



August 1999

Separating Content from Presentation with Server-Side XML

by Andy Hoskinson

Since the early days of the Web, developers, designers, and content providers have grappled with the problem of separating presentation and design from content and functionality. This article will show you how to accomplish this using the Microsoft Internet Explorer (IE) 5 XML Document Object Model (DOM) in your ASP applications.

Separating content from presentation

When developing a Web site, you ideally want to design your Web application in a modular manner, keeping your presentation components separate from both your content and your executable business logic code. That way, your developers can focus on writing clean, re-usable code that generates dynamic content based on your organization's business rules, without having to worry about the look and feel. Your designers can focus on creating great-looking Web templates without having to worry about integrating any application logic into their work. Your content authors can also focus on writing good content, without having to worry about the look and feel or application logic.

Unfortunately, this is difficult to do with HTML and existing server-side scripting technologies. With HTML, content is thoroughly intertwined with tagging that defines presentation, and client-side scripting that defines application logic. The use of server-side scripting technologies, including CGI and ASP, improves the situation somewhat by segregating content--which is usually stored in backend databases. Nevertheless, you must contend with the intermixing of application logic in the form of scripting language code, and presentation in the form of HTML mark-up.

This article will show you how you can use Microsoft's XML DOM implementation that ships with Internet Explorer 5 to alleviate this problem. Specifically, we'll walk through the construction of an FAQ (Frequently Asked Questions) ASP application that shows you how you can use emerging XML standards to separate content and presentation into discrete components. This application is shown in Figure A.

Figure A: This is the server-side XML FAQ ASP application.



You'll learn how to write an ASP script that uses the IE 5 XML DOM to merge these components on the server at runtime, and stream the output to the downstream browser as HTML. This allows you to use emerging XML technologies to help simplify site development and maintenance, while at the same time maintaining backward compatibility with older browsers. It also allows you to change your Web site's look and feel quickly and easily, without having to modify your content.

Developing the server-side XML application

To build our server-side XML FAQ application, we'll perform the following specific tasks:

- Author the FAQ content as an XML document.
- Write an XSL style sheet to define how a Web browser displays the XML document's elements and attributes. In a nutshell, the XSL style sheet is your XML document's presentation template.
- Write an ASP script to perform the XML to HTML transformation on the server. This way, you get all of the benefits of using XML to simplify Web site maintenance, while maintaining backward compatibility with older browsers. This ASP script uses IE 5's XML DOM implementation to parse both documents, perform the transformation, and stream the output to the browser as raw HTML.

In addition to the code listings provided with this article, you can also download this sample application from our Web site.

Step 1: Author your FAQ content as an XML document

Your first step is to author your content as an XML document. Before we get into that, let's first review some of the basics of XML. *XML* is best described as the universal format for data on the Web. XML allows you to create human-readable custom document formats to describe content and data in a manner that's independent from how the end user will view or consume the data. XML is a subset of SGML (Standard Generalized Markup Language), the same document meta-language from which HTML is

derived. Covering all of the details of XML is beyond the scope of this article. There are many excellent sources of XML information on the Web, including Microsoft's XML tutorial at www.msdn.microsoft.com/xml/tutorial/default.asp, and the actual W3C standards document at www.w3.org/TR/REC-xml. Our FAQ document is shown in Listing A. Just like any other XML document, FAQ.XML creates a tree hierarchy of elements and their sub-elements.

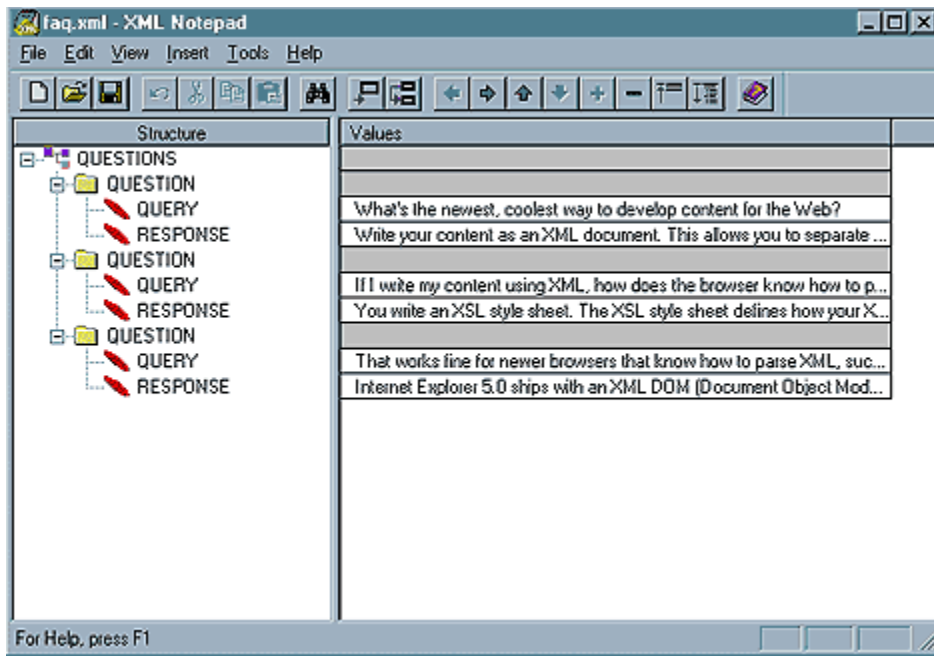
The root element of FAQ.XML is the QUESTIONS element. We can best describe this element as a collection of one or more QUESTION elements. Each QUESTION element, in turn, contains a QUERY element (the actual question) and a RESPONSE element (the answer).

