDoc
dim objXSLDoc
dim strResults
const PROG_ID = "Microsoft.XMLDOM"
if err.number = 0 then
'Parse the XML Document
set objXMLDoc = server.CreateObject(PROG_ID)

```

```

objXMLDoc.async = false
objXMLDoc.load(strXMLDoc)if objXMLDoc.parseError.errorCode = 0 then
    'Parse the XSL stylesheet
    set objXSLDoc = server.CreateObject(PROG_ID)
    objXSLDoc.async = false
    objXSLDoc.load(strXSLDoc)
    if objXSLDoc.parseError.errorCode = 0 then
        'If no errors, transform the XML
        'into HTML using the XSL stylesheet
        strResults = objXMLDoc.transformNode(objXSLDoc)
    else
        strResults = "The following error " & _
            "occurred while processing the XSL " & _
            "stylesheet: <br>" & _
            objXSLDoc.parseError.errorCode & ", " & _
            objXSLDoc.parseError.reason
    end if
else
    strResults = "The following error " & _
        "occurred while processing the XML " & _
        "document: <br>" & objXMLDoc.parseError.errorCode & _
        ", " & objXMLDoc.parseError.reason
end if
else
    strResults = "The following error occurred: <br>" & _
        err.number & ", " & err.description
end if TransformXML=strResults
'Clean up
set objXSLDoc = nothing
set objXMLDoc = nothing
end function

```

As you can see, this function takes two arguments: the path to our XML site map, and the path to an appropriate XSL navigation view. First, it parses the XML document. Assuming that there are no errors, it parses the XSL stylesheet. If the stylesheet loads cleanly, it merges the XML document with the XSL stylesheet using the TransformNode() method of the XMLDOMNode object. This method essentially converts the XML document to HTML using the XSL stylesheet as a template. Finally, the function returns the resulting HTML to the calling application.

## Use XML navigation assets with existing ASP scripts

Your XML navigation assets are ready to go. To use them in an ASP application, simply paste this code into your ASP script at the place where you want the navigation asset to be displayed:

```

<%=TransformXML(Server.MapPath
=>("sitemap.xml"),Server.MapPath
=>("sitemap.xsl"))%>

```

Of course, you should make sure that the paths to your XML and XSL documents accurately reflect their location on your server. Also, you want to ensure that you use the correct XSL stylesheet for the particular navigation view you want to display. In the code snippet above, we used the stylesheet for the simple HTML menu. Finally, you make sure that the TransformXML() function is accessible to the ASP script. We recommend pasting this function into an include file, and including it in all of the ASP scripts that need to call it.

## Conclusion

In this article, we showed you how to use Microsoft's XML technologies on the server to simplify navigation of an ASP/IIS Web site in a browser-independent manner. Using XML, you can normalize your navigation data into a single XML site map, and create numerous views for that data using XSL. That way, if your site map changes in any way, you only need to change it in one place to propagate the change throughout your entire site. In this article, we've shown you how to use Microsoft's exciting XML technologies in an ASP application.

Copyright © 1999, ZD Inc. All rights reserved. ZD Journals and the ZD Journals logo are trademarks of ZD Inc. Reproduction in whole or in part in any form or medium without express written permission of ZD Inc. is prohibited. All other product names and logos are trademarks or registered trademarks of their respective owners.