Jul 15, 1999

Using Microsoft FrontPage to Create ASP pages

By Kevin Spencer

ASP Tricks

enter the discussion

I develop ASP/ADO Internet Database applications for a living. I almost always use Microsoft FrontPage 98 to do this. Now, I've heard many people swear by Visual InterDev as the tool of choice for developing ASP/ADO applications, claiming that FrontPage messes up their code, but there are a number of excellent reasons why I prefer FrontPage:

Most of my clients are entrepreneurs, who had an idea for a web site and bought into the claims that FrontPage and the other WYSIWYG Web Site development tools enable you to create web sites without programming. Well, "without programming" may be true to a certain extent, but not *ASP* web sites, for sure!

You can use NotePad, another text editor or Visual InterDev to create ASP web pages that work fine, but the same pages will be mangled by FrontPage, or another WYSIWYG Editor, because these editors aren't capable of reading and understanding the ASP scripting in the pages. They think the HTML in the pages is wrong and so they helpfully "correct" it.

But, if you understand what FrontPage likes and doesn't like, and are clever about your coding, then you'll find all your problems disappear. FrontPage Editor *can* create ASP pages that can do literally anything that ASP pages created in NotePad, Visual InterDev, etc. can do.

So, what's the advantage? Well, I charge $100.00/hour for programming services, and my clients don't want to pay $100.00/hour for HTML layout. So, they either do their own layout, or hire an HTML designer to do their layout and then give it to me. I build the ASP code into the application, and give it back to them. After that, if they want to change the look or style of the application, they can open the pages up in FrontPage Editor, or another WYSIWYG Editor, and, without any knowledge of programming, change the HTML in the page easily and quickly.

For example...

Let's begin with a simple FrontPage quirk. This discussion will focus on FrontPage 98, which is much less "friendly" to HTML than FrontPage 2000, but is far more widely in use (largely because, as of writing this, FrontPage 2000 has not been commercially released).

ASPTODAY Diary

S	M	T	W	T	F	S
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
1	2	3	4	5	6	7
8	9	10	11	12	13	14

Links
Author Page
About ASPToday

Suppose you have a form on a page, and you want to conditionally set the form's `ACTION` property according to a variable being passed from the query string. You can write several form tags, and use the value of the variable to select which form tag you're going to put in the page. For the purpose of this discussion, I've indicated the HTML code through and square brackets and, where possible, through indentation - the said brackets and spaces are not part of the code!

```
<%
Dim myvar
myvar = Request.QueryString("myvar")%>

<%if myvar = "1" then%>
   [<form name="myform" ACTION="mypage1.asp" METHOD=POST>]
<%elseif myvar = "2" then%>
   [<form name="myform" ACTION="mypage2.asp" METHOD=POST>]
<%elseif myvar = "3" then%>
   [<form name="myform" ACTION="mypage3.asp" METHOD=POST>]
<%else%>
   [<form name="myform" ACTION="mypage4.asp" METHOD=POST>]
<%end if
%>

      'form HTML here

   [</form>]
```

Now, this is perfectly legal. Since the variable will only have one value, there will only be one form tag on the page when it is sent to the browser. The HTML will be correct. But FrontPage Editor can't interpret the ASP code, and it doesn't "know" that this is what is going to happen, so it will change the code to the following:

```
<%
Dim myvar
myvar = Request.QueryString("myvar")

if myvar = "1" then
%>

   [<form name="myform" ACTION="mypage1.asp" METHOD="POST">]
<%elseif myvar = "2" then%>
   [</form>]
```

```
   [<form name="myform" ACTION="mypage2.asp" METHOD="POST">]
<%elseif myvar = "3" then%>
   [</form>]

   [<form name="myform" ACTION="mypage3.asp" METHOD="POST">]
<%else%>
   [</form>]

   [<form name="myform" ACTION="mypage4.asp" METHOD="POST">]
<%end if%>

      'form HTML here...

   [</form>]
```

Whoa! What has happened? Well, FrontPage has added in a bunch of `</form>` tags, thereby creating four separate forms on the page. Now, this isn't what we intended, so let's just forget about using FrontPage in this instance, right? Oh no. Try this out instead:

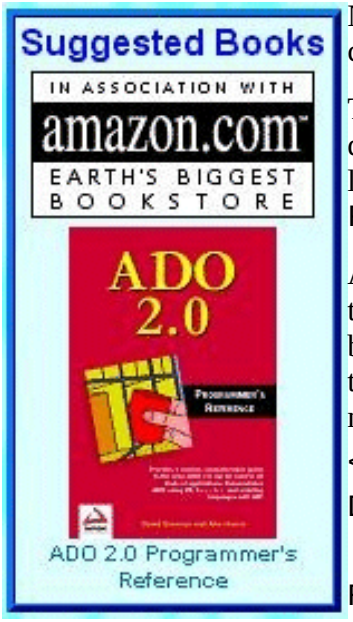
```
<%
Dim myvar, strAction

myvar = Request.QueryString("myvar")

if myvar = "1" then
strAction = "mypage1.asp"
elseif myvar = "2" then
strAction = "mypage2.asp"
elseif myvar = "3" then
   strAction = "mypage3.asp"
else
   strAction = "mypage4.asp"
end if
%>

   [<form name="myform" ACTION="]<%=strAction%>" [METHOD="POST"] >
```

```
    'form HTML here...

[</form>]
```

Instead of writing the entire form tag, we have written a bit of code which defines the value of a string variable, and then plugged the string into a single form definition. Voila! FrontPage is happy, I'm happy, and the client is happy.

This technique can be applied to any kind of object on the page. Instead of writing out a whole bunch of code that writes HTML to the page, just substitute the different bits with variables set in blocks of ASP code.

How FrontPage can speed up production



Now, here's something that makes FrontPage positively *useful* when you're developing ASP pages. Not so long ago, I wanted to construct an include file that would display a randomly selected book from the Amazon.com online book club - a bit like this:

This was going to involve creating a table of fixed width, and combining it with code that would select a random title from 15 different titles, display an Amazon Associates image, an image of the book title, and the title of the book, with a link to the book's location on their web site. I could then insert this small table into any page with an include tag, i.e. `<!— #INCLUDE FILE="books.inc">`

An included ASP file shouldn't have any extraneous `<html>`, `<head>`, `<body>` or `<title>` tags, and these are the sorts of tags which FrontPage Editor puts into all pages whether you want it to or not. So, I created a new page in FrontPage Editor, then built the ASP code, visually designed the table, tested it as an ASP page and saved it. Then I opened it in a text editor, removed all the extraneous tags that FrontPage Editor had added, and saved it as `books.inc`. Voila! Instant ASP/HTML! Took about 30 minutes, testing included. Here's the ASP, including all the HTML FrontPage generated for me, but without the extra tags:

```
<%
Dim str, title

Randomize

display = Int((15 * Rnd) + 1)    ' **Generate random value
                                 ' ** between 1 and 15.

Select Case display
Case 1 str = "0078825482"        ' ** The "str" variable is used in
                                 ' ** both the img src tag and the URL
```

```
        title = "Sql Server 7 Developer's Guide"
Case 2 str = "0471317012"
        title = "Building Distributed Applications With ADO"
Case 3 str = "0789716283"
        title = "Using Microsoft Sql Server 7.0"
Case 4 str = "1861001797"
        title = "Professional Asp Techniques for Webmasters"
Case 5 str = "1861001835"
        title = "ADO 2.0 Programmer's Reference"
Case 6 str = "1861001266"
        title = "Professional Active Server Pages 2.0"
Case 7 str = "0672312905"
        title = "Teach Yourself Microsoft Sql Server 7.0 in 21 Days"
Case 8 str = "1572315164"
        title = "Microsoft Odbc 3.0 Software Development Kit and Programmer's Reference"
Case 9 str = "1572319615"
        title = "Programming Distributed Applications With Com and Microsoft Visual Basic 6.0"
Case 10 str = "1861001460"
        title = "Professional MTS and MSMQ Programming with VB and ASP"
Case 11 str = "1575211394"
        title = "Teach Yourself Active Web Database Programming in 21 Days"
Case 12 str = "1572317000"
        title = "Programming Active Server Pages (Microsoft Programming Series)"
Case 13 str = "1575213516"
        title = "Active Server Pages Unleashed"
Case 14 str = "1575213494"
        title = "FrontPage 98 Unleashed"
Case 15 str = "1576101266"
        title = "Build Your Own FrontPage 97 Web Sites"
End Select
%>


      <!-- Here, a table is created in HTML -->
```

```
   [<div align="center"><center>]
    [<table border="3" cellpadding="2" bordercolorlight="#0080FF"]
    [bordercolordark="#000080"]
    [cellspacing="0" bgcolor="#E6FFFF" width="150">]
      [<tr>]
          [<td width="100%" align="center"><font face="Arial" size="3" color="#0000FF"><strong>Suggested]
    [Books</strong></font><div align="center"><center><table border="0">]
      [<tr>]
        [<td width="100%" align="center">]
        [<img src="../images/tight-black-amzn-associates.gif"]
        [alt="tight-black-amzn-associates.gif (1294 bytes)" WIDTH="127" HEIGHT="66"></td>]
      [</tr>]
      [<tr>]

<!-- Here, a hyperlink is dynamically created by inserting an ASP -- >
<!-- script into the hyperlink. The "img src" is generated -- >
<!-- in the same way -- >

        [<td width="100%" align="center"><a
href="http://www.amazon.com/exec/obidos/ISBN=<%=str%>/sitedesignbytakeA/"><img src="../images/] <%=str%>
[.m.gif" alt="] <%=title%>["
border="0"></a><br>]

<!-- Here again, a hyperlink is dynamically created by inserting -- >
<!-- an ASP script into the HTML -- >

   [<font face="Arial" size="1"><a    ]
[href="http://www.amazon.com/exec/obidos/ISBN=]<%=str%>[/sitedesignbytakeA/">]<%=title%>[</a></font></td>]
        [</tr>]
      [</table>]
      [</center></div></td>]
    [</tr>]
   [</table>]
```

```
[</center></div>]
```