Listing A: FAQ.XML code listing

```
<?xml version="1.0"?><QUESTIONS>
<QUESTION>
    <QUERY>Question goes here</QUERY>
    <RESPONSE>Answer goes here.
</RESPONSE>
</QUESTION>
<QUESTION>
<QUERY>Another question goes here.
</QUERY>
    <RESPONSE>The answer goes here.
</RESPONSE>
</QUESTION>
</QUESTIONS>
```

To edit 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. An example is Microsoft's XML Notepad, which you can download from www.msdn.microsoft.com/xml/notepad/intro.asp. XML Notepad displays the hierarchy of a valid or well-formed XML document using a collapsible outline control. Figure B shows our FAQ.XML document opened with Microsoft XML Notepad.

Figure B: This is FAQ.XML in Microsoft XML Notepad.



Step 2: Write an XSL style sheet to transform the XML document to HTML

After placing your content into an XML document, the next step is to define how that content will be displayed. To do this, we'll write an XSL (eXtensible Style Language) style sheet. You can view the code listing for this XSL style sheet in Listing B.

Listing B: FAQ.XSL code listing

```
<?xml version="1.0"?>
<html xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<head>
<title>Frequently Asked Questions</title>
</head>
<body bgcolor="#000080">
<div align="center">
<center>
<table border="0" width="580" cellspacing="0">
<tr>
<td width="580" valign="top" bgcolor="#800000">
<strong>
<h1 align="center">
<font face="Arial" color="#FFFFFF">
<a name="top">Frequently Asked Questions</a>
</font>
</h1>
</strong></td>
</tr>
<tr>
<td width="580"><table border="0" width="100%"
cellspacing="0" cellpadding="10">
<tr>
<td width="580" bgcolor="#FFFFFF">
<xsl:for-each select="QUESTIONS/QUESTION">
<b><font face="Arial" color="#000080">
<xsl:value-of select="QUERY"/>
```

```

</font></b>
<p><font face="Arial">
<xsl:value-of select="RESPONSE"/>
</font></p>
<h5><font face="Arial"><a href="#top">
    Back to Top</a></font></h5>
<hr/>
</xsl:for-each>
<h5 align="center"><font face="Arial">
    Copyright (c) 1999, Andy Hoskinson.
    All rights reserved.</font></h5>
</td>
</tr>
</table>
</td>
</tr>
</table>
</center></div>
</body>
</html>

```

An *XSL style sheet* is, in essence, a display template for an XML document. A typical XSL style sheet contains standard HTML markup to define the page's design, and intersperses the HTML with special XSL tags. These tags include for-each tags that enumerate through all of the elements in the XML document, and value-of tags that act as placeholders for content contained in the XML elements.

To give you a specific example, let's look at what happens when we merge our FAQ.XML document in Listing A with this XSL fragment using the IE 5 XML DOM (more on this later):

```

<xsl:for-each select="QUESTIONS/QUESTION">
<b><font face="Arial" color="#000080">
<xsl:value-of select="QUERY"/>
</font></b>
<p><font face="Arial">
<xsl:value-of select="RESPONSE"/>
</font></p>
<h5><font face="Arial"><a href="#top">
    Back to Top</a></font></h5>
<hr/>
</xsl:for-each>

```

The IE XML parser enumerates through all of the QUESTION elements in the QUESTIONS element (our document's root element). For each QUESTION element, the parser outputs the value of the QUERY and RESPONSE element. This process produces the following HTML output:

```

<b>
<font face="Arial" color="#000080">
Question goes here
</font></b>
<p><font face="Arial">
Answer goes here.</font></p>
<h5><font face="Arial"><a href="#top">
Back to Top</a></font></h5><hr/>
<b>
<font face="Arial" color="#000080">
Another question goes here.</font></b>

```

```

    <p><font face="Arial">
The answer goes here.
</font></p><h5><font face="Arial">
<a href="#top">Back to Top</a>
</font></h5><hr/>

```

One important caveat about authoring an XSL style sheet: an XSL style sheet is technically an XML document; therefore, it must be well formed. This includes closing every tag, encapsulating attribute values in quotes, and refraining from overlapping opening and closing tags. HTML, on the other hand, is much more forgiving. Therefore, you might need to modify your HTML a little.

To give you a specific example, in HTML, you can create a horizontal rule with the <HR> mark-up. A properly written XML parser will choke on this. Write it this way instead, <HR/>.

Step 3: Write an ASP script to perform the transformation on the server

The third and final step is to write your ASP script that uses the IE 5 XML DOM implementation to parse and merge the XML document and the XSL style sheet, and stream the HTML output back to the browser. We'll call this ASP script XML2HTML.ASP. The code listing for XML2HTML.ASP is in Listing C.

Listing C: XML2HTML.ASP code listing

```

<%@ Language=VBScript %>
<%option explicit
on error resume next
dim objXMLDoc
dim objXSLDoc
dim strXMLDoc
dim strXSLDoc
dim strResults
const PROG_ID = "Microsoft.XMLDOM"
'Get the XML file name from the Request object
strXMLDoc = server.mappath(request("xml"))
'Get the XSL file name from the Request object
strXSLDoc = server.mappath(request("xsl"))
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 style sheet
        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 style sheet
            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
'Put the transformed document into the
'response stream
Response.Write strResults
'Clean up
set objXSLDoc = nothing
set objXMLDoc = nothing
%>

```

The XML2HTML.ASP code is easy to follow. First, it gets the virtual path to the XML and XSL files from the ASP Request object, and converts them to a physical directory using the MapPath method of the ASP Server object. Next, it creates an instance of IE 5's XMLDOMDocument class using the Server object's CreateObject class factory, passing in Microsoft.XMLDOM as the Prog ID.

Third, it does a synchronous load of each document. This is notable because the default load type is asynchronous. To make the load process synchronous, you want to set the async property to false to prevent the Load method from returning before the document is completely parsed.

Once both documents are parsed successfully, the script calls the transformNode method of the XML document's XMLDOMDocument object, passing in the XSL style sheet's XMLDOMDocument instance as a parameter. This method will perform the actual merging of the XML document and the XSL style sheet, and will return a string that contains the operation's HTML output. As a final step, we'll place this in the HTTP response stream by calling the write method of the ASP Response object.

For error handling, we'll do two things. First, we'll query the ParseError property of the XMLDOMDocument object for both the XML document and the XSL style sheet. This will tell us if any errors occurred parsing either document. Second, we'll place On Error Resume Next at the top of our ASP script to force script execution to continue when an error is raised. To handle any errors that might occur, we'll query the err object at certain parts of the script to test for any error conditions that might exist.

Once you've written all of your code, upload all three files (FAQ.XML, FAQ.XSL, and XML2HTML.ASP) to your IIS 4 Web server, and open up the following URL with any Web browser:

```
http://your_host/dir/xml2html.asp?xml=faq.xml&xsl=faq.xsl
```

Please remember to install IE 5 on your Web server. Otherwise, this code won't work. If installing IE 5

on your server isn't an option, there's one more thing you can do. Microsoft and DataChannel have co-developed a Java implementation of the IE 5 XML DOM. You can download this package at www.datachannel.com/xml_resources/. After you download it, unzip it to your \winnt\java\trustlib directory, and follow the instructions in the readme.html file to register the key Java classes as COM classes. You can then use the ASP code in Listing C virtually without modification. The only thing you need to modify is the Prog ID, changing it from Microsoft.XMLDOM to *DCXML.Document.1*.

Conclusion

In this article, we've shown you how to create an ASP application that uses the IE 5 XML DOM on the server to generate HTML from XML content and XSL presentation rules at runtime. This allows you to separate content from presentation, and simplifies the task of developing and maintaining your Web site. In future articles, we'll investigate other useful ways to use Microsoft's XML technologies in your ASP applications.

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.