As you can see, the HTML was rather lengthy. But using the WYSIWYG Editor, I was able to put it together in a manner of minutes. The advantages of using FrontPage to create the ASP/HTML combination are pretty significant. In the first place, the code was developed in a relatively short length of time, using FrontPage Editor, much less time than it would have taken to use a text editor. Time is money, and less time means more money! In the second, the code can be edited in FrontPage from this point forward. If you're working for a client who isn't knowledgeable about HTML, this is an attractive feature for the client. It's also a pretty attractive feature for the developer!

Related Articles on ASPToday

Using HomeSite 4.0 from Allaire Software for ASP Development
Implementing Multi-Page Web Forms in ASP

Related Links

Article: MSDN -- Introducing FrontPage 98
Article: MSDN -- Microsoft FrontPage 98 Server Extensions Resource Kit
Article: 15seconds -- Simple Web-Database System Application by Joung-ho Jay Jung (using FrontPage to connect to an Access Database)
Article: Swynk.com -- Running a Perfect Web Site 2nd Ed.
Article: 15second -- Microsoft Visual InterDev: The New Way To Web, by John Shewchuk and Gregory Leake
Article: Active Server Pages -- Creating a Web page with a Text Editor
New ASP HTML Editor: ASP Express
Book: Beginning Active Server Pages

Book: HTML 4 Programmers Reference
Book: Professional Active Server Pages
Book: Professional ASP Techniques For Webmasters

If you would like to contribute to ASPToday, then please get in touch with us by clicking here.

keyword search

Search

free email updates

email address

Register

ASPTODAY Diary

S	M	T	W	T	F	S
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
1	2	3	4	5	6	7
8	9	10	11	12	13	14

63 Matches sort current matches by Author Date Rating

21 responses (0-5): Overall (3.81) | User Level (3.05) | Useful (3.19)
October 11, 2000
Recursion in XSLT
Corey Haines
As people begin delving into using XSLT, more and more complicated transformations are needed. In our quest to convert data into another format, several inherent limitations in XSLT are being realized and apparent barriers reached. Things that are simple in procedural programming languages are difficult or non-existent in XSLT, requiring a shift in our standard programming paradigm when we move to a template-based language. In this article, Corey Haines explores the use of recursion, a powerful trick to use in XSLT, to solve several problems that might arise. Read on >>

43 responses (0-5): Overall (3.4) | User Level (2.74) | Useful (3.28)
October 5, 2000
Bind XML data to a Tree–Like Structure
Ricardas Kunevicus
If you are looking for a way to create a tree structure like Windows Explorer, and want to be able to persist the state of the structure even when you have browsed to a different site before returning, then read this article. Ricardas Kunevicus uses XML and some DHTML features to create his tree. Read on >>

61 responses (0-5): Overall (4.18) | User Level (2.89) | Useful (3.92)
September 28, 2000
SVG - A New Standard in Web Graphics
Jon Frost
In this article Jon Frost explores a new language that is about to revolutionize the Web nearly as much as HTML did in 1992. Scalable Vector Graphics (SVG) is a new open source text–based language for describing two–dimensional graphical objects in XML, and will herald a new Web era by eliminating the layers of production that content originators currently face. This article briefly explores how SVG emerged and quickly moves into analyzing some SVG code samples - future articles will cover a complete working mapping application. Read on >>

31 responses (0-5): Overall (3.74) | User Level (3.1) | Useful (3.77)
September 22, 2000
Creating Summary Reports in XML and XSLT
Corey Haines
Over the past couple of years, XML has become the latest and hottest buzzword in the industry. As we become more and more familiar and comfortable with using XML to handle our data storage or transfer needs, it is increasingly important that we become comfortable with XML's less-understood sister, XSLT. VBXML's Corey Haines, and co-author of Wrox's Professional ASP XML, develops a simple example of writing a summary report with XSLT. Along the way, we will discuss several XSLT techniques, including named templates, moded templates, parameters, calling templates and several functions. We will also look at some XPath functions along with making use of the XPath axis functionality. Read on >>

65 responses (0-5): Overall (3.95) | User Level (3.29) | Useful (3.52)
September 19, 2000

Creating an e-Book using XML, XSL and CSS
Mani Raja
e-Book publication is a growing business, and Mani Raja shares with us his vision of creating a virtual library resource on the Web. He will step you through the process of creating an e-book using XML, XSL and CSS. The download includes a fully working example of what you can produce yourself. Read on >>

33 responses (0-5): Overall (3.79) | User Level (3.06) | Useful (3.27)
August 31, 2000
Generating WBMP Images With ASP
Ewen Stewart
There is an old saying 'A picture is worth a thousand words', and if you have to fit that thousand words into a 1 inch display then a picture is without doubt the best way of doing so. Maps and graphs are great examples of how images are used to convey information that would be difficult to convey in words, so fire up your WAP emulator and let Ewen Stewart show you how to create wireless images for the new breed of graphics-enabled mobile phones. Read on >>

28 responses (0-5): Overall (3.75) | User Level (2.54) | Useful (3.39)
August 30, 2000
Data Interoperability
Stephen T. Mohr
Life would be easy if all data existed in one form and could be stored using one technology – but important data resides in a variety of forms, and our applications need to be able to access and manipulate it in useful and meaningful ways. With the trend toward networked applications, we have the burden of needing to be able to exchange data simply and without losing its integrity or utility. Steven T. Mohr shows us one method of dealing with this increasingly common problem. Read on >>

59 responses (0-5): Overall (3.85) | User Level (3.1) | Useful (3.76)
August 24, 2000
You [too] Can Develop a C++ ATL Component
James Brannan
ATL is used to produce small, fast, industrial strength components. It also calls methods very quickly and allows you a fine control over COM features. In this article, James Brannan explains how easy it is to create a useful ATL C++ component that translates ADO to XML. By including a parameter as a path to an XSL template, this component can return any type of output an XSL style sheet is capable of producing. Read on >>

25 responses (0-5): Overall (3.92) | User Level (3.04) | Useful (3.48)
August 21, 2000
Creating applications using SOAP and XMLHTTP Part 3
Craig Murphy
In this, the final part of Craig Murphy's SOAP/XMLHTTP series, we look at using XSLT (eXtensible Stylesheet Language for Transformation) to process SOAP requests. Whilst this might sound like an unorthodox approach, it's actually the mechanism at the heart of the BizTalk mapper – and it shows that XML is a very flexible and concise mechanism for Application–to–Application (A2A) and Business–to–Business (B2B) communication. Read on >>

97 responses (0-5): Overall (4.08) | User Level (2.92) | Useful (3.81)
August 18, 2000
Site News Application Using ASP and Server-side XML
Phil Sandler
Think of a website that doesn't update its content on a regular basis, and you have a site that will never make it into anyone's favorites folder, and will have a repeat visitor rate that hovers around zero. Phil Sandler shows you how to add an easily maintainable news facility to your site - and if your customers know they will find something different from their last visit, they are more likely to return. Read on >>

18 responses (0-5): Overall (3.11) | User Level (3.28) | Useful (3.11)

August 17, 2000
Using XML Persistence to Overcome RDS Security Issues
Michael Mullins
Much has been written in the past year about the security issues associated with enabling RDS (Remote Data Services) on your Microsoft IIS web server. As well as referring you to sources for further investigation of these security implications, Michael Mullins presents an alternative method for obtaining data from the web server and working with it on the client - by using XML recordset persistence.
Read on >>

66 responses (0-5): Overall (3.88) | User Level (2.18) | Useful (3.58)
August 11, 2000
Tic–Tac–Toe – An Introduction to WMLScript
Michael Jones
WML is the HTML of the wireless world. It brings web pages to mobile phones and other portable devices, but by itself WML does not provide for much interaction without a trip to the server - not an ideal situation with the low bandwidth of current mobile networks. However if you add WMLScript you can accomplish quite a bit. In this article Michael Jones develops a tic–tac–toe game using WML and WMLScript.
Read on >>

27 responses (0-5): Overall (2.63) | User Level (2.26) | Useful (1.93)
August 10, 2000
Comparison of VML and SGC
Bryan Ramussen
There have been two main graphics related advances in the history of the web. The first was the use of images, the second was dynamic html, and we are now approaching a third – a major part of which will be the introduction of vector graphics in a variety of formats. Bryan Ramussen demonstrates the use of two currently available vector formats, VML and SGC (Structured Graphic Control). Read on >>

160 responses (0-5): Overall (3.71) | User Level (2.79) | Useful (3.62)
August 3, 2000
Enhance Input Forms Using XML
Chris Knowles
Web forms can be a necessary evil. An integral component in many a web site, they are tedious and often time-consuming to code. In this article, Chris Knowles takes a look at HTML forms, identifing their shortcomings and discovering that by judicious use of XML, it is possible to build form–handlers of virtually unlimited sophistication. Read on >>

41 responses (0-5): Overall (4.17) | User Level (3.61) | Useful (4.27)
July 28, 2000
Creating Applications using SOAP and XMLHTTP (Part 2)
Craig Murphy
In my previous article, I discussed the anatomy of a SOAP message, and created an example application to request information from the server, in a read-only fashion. This article will deal with the slightly more complicated subject of sending updates back to the server. We will also see how SOAP allows us to handle errors – or faults in SOAP–speak. Read on >>

30 responses (0-5): Overall (3) | User Level (2.77) | Useful (2.87)
July 27, 2000
Code Layer Separation in IIS Webclass Applications using XML and XSL
Pieter Siegers
Web applications can quite easily become a 'spaghetti' of interlocking code tiers, making it difficult to maintain business or presentation layers. Here Pieter Siegers will show you how you can keep these layers apart from the outset, using XML Read on >>

63 responses (0-5): Overall (3.78) | User Level (3.02) | Useful (3.57)

July 24, 2000
Dynamic XML Transformation
Adam Griffin
Have you ever wanted to apply different XSL methods to a set of XML based upon user inputs? A static XSL file can transform your XML data but what if you want different transformations for different user inputs? In this article, I will cover two key points of interest to help achieve this aim. Read on >>

115 responses (0-5): Overall (3.86) | User Level (2.97) | Useful (3.8)
July 18, 2000
Creating applications using SOAP and XMLHTTP Part 1
Craig Murphy
Application–to–application communication, whether over the Internet or over an Intranet, has always kept developers awake at night. Most solutions tend to be platform specific that don't scale very well, and often involve many round-trip requests from the client to the server. Other solutions, such as those involving DCOM, CORBA, etc. encounter problems communicating through firewalls. The Simple Object Access Protocol (SOAP) alleviates many of these problems. Read on >>

242 responses (0-5): Overall (4.04) | User Level (2.03) | Useful (3.91)
July 17, 2000
WAP – An Introduction
Karli Watson
Wireless Application Protocol, or WAP, is a term that seems to be everywhere at the moment. When you've finished reading this article you will not only have enough basic knowledge to understand a lot of the technical WAP material out there, you will also have taken your first step towards being able to implement Internet services for mobile users. Read on >>

32 responses (0-5): Overall (3.47) | User Level (2.78) | Useful (3.59)
July 10, 2000
Introduction to HDML and ASP
Christina Biggs
This article is intended to aid an HDML developer in using ASP to produce dynamic HDML code. Although Europe and Japan have already moved to WML, the US and Canada holds a large percentage of wireless users. Their phones still mostly interpret HDML, Phone.com’s wireless phone browser (UP.Browser) which Phone.com promises to continue to support. Read on >>

56 responses (0-5): Overall (3.57) | User Level (2.71) | Useful (3.2)
July 6, 2000
Generating XML Based "Auto-Request Quotes" From the Internet
Syed Amir Raza
As the e-business continues to grow, the demands placed on the automotive retailer grows as well. The auto industry is moving from a dealership focus to an e–focus, therefore to provide a business–to–business service to the customer there is a need for open standards – and XML fits perfectly in that domain. This article deals with the scenario of creating a web application for an online Auto Store, allowing a user to configure a vehicle, and submit an XML document that complies with the ADF standard. ADF is an open XML standard of the auto industry to exchange vehicle request for quotes. Read on >>

31 responses (0-5): Overall (3.45) | User Level (2.42) | Useful (3)
June 26, 2000
Writing to the BizTalk Framework
Stephen T. Mohr
Last week, we looked at BizTalk.org as a B2B intermediary. I mentioned that they provided the BizTalk Framework for handling communications related issues. Today we’re going to roll up our sleeves and look at what an e-commerce document looks like under the BizTalk Framework. You’ll find that you can wrap your data in a very simple layer, or evolve your code to convey very sophisticated information. BizTalk Framework, now in version 1.0, is platform neutral, but the forthcoming BizTalk Server product is expected to support the Framework upon release. Read on >>

78 responses (0-5): Overall (2.96) | User Level (1.9) | Useful (2.82)
June 23, 2000
BizTalk, XML, & the Future of e-Commerce
Stephen T. Mohr
Much has been said about how XML stands to revolutionize business to business (B2B) e-commerce. The ability of a business to electronically connect to a supplier or customer and conduct routine business without manual intervention promises to cut costs and reduce time to market. In reality, practical problems remain. And since you live in the real world, you have a keen interest in the emergence of a variety of B2B intermediaries, third parties who look to facilitate XML-based e-commerce. In this, the first of a two-part series, we'll see why intermediaries are important and how they work. I'll focus on BizTalk, an initiative driven by Microsoft but not dependent on Microsoft software, but also mention some competing efforts and how they differ. Read on >>

180 responses (0-5): Overall (3.8) | User Level (2.71) | Useful (3.69)
June 16, 2000
B2B Communication using XML over http Part 1
Larry Clarkin
Larry Clarkin introduces Business to Business (B2B) transactions, and why we should take heed of their different requirements to Business to Consumer transactions. One model for implementing B2B transactions is XML over http, and he demonstrates a simple method of doing so. Read on >>

37 responses (0-5): Overall (3.65) | User Level (2.89) | Useful (3.54)
June 13, 2000
Microsoft's Next Generation XML Parser version 3.0
Stephen T. Mohr
Microsoft's Technology Preview brings XML support a long way toward full compliance with the latest developments in XML. Full XPath and XSLT support should be ready when MSXML 3.0 is fully released, and the comparatively new SAX2 API is now supported. Steven Mohr gives an overview of the new features available with MSXML 3.0 beta, and includes some sample code to try. Read on >>

31 responses (0-5): Overall (3.81) | User Level (2.74) | Useful (3.97)
June 7, 2000
Monitor Your Web Site Performance with VML - Part 2
Alex Homer
In the previous article, I demonstrated how the XMLHttpRequest object, which is part of the MSXML parser component, can be used to fetch a Web page or other file, and showed how to use this ability to automatically monitor a website's response. Here I expand on that example, showing you how to store the results and create a chart from the resulting database using VML (Vector Modelling Language). Read on >>

84 responses (0-5): Overall (4.05) | User Level (3.04) | Useful (4.24)
June 6, 2000
Monitor Your Web Site Performance with the XMLHTTP Component Part 1
Alex Homer
The MSXML component that Microsoft provides with IE5 and Windows 2000 exposes a useful object called XMLHttpRequest. This is used to send an HTTP request to a remote or local Web server, and collect the server's response. Although predominantly intended for use when fetching or posting XML documents, it can also be used to fetch other kinds of documents as well - for example in monitoring a number of websites to check their performance and ensure that they have not 'fallen over' while you weren't looking. Read on >>

101 responses (0-5): Overall (2.83) | User Level (2.43) | Useful (2.47)
June 2, 2000

A Technique for Porting Windows Applications to the Web
Himansu M Karundasa
A problem facing may companies today is their need to port their VB windows applications for use on the web. Himansu Karunadasa shows us a method of performing such a task, using ASP generated COM objects converted into XML. Read on >>

203 responses (0-5): Overall (3.29) | User Level (2.32) | Useful (3.17)
May 30, 2000
Writing a Search Engine using XML and ASP
Peter McMahon
Peter McMahon takes you through the process of creating a search engine using only XML ad ASP. The light-weight yet powerful sample application is fully customisable for your website. Read on >>

52 responses (0-5): Overall (3.15) | User Level (2.42) | Useful (3.1)
May 19, 2000
Reporting with ADO, XML, and XSL - Part 2
Anthony Young
In the first part of this two part series, Anthony Young demonstrated how to create reports with a generic XSL document and ADO 2.1. Here he extends the functionality of his code by adding client-side XML parsing to the server-side parsing previously demonstrated, and the ability for the user to change the sort order of the report. You'll agree that combining ASP, MS XML Parser, ADO, and VB creates a powerful XML-related toolset. Read on >>

119 responses (0-5): Overall (3.56) | User Level (3.02) | Useful (3.36)
May 18, 2000
XML Documents and Stored Procedures on SQL Server
Dejan Stojanovic
Dejan Stojanovic shows you how to build reusable XML data islands from existing data on Microsoft SQL Server using stored procedures. By combining two common techniques, he shows you how to better divide the database services tier and the business logic tier, and what gains can be achieved with such an approach. Furthermore, he explains how to exchange XML documents through SQL Server, using extended stored procedures and SQL Mail. Read on >>

165 responses (0-5): Overall (4.22) | User Level (3.25) | Useful (4.15)
April 25, 2000
Client/Server Side XML Hierarchical Menus
Dan Wahlin
Proper space utilization can be a difficult and daunting task to undertake when designing a website. This is especially true if the site contains a large number of links. In this article Dan Wahlin discusses the implementation of a menu system similar to the Start menu in Windows that lets users access sub-sites without clicking multiple times to get there. The menu system will be hierarchical in nature, rely on DHTML, and be based upon data contained in an XML document. Read on >>

58 responses (0-5): Overall (3.86) | User Level (3.17) | Useful (3.98)
March 22, 2000
SQL Server XML Data Access with SQLXML.DLL: the Polymorphic Spreadsheet - Part II
Michael Corning and Michael Rys
In this article, Michael Corning and Michael Rys explain how SQLXML might be used to generate XML output from SQL queries. Paying particular attention to using SQLXML's EXPLICIT mode, they show how certain limitations of SQLXML's AUTO mode might be overcome. Read on >>

55 responses (0-5): Overall (3.42) | User Level (2.31) | Useful (3.35)
March 1, 2000

Handy Tools for XML Schema Design and Conversion
Craig McQueen
There are many tools that make life easier for progammers. Craig McQueen shares his discovery of a great software development tool for designing and creating XML schemas called XML Authority from Extensibility, and how it can be used with BizTalk's JumpStart Kit to convert XML schemas into COM components. Read on >>

160 responses (0-5): Overall (3.52) | User Level (3.07) | Useful (3.51)
February 29, 2000
Reporting with ADO, XML and XSL - Part I
Anthony Young
Anthony Young shows how to save, or persist, a recordset into an XML file; create a XSL file that will later be used to transform the XML into HTML; and transform the XML into HTML. Read on >>

55 responses (0-5): Overall (3) | User Level (2.13) | Useful (2.93)
February 7, 2000
The BizTalk Initiative
Alex Homer
Alex Homer considers BizTalk, paying attention to its schema library, server and tools. Read on >>

120 responses (0-5): Overall (3.41) | User Level (2.88) | Useful (3.31)
February 3, 2000
Using the BizTalk JumpStart Kit to create a platform-independent e-commerce site
Fabio Claudio Ferracchiati
Fabio Ferracchiati analyzes the new BizTalk JumpStart Kit, based on the BizTalk XML Framework, which allows the creation of easy application integration and platform abstraction programs. Read on >>

199 responses (0-5): Overall (3.36) | User Level (2.59) | Useful (3.28)
January 5, 2000
Applying XML and ASP to Create a Web Application Framework
Michael Awai
Michael Awai demonstrates how to apply XML, XSL, and ASP to develop flexible, portable frameworks for building Web applications, using a brief case study. Read on >>

115 responses (0-5): Overall (3.75) | User Level (2.87) | Useful (3.79)
December 31, 1999
Web Queries Using VB, Excel, XML, and ASP - Part II
Dan Wahlin
In the second and concluding part of his article, Dan Wahlin focuses on retrieving data from a website and transferring it from Excel and into an XML file that can used to display it on your own web page
Read on >>

146 responses (0-5): Overall (3.69) | User Level (3.05) | Useful (3.79)
December 29, 1999
Web Queries Using VB, Excel, XML, and ASP - Part I
Dan Wahlin
In this first of a two-part series, Dan Wahlin shows us how to capture web information using Visual Basic and Excel and then XML/XSL in combination with ASP to display the captured information in a desirable manner Read on >>

69 responses (0-5): Overall (3.35) | User Level (2.94) | Useful (3.23)
December 28, 1999

Using XML to Dynamically Create IIS Application Objects
Jason Bock
Jason Bock uses an XML file alongside a COM interface to dynamically add pages to his web site without having to recompile the gateway component. Read on >>

111 responses (0-5): Overall (2.87) | User Level (2.39) | Useful (2.86)
December 10, 1999
Inspired XML - Using an XML Glossary to Explain Terminology
Paul Spencer
XML and ASP produce powerful web applications that operate equally well with any web browser, and if you add XSL you're really adding to the power and speed of development. Paul Spencer shows one such example, in a slightly unusual fashion. Read on >>

156 responses (0-5): Overall (3.76) | User Level (3.08) | Useful (3.67)
November 25, 1999
SQL Server XML Data Access with SQLXML.DLL: the Polymorphic Spreadsheet - Part I
Michael Corning
Michael Corning shares his experiences of using XML with the technology preview of SQL Server 7.5 - SQLXML - demonstrating a 'polymorphic spreadsheet' that uses the new technology. Read on >>

35 responses (0-5): Overall (3.71) | User Level (3.17) | Useful (3.69)
September 10, 1999
Getting Configuration Information from an IE5 Client
Alex Homer
In Monday's article Alex Homer determined and catered for the ASP version installed on the client's machine. Here he extends the technique to exploit IE5's new default behaviors and determines the system, set up and installable components for use throughout the session Read on >>

86 responses (0-5): Overall (3.74) | User Level (3.14) | Useful (3.53)
August 24, 1999
Extending Your Web Application’s Interface using XML, XSL, Visual Basic, and ASP
James Britt
A great deal of db access occurs on the Web. James Britt shows how to develop a data management application that presents a flexible user interface, whilst still allowing external applications to "talk" with it. Read on >>

27 responses (0-5): Overall (3.56) | User Level (2.63) | Useful (3.41)
August 17, 1999
Batch Processing XML to HTML
Paul Spencer
If your ISP doesn't support XML, or you've a large website needing many pages updated with little hassle, then Paul Spencer's advice on the implementation of stylesheets may just help. Read on >>

58 responses (0-5): Overall (3.48) | User Level (2.86) | Useful (3.4)
August 11, 1999
Scriptlets - A better way to componentize
Sanjeev Yadav
Sanjeev Yadav shows you how to instantiatize a scriptlet that will increase the resources available for each client on the server. Thus a brief introduction to component pooling in MTS is shown. Read on >>

76 responses (0-5): Overall (3.14) | User Level (2.47) | Useful (3.07)
August 10, 1999

Using XML and ASP to separate Web Site Content from Design
Paul Spencer
Paul Spencer guides us through his two-page application, exploring the interrelationship between XML, XSL, CSS and HTML, explaining the technologies and where each is best suited. Read on >>

12 responses (0-5): Overall (3) | User Level (2.25) | Useful (3.5)
July 9, 1999
Site Navigation and Maintenance Made Easy with XML - Part 2 of 2
Paul Spencer
In Part 1 Paul Spencer separated style and content on his web site using XML and XSL, simplifying site maintenance and collaborative work. Here, in Part 2, he uses asp pre-processing to create a frame that adapts to browser capability. Read on >>

60 responses (0-5): Overall (3.55) | User Level (2.45) | Useful (3.48)
June 24, 1999
Easy -To- Maintain Navigation with XML
Paul Spencer
Combining XML and ASP makes for easy web site maintenance and design changes, by separating style from content. In the first of two articles, Paul Spencer introduces XML and looks at its uses for ASP supporting browsers. Read on >>

21 responses (0-5): Overall (3.81) | User Level (2.86) | Useful (4.05)
June 17, 1999
Fifth Generation Server-Side Scripting - Part III: XSL
Michael Corning
Michael Corning continues with the final part in his series on XML, and investigates the controversial XSL Read on >>

13 responses (0-5): Overall (3.54) | User Level (2.92) | Useful (3.31)
June 9, 1999
Fifth Generation Server Side Scripting Part 2
Michael Corning
Michael Corning continues his look at scripting server-side XML. This week he concentrates on marshalling XML data, managing the container database and trapping errors on the fly. Read on >>

31 responses (0-5): Overall (3.87) | User Level (2.84) | Useful (4.06)
June 2, 1999
Fifth Generation Serve Side Scripting - Part I
Michael Corning
Michael Corning introduces the new series on scripting server-side XML applications, taking as his basis an XML spreadsheet app wrapping data from the SQL Pubs directory into XML. Read on >>

10 responses (0-5): Overall (3.8) | User Level (3.6) | Useful (3.7)
May 18, 1999
IE5 Behaviors 2 : In Practice
Alex Homer
Following on from last weeks introduction to behaviors we take a closer look at building custom behaviors and the default behaviors included with IE5. Read on >>

16 responses (0-5): Overall (3.12) | User Level (2.75) | Useful (3.19)
May 3, 1999

XML Scriptlets 2/2 : In Action
Dino Esposito
Dino Esposito demonstrates how simple it is to work with Scriptlets by building an IE4/IE5 compatible scriptlet that gets and displays an ADO recordset in an HTML Table. Read on >>

203 responses (0-5): Overall (3.84) | User Level (1.37) | Useful (3.76)
May 1, 1999
An Introduction to XML Document Structure
Alex Homer
Following on from our look at the XML technologies, we look at how to build a well-formed and valid XML document, considering the various XML elements and rules of construction. Read on >>

50 responses (0-5): Overall (3.3) | User Level (1.6) | Useful (2.98)
April 30, 1999
The World Of XML
Alex Homer
As the XML Bandwagon starts to gain pace, we take a two day look through the World of XML and its associate technologies - XSL, XQL, XLink and XPointer. Read on >>

36 responses (0-5): Overall (3.25) | User Level (2.92) | Useful (3.06)
April 26, 1999
XML Scriptlets 1/2 : How they Work
Dino Esposito
Having previously introduced us to the various scripting technologies available, Dino Esposito takes a more detailed look at XML Scriptlets - script based pseudo-COM objects. This week, he introduces their basic syntax. Read on >>

96 responses (0-5): Overall (3.65) | User Level (2.12) | Useful (3.52)
April 16, 1999
Server Side XML in ASP
Adam Cartwright
With the release of Internet Explorer 5.0, it is much easier to use XML in web applications. Adam Cartwright describes how to harness the power of the IE 5.0 XML Document Object Model on the server with ASP. Read on >>

28 responses (0-5): Overall (2.5) | User Level (1.61) | Useful (2.5)
April 8, 1999
The ABC's of Scripting Technology
Dino Esposito
In our latest beginner's column, Dino Esposito walks us through the various terms and technologies that have recently entered the web scripter's armoury. Specifically, he introduces, DHTML, DHTML Scriptlets, the DOM, XML, XML Scriptlets, IE5 Behaviors and HTML Components. Read on >>

56 responses (0-5): Overall (3.84) | User Level (2.45) | Useful (3.8)
April 1, 1999
XML Hierarchical Data Binding
David Sussman
It's common knowledge that you can bind an XML data source to an HTML table in IE5, but you may not know that you can bind hierarchical XML data into nested tables as well. David Sussman shows you how. Read on >>

93 responses (0-5): Overall (3.73) | User Level (2.7) | Useful (3.7)
March 18, 1999

Creating XML Recordsets
David Sussman
To coincide with the release of IE5.0, David Sussman looks at one noticeable flaw in all of its beta releases - the inability to write database fields into an XML-delimited Recordset - and produces a solution.
Read on >>

87 responses (0-5): Overall (3.38) | User Level (2) | Useful (3.43)
February 23, 1999
Expose Your Site In XML
Alex Homer
In our first look at XML on the web, Alex Homer demonstrates how we can use Document Type Definitions to expose our own data as XML and display it in IE5. Read on >>

Oct 11, 2000

Recursion in XSLT

By Corey Haines

XML

enter the discussion

XML is taking the industry by storm. Not just the latest buzzword anymore, it has truly become a tool for working with data, delivering on the promise of display-independence, abstracting out the essentials of the data and serving it up for use in any sort of interface we want. The possibility of true UI-independence is a great one, bringing with it enormous ramifications.

Confusing, Confusing

The problem, though, is how to get the data, which you have so pains-takingly created and converted into a display format that others can view and use, to truly take advantage of your valuable information. This is where the eXtensible Stylesheet Language (XSL) comes into play. The original intention was to create a language that would specify how a set of data was displayed, but it soon became obvious that there was much more available here. When most people think of display and user-interfaces, they think of things like web browsers, wireless browsers (supporting WAP), physical print-outs and the like. With the advent of business-to-business (B2B), this definition was necessarily expanded to include the reading and understanding of data by automated systems.

Thus, the "user" in "user-interface" meant much more than a human. Page breaks, bolded and italicized text became less important than simply presenting the data in a standard format. To accomplish this, XSL began to be thought of as a transformation language. As time went on, the W3C realized that a specific transformation language was needed, and the eXtensible Stylesheet Language was expanded to include what is now known as XSLT, or the eXtensible Stylesheet Language – Transformation (the W3C specification for XSLT can be found here. Now, rather than including support for XSL's data display tags (called XSL formatting objects) that can be used for page breaks, line feeds and so on, vendors are adding support for XSLT, which is then used to transform an associated XML document into a format that the program can understand. For example, Microsoft's Internet Explorer web browser has support for XSLT that can be applied to an XML document to convert into HTML (conformance listing can be found on VBXML's MSXML XSLT and XPath conformance pages).

As people began delving into using XSLT, more and more complicated transformations were needed. In our quest to convert our data into another format, several inherent limitations in XSLT were realized and apparent barriers were reached. Things that are simple in procedural programming languages are difficult or non-existent in XSLT. The simple act of looping from 1 to 10 requires a shift in our standard programming paradigm when we move to a template-based language, such as XSLT.

Over the course of coming articles, we will be investigating, presenting and explaining several useful concepts and methods in XSLT for creating complicated transformations, creating some reusable techniques for adding additional functionality that XSLT doesn't inherently provide us with. Throughout this series, we will emphasize generalization of techniques to create reusable templates, so problem projects will be simpler, faster and easier to solve. Not all of these methods will be the most efficient, perhaps even seeming to be rather over-complicated, but the intent is to give you examples of different methods, some of which will hopefully spark some ideas in your head on how to solve other similar problems that can't be easily done with the simpler methods.

In this first article, we will solve a very simple problem: computing the maximum and minimum values of an attribute. The first method will use recursion (giving us a good chance to explain recursion in XSLT), the second method will be incredibly simple and probably make us wonder why we bothered to use the recursion method in the first place. That is, until we find ourselves using recursion in the solution of another problem and find that we have a better understanding of it.

Running a Transformation

One of the most common questions that arises when someone begins working with XSLT isn't one of how to write a specific transformation, but, rather, the seemingly simplistic question of "how the heck do I run a transformation in the first place". There are many great editors and tools out there that can run transformations for you, but the simplest and easiest transformation tool (and one that even provides you with fairly useful debugging information) probably is present already on your computer (and you may even be using it now): Microsoft's Internet Explorer.

In 1998, Microsoft added XML and XSL support to their popular browser software, including with the installation a DLL named MSXML. Wrapped in this DLL was a COM component that parses and presents XML in a tree format, called the Document Object Model, or simply, the DOM. It also supported what was then a not yet standardized dialect of XSL. Not long after the W3C finally created a standard for XSLT in 1999, Microsoft released a "technology preview" of its MSXML component (called, appropriately enough, MSXML2) that supported a subset of the W3C specification. This was in January of 2000. Since then several more upgrades have been offered up for public consumption: one in March, one in May and the latest in July. Each of these was more and more increasingly compliant with the W3C recommendation.

The latest version has support for the majority of the features and tags, and a large part of the day-to-day work that is done with XML and XSLT is supported by MSXML. Sadly, because the term "technology preview" is really just a fancy way of saying "beta", the new and XSLT-compliant version of MSXML was not included in either Internet Explorer 5 or Internet Explorer 5.5. On the other hand, Microsoft released an XML SDK (available at http://msdn.microsoft.com/xml/general/msxmlprev.asp) that included a small program (`xmlinst.exe`) that alters the registry in such a way that Internet Explorer can be coerced to use the new MSXML by default for doing its transformation.

For Internet Explorer to run an XSLT transformation over an XML document, it must be told which XSLT document to use and where it is located. This is done through the stylesheet attribute of the root node of your XML document. When this is done, you simply open the XML document in Internet Explorer, and the transformation is done automatically. Handily, Microsoft included a default XSLT transformation for viewing XML documents when no stylesheet is associated with it that presents your data in an expandable tree view. This is what happens when you open an XML document that has no stylesheet attribute.

Note: All of the code blocks in this article have an associated filename, which is where you can find the code in this article's download.

On to Our Technique!

Suppose you have a set of values in your document, perhaps the years that your loyal employees have worked at your company. So, you have created a bonus system, every year giving an exquisite authentic artificial-gold-plated lapel pin to the employee who has been at the firm the longest.

Naturally, you keep employee information in an XML format that looks something like the following:

employees.xml

```
<employees>
  <employee name='Corey Haines' yrs='1' />
  <employee name='John Slater' yrs='3' />
  <employee name='Dinar Dalvi' yrs='2' />
</employees>
```

The task at hand is to find the employee with the maximum value for the `yrs` attribute.

We will present three different methods for doing this, two fairly complicated and one incredibly simple.

First Method (Recursion)

Recursion is a word that can sometimes strike fear into the hearts of programmers, bringing up terms like "stack-overflow", "terminating case" and "endless looping". Luckily, the concept is simple and with a little concentration and attention to detail, easily enacted safely. Since XSLT is, by nature, a recursive language, we can hope that our processor is optimized to handle the inherent issues intelligently. Even so, we should watch out for filling up our memory by creating a recursion that is too deep.

To briefly explain the concept of recursion, think of the mathematical concept of a factorial (represented by an exclamation (!) symbol). Very simply, `n! = n x (n-1) x (n-2) x … x 3 x 2 x 1`. So, 3! = 3 x 2 x 1 = 6, 7! = 7 x 6 x 5 x 4 x 3 x 2 x 1 = 5040. To compute the factorial of a number, you can take notice of the fact that n! = n x (n-1)!. If we have a function F(n) which computes the factorial, you can write this as F(n) = n x F(n-1). So, we simply need to call F() again with the number minus 1 and multiply the result by n. As is readily apparent, we need a way to stop this cycle, what we call the "terminating case". In the sense of a factorial, this is when n = 1, since we know that F(1) = 1. This needs to be coded into our function to alleviate the problem of creating an infinite recursion. Here is the VBScript code for writing the factorial function:

f1.vbs

```
Function F(n)
   dim lRetVal
   if n <= 1 then
      lRetVal = 1
   else
      lRetVal = n * F(n-1)
   end if
   F = lRetVal
End Function
```

Now, recursion in XSLT is the same, we simply call the template again. Here is the XSLT code for the same function:

f1.xsl

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template name='F'>
<xsl:param name='n' select='1' />
   <xsl:variable name='retval'>
      <xsl:choose>
      <xsl:when test='number($n) &lt; 1'>1</xsl:when>
      <xsl:otherwise>
         <xsl:variable name='lessone'>
            <xsl:call-template name='F'>
            <xsl:with-param name='n'>
               <xsl:value-of select='number($n) – 1' />
            </xsl:with-param>
            </xsl:call-template>
         </xsl:variable>
         <xsl:value-of select='number($n) * number($lessone)' />
      </xsl:otherwise>
      </xsl:choose>
   </xsl:variable>
   <xsl:value-of select='$retval' />
</xsl:template>
```

Admittedly, the XSLT version is a bit more complicated and needs a bit of explanation. The first line of the template simply defines what parameters can be passed on. If nothing is passed, then a default value of `1` is set.

We then set up a variable, `retval`, which will hold the result of our calculation. Admittedly, we don't need to set a variable, we could simply rely on the fact that any output of the template will get sent back to the calling template, but it is a good programming habit to have your return value set at the end of your function, so it is easier to debug if you are getting some erroneous results.

The value of `retval` is then computed using an `<xsl:choose>` (the equivalent of a switch statement in other languages). The first `<xsl:when>` tests for when our parameter is the terminating case, namely the number 1 (or less than 1). In this case, `retval` is set to `1`. Otherwise (`<xsl:otherwise>`) we call our template again, passing in the initial value minus 1. The result of this is then multiplied by `n` and returned to the variable.

Our last step is to "return" the value of `retval` to our calling template, which the `<xsl:value-of>` tag does.

Note: We are using the XPath function, `number()`, which explicitly takes a value and casts it into a number. This is to ease the burden of the processor in doing automatic type-casting.

Now, back to our problem of finding the maximum value of an attribute. In VBScript, finding the maximum value of an array is simple:

max1.vbs

```
Function Max1(Vals())
   Dim lIndex
   Dim lCurMax

   LCurMax = Vals(Lbound(Vals))
   For lIndex = Lbound(Vals)+1 To Ubound(Vals)
```

```
      If lCurMax < Vals(lIndex) then
         LCurMax = Vals(lIndex)
      End if
   Next
   Max = lCurMax
End Function
```

Sadly, XSLT does not provide us with a mechanism for looping from `m` to `n`, but if we alter our view of the loop, we can solve this problem fairly simply using recursion.

Looping from `m` to `n` is simply taking the lower-bound and comparing it to the upper-bound. If they are not the same, then you have more than one element. Rerun the function on all the elements starting at the lower-bound + 1. Continue doing this until the lower-bound is equal to the upper-bound. This is a little confusing in words, but the code, itself, is fairly self-explanatory:

max2.vbs

```
Function Max2(Vals(), lLowBnd)
   Dim lIndex
   Dim lCurMax
   Dim lNewVal

   lCurMax = Vals(lLowBnd)
   If lLowBnd < ubound(Vals) then
      LNewVal = Max2(Vals, lLowBnd + 1)
   Else
      LNewVal = ubound(Vals)
   End if

   If lNewVal > lCurMax then
      LCurMax = lNewVal
   End if

   Max2 = lCurMax
End Function
```

Basically, what we are doing is capturing the value of the lowest entry in the array. We consider this to be our maximum value. We then call a function that will (presumably) return the maximum value of the rest of the array. These two values are compared, and the max is returned. This process is repeated, until we reach the ubound of our array (our terminating case). When this condition is true, we simply return that value.

Now, since we have mapped our function into a recursive one, the conversion to an XSLT template is a fairly easy:

findmax1.xsl

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template name="findmax">
   <xsl:param name="list" />
   <xsl:variable name='retval'>
   <xsl:choose>
   <xsl:when test="count($list) &gt; 0">
      <xsl:variable name="curval">
         <xsl:value-of select="$list[1]/@yrs" />
      </xsl:variable>
      <xsl:variable name='newval'>
         <xsl:call-template name="findmax">
         <xsl:with-param name="list" select="$list[position() &gt; 1]" />
         <xsl:with-param name="tocomp" select="$tocomp" />
         </xsl:call-template>
      </xsl:variable>
      <xsl:choose>
      <xsl:when test="string(number($newval)) = 'NaN'">
         <xsl:value-of select="$curval" />
      </xsl:when>
      <xsl:when test="number($curval) &gt; number($newval)">
```

```
         <xsl:value-of select="$curval" />
      </xsl:when>
      <xsl:otherwise>
         <xsl:value-of select="$newval" />
      </xsl:otherwise>
      </xsl:choose>
   </xsl:when>
   <xsl:otherwise>NaN</xsl:otherwise>
   </xsl:choose>
   </xsl:variable>
   <xsl:value-of select="$retval" />
</xsl:template>
```

In this template, our terminating case is when the passed-in node-set (our array) has no elements. If this is true, then the value `NaN` is passed back. Otherwise, we grab the value of the first element and then calculate the max value of the rest of the list by making a recursive call to the same template. The final test starts with the terminating case (when we receive `NaN` back from our template). This is done by checking for the value, itself with the code `string(number($newval)) = 'NaN'`. In this case, we grab the current value, that is, the value of the first element. The next two comparisons are there to decide whether our current value is the max, or if the recursively computed value is. Whichever one is the max is returned to the calling template. Compare the XSL to the VBScript above it, and it will become clear.

Now, the big problem with this template is that it isn't very general. This only compares the `yrs` attribute of the node; what would be really useful is a generic template that could compare any attribute. Luckily, we can do this; just add a parameter that contains the name of the attribute to compare. Here is the slightly revised code that is a completely generic version of the template:

findmax2.xsl

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template name="findmax">
<xsl:param name="list" />
  <xsl:param name="tocomp" />
  <xsl:variable name='retval'>
  <xsl:choose>
  <xsl:when test="count($list) &gt; 0">
     <xsl:variable name="curval">
        <xsl:value-of select="$list[1]/@*[name()=$tocomp]" />
     </xsl:variable>
     <xsl:variable name='newval'>
        <xsl:call-template name="findmax">
        <xsl:with-param name="list" select="$list[position() &gt; 1]" />
        <xsl:with-param name='tocomp' select='$tocomp' />
        </xsl:call-template>
     </xsl:variable>
     <xsl:choose>
     <xsl:when test="string(number($newval)) = 'NaN'">
        <xsl:value-of select="$curval" />
     </xsl:when>
     <xsl:when test="number($curval) &gt; number($newval)">
        <xsl:value-of select="$curval" />
     </xsl:when>
     <xsl:otherwise>
        <xsl:value-of select="$newval" />
     </xsl:otherwise>
     </xsl:choose>
  </xsl:when>
  <xsl:otherwise>NaN</xsl:otherwise>
  </xsl:choose>
  </xsl:variable>
  <xsl:value-of select="$retval" />
</xsl:template>
</xsl:stylesheet>
```

Finding the minimum value of an attribute is simply a matter of changing a single line; since we are looking for something that is the least of all, then we can replace `<xsl:when test="$curval > &newval">` with `<xsl:when test="$curval <`

&newval">.

(the `findmin` templates are also included in the code download that accompanies this article, in `findmin1.xsl` and `findmin2.xsl`)

So, whenever we need to find the maximum value, we can simply call this template, passing in a list of nodes. Remember our example of employee appreciation? Here is the call for this (formatted for XHTML display):

employeemax.xsl

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
   <html><head />
   <body>
   <p>The maximum number of years an employee has stayed at the firm is:
   <xsl:call-template name='findmax'>
   <xsl:with-param name='list' select='employees/employee' />
   <xsl:with-param name='tocompare' select='"yrs"' />
   <xsl:with-param name='curval' select='"0"' />
   </xsl:call-template>
   </p>
   </body>
   </html>
</xsl:template>
```

Second Method (Non-Recursion)

As promised in the beginning, we will present a much simpler, non-recursive way to solve this particular problem.

findmax3.xsl

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template name="findmax3">
<xsl:param name="list" />
<xsl:param name="tocomp" />
   <xsl:variable name='sortedlist'>
   <xsl:for-each select="$list">
      <xsl:sort select="@*[name()=$tocomp]" data-type="number" />
      <xsl:value-of select="@*[name()=$tocomp]" /><xsl:text>|</xsl:text>
   </xsl:for-each>
   </xsl:variable>
   <xsl:value-of select='substring-before($sortedlist, "|")' />
</xsl:template>

This template first uses XSLT's support for <xsl:sort> to create a pipe-delimited
list of values in sorted order. We then simply need to grab the value before the
first pipe and pass it back to the calling template. Now, this is simpler from an
understanding point of view, but you must be aware that this is going through the
processing of sorting the list. This must be weighed against the possible performance
issues when you have large lists.
```

Conclusion

In this article, we explored the use of recursion to solve several problems that might arise in XSLT. Finally, by inspecting our algorithm for one of the recursive problems, we found a much simpler way to solve it. This is, of course, a common occurrence when programming, and a technique that should always be explored. Too often, we find ourselves taking the first solution, no matter how inefficient it might be.

Recursion is a powerful trick to use in XSLT, and one which deserves to be inspected much more in-depth. The intent of this article was to show you examples that will spur you on to use this technique in your own programming when you need to do loops.

Click here to download this article's support material.

RATE THIS ARTICLE

Overall
Poor Excellent
User Level
Beginner Expert
Useful
No! Very
Submit

enter the discussion

Related Links

MSXML XPath Conformance
XML/XSL Portal
MSXML Technology Preview and SDK
MSXML XSLT Conformance
W3C XSLT Specification
Jeni's XML Pages

---

keyword search

Search

Home | Today's Article | Search | Feedback | Write For Us | Suggest an Article | Advertise

ADSI/CDO (13)
ASP Tricks (93)
ASP+ (12)
BackOffice (32)
Components (74)
Data Access (126)
Miscellaneous (35)
Non-MS ASP (10)
Scripting (82)
Security/Admin (44)
Site Design (42)
Site server (13)
XML (63)

free email updates

email address

Register

ASPTODAY Diary

S	M	T	W	T	F	S
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
1	2	3	4	5	6	7
8	9	10	11	12	13	14

Links
Author Page
About ASPToday

Oct 05, 2000

Bind XML data to a Tree–Like Structure

By Ricardas Kunevicus

ASP Tricks

XML

enter the discussion

Dynamic HTML and XML allow us to do many things on the Web not possible before. One of them is the data binding to various HTML controls. In this article we will build a tree–like structure using XML data and some DHTML features. We will see show how to bind XML data to the HTML controls and change visibility of some data areas, forming a tree–like structure (like Windows Explorer) with expanding/collapsing branches. We will save/restore the current view of the tree on leaving the page and restore it on opening the page again.

The demo consists of two HTML pages: one with XML data bindings and scripts and another just for navigation to demonstrate saving /restoring the view. They work with MS IE5 and above. To run the sample on other browsers you need to change the XML data island (supported be IE5 only) to the different data source object (DSO).

Using XML in HTML Documents

One of the main requirements of the Web is to take data from the database and display it in a web page. All techniques have one thing in common. They retrieve the data from the database and manipulate it on a server to produce an HTML page. It works fine with a few records, but if you want to change something in a big record set, you need to ask a server to rebuild a page and load it again. With Internet Explorer 4 Microsoft introduced a set of **remote data access techniques**. These allowed downloading a recordset to the client, manipulating it locally and sending changes back to the server. These techniques used specific formats for data (delimited text, SQL queries). The necessity of transmitting data in a standard formats involved XML formatted data into the remote data access area.

There are several ways to get XML data on the client. Microsoft has made a Data Source Object in Java that can read XML files. The XML DSO works with IE4 and IE5. Another way is suitable for IE5 only, getting XML data as a part of HTML document. This technique is known as an **XML Data Island**. It allows us to add an entire XML document into HTML surrounding it with `<XML></XML>` tags.

```
<XML ID="Vendors">
<?xml version="1.0" ?>
   <Vendors>
      <Vendor>
         <VendorID>APC</VendorID>
         <VendorName>APC (UPS)</VendorName>
         <PG>
...
   </Vendors>
</XML>
```

It's also possible to link to the XML file instead of typing it directly. One just needs to enter a file name as an SRC attribute of the XML tag.

```
<XML ID="Vendors" SRC="dataforcatalog.xml"></XML>
```

Both techniques are useful. Using inline XML makes it easier to form an ASP file but external files allow you to modify XML data without changing the HTML file. I have included samples of using both techniques.

- Catalog.html – with inline XML;
- Catalog1.html – with linked XML file.

Please note that IE 5.1 has problems maintaining and binding big inline XML data islands over the web. As far as I know this issue isn't fixed

yet. Look to the Microsoft Knowledge Base Article Q257723.

When our XML data is added (cached locally in RDS terms) we can add it to HTML tags on a web page. This process is called **data binding**. The HTML elements that can be bound to the DSO (an XML island is a particular case of the DSO) recognize special attributes:

- DATASRC – the ID of the data source
- DATAFLD – the name of the field in the data source
- DATAFORMATAS – either `TEXT` (default) to display the field value as plain text, or `HTML` to make the browser to render any HTML context within the value.

The Example

Let's see how XML data binding to the tree–like structure works using an example. Imagine we want to build a product catalog for an e–Shop selling computer components. We have products grouped by vendors (Hewlett Packard, Compaq, Microsoft and so on). The products of each vendor are divided into the product groups like printers, disks, monitors – (a detailed list of the products will be in separate pages not covered by this example). We have to publish this catalog implementing two levels of hierarchy on one page. The data for the catalog is formatted in XML:

```
<?xml version="1.0" ?>
<Vendors>
   <Vendor>
      <VendorID>APC</VendorID>
      <VendorName>APC (UPS)</VendorName>
      <PG>
         <pgID>APBK</pgID>
         <pgName>APC Back UPS</pgName>
         <qty>5</qty>
      </PG>
      <PG>
         <pgID>APOT</pgID>
         <pgName>APC Other Products</pgName>
         <qty>2</qty>
      </PG>
      <PG>
         <pgID>APPN</pgID>
         <pgName>APC Protect Net</pgName>
         <qty>2</qty>
      </PG>
      <PG>
         <pgID>APSU</pgID>
         <pgName>APC Smart UPS</pgName>
         <qty>3</qty>
      </PG>
   </Vendor>
   <Vendor>
      <VendorID>CPQ</VendorID>
      <VendorName>COMPAQ</VendorName>
      <PG>
         <pgID>CPQD</pgID>
         <pgName>Compaq PC's</pgName>
         <qty>3</qty>
      </PG>
      <PG>
         <pgID>CPQM</pgID>
         <pgName>Compaq Monitors</pgName>
         <qty>1</qty>
      </PG>
      <PG>
         <pgID>CPQN</pgID>
         <pgName>Compaq Notebooks</pgName>
         <qty>1</qty>
      </PG>
```

```
      <PG>
         <pgID>CPQO</pgID>
         <pgName>Compaq Options</pgName>
         <qty>1</qty>
      </PG>
   </Vendor>
</Vendors>
```

The quantities `<qty>` just show the size of the product list on the second screen.

It's possible just to list all this stuff in plain HTML, but it isn't easy to navigate with many records. Some vendors have similar product groups, so the page could be pretty messy, or we could build a separate page for each level. But we'll go for more attractive solution – we'll to bind this XML to the tree like structure. We are going to implement these pages:

- Catalog.html – product catalog containing vendor and product group information
- Catalog1.html – the same page with XML data in the linked file
- dataforcatalog.xml – XML data file
- Products.html – just to show navigation.

Binding XML to the table

The XML structure presented above consists of the two levels:

- Vendors
- Product Groups

Vendors and product groups have a one–to–many relationship. We can bind data to the table using data source ID:

```
<TABLE ID=tblVendors DATASRC="#Vendors" >
```

We can also bind the nested level (product groups) for the nested table adding it to the next row. This way we can have two rows per vendor:

```
<TABLE ID=tblVendors DATASRC="#Vendors" >
<tr>
...
</tr>
<tr>
<td><table id=tblPG datasrc="#Vendors" datafld="PG" class=cattext >
```

We assign the same data source for the nested level (`datasrc="#Vendors"`), but we also say that we need the data field PG only (`datafld="PG"`). `PG` itself contains a recordset that is bound to the nested table. This allows us to present our XML data as the master/details structure having one or more detail rows (in nested table) for each master row.

Hiding some data

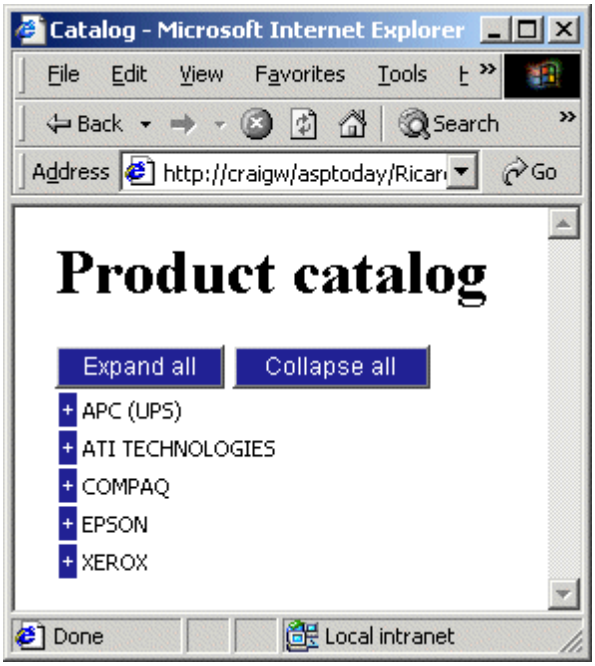
It is possible to control visibility of the HTML tags using the `style` property `display`.

The line `<tr style="display: none">` is used to hide the table row. It's also possible to control visibility using the `style.display` property from script. The final structure of the table with data binding and hidden areas looks like:

```
<TABLE ID=tblVendors DATASRC="#Vendors" style="tah8b" >
<tr>
   <td class="catbutton"
      onclick="toggleDetail(this)" align=center>+</td>
   <td class=cattext><div datafld="VendorName"></div></td>
</tr>
<tr style="display: none">
   <td></td>
   <td>
      <table id=tblPG datasrc="#Vendors" datafld="PG" class=cattext >
      <tr>
         <form action="products.html" method=get>
            <td class="catbutton"
               onclick="submit()" align=center>+
               <input name="pid" type=hidden datafld="pgID" >
```

```
            <input name="pname" type=hidden datafld="pgName" >
            <input name="vid" type=hidden datafld="VendorID">
            <input name="vname" type=hidden datafld="VendorName">
         </td>
         <td>
            <div datafld=pgName nowrap></div>
         </td>
         <td>(</td>
         <td align=right>
            <div datafld=Qty nowrap></div>
         </td>
         <td>)</td>
      </form>
   </tr>
   </table>
</td>
</tr>
</TABLE>
```

We now have the one level list of vendors (see screenshot below). The detail sections are hidden. The + buttons are used to make the page look like a tree like structure (similar to Windows Explorer). The `form` is used to go to the details page – `products.html`. This page is empty in the sample. It is used just to illustrate navigation. The hidden parameters pass the client's choice if the products page was real.



Dynamic show/hide features

It is not hard to hide some data with the option of showing it to customers later. The best solution is to do it dynamically– this is possible by adding the `onclick` function to the buttons. Note that the + actually are just table cells not HTML buttons – IE5 allows managing `onclick` events almost for all HTML tags. As I have mentioned above it's possible to manage visibility of the tags using `style.display` property. There is a JavaScript function to show or hide the second row:

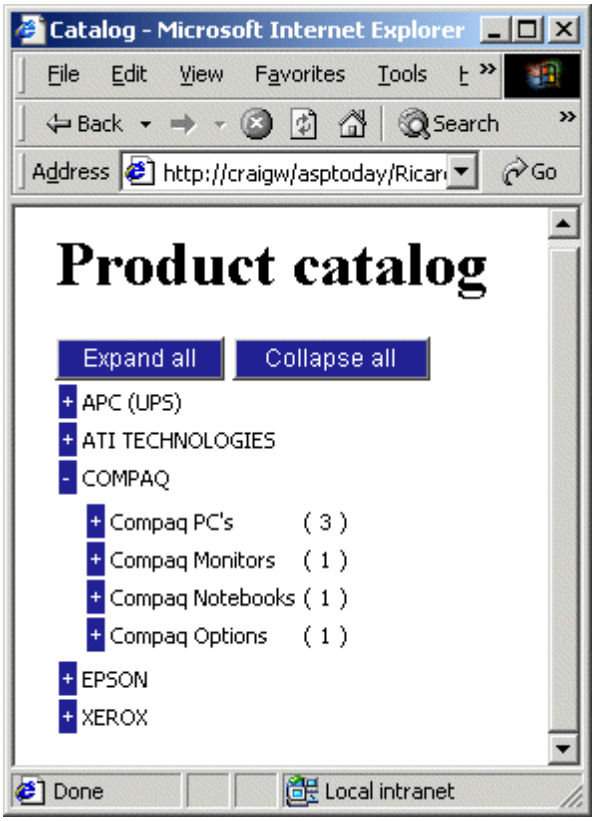
```
function toggleDetail(button) {
   var hiddenTR, parent;
   parent = button.parentElement;
   hiddenTR = parent.nextSibling;
   if (hiddenTR.style.display == "none") {
      hiddenTR.style.display = "";
      button.innerText="–";
   } else {
      hiddenTR.style.display = "none";
      button.innerText="+";
   }
```

```
}
```

The "button" is the cell of the first row, so we need to find the second row navigating though DOM. Using the `parentElement` property we get a first row and move to the next using the `nextSibling` property. Now we have the details table row and can change its display property:

```
hiddenTR.style.display = "none";
```

The function also changes + to – to show expanded /collapsed tree branches.



Saving /restoring the tree view

This tree is very attractive when you work with only one page. Each time we access the page again however all branches are collapsed – it hasn't saved the current state of the tree view. This could be really annoying when working with a lot of data rows – we need to go manually to the place you are interested in each time we access the page. Even the back button doesn't help because the data is re–bound each time. We want to keep the same tree view we left when we access the page again.

Several users could browse the catalog page at once, so the view should be stored for each user separately. The best point to save it is when the user is about to leave the page. It's impossible to detect within ASP logic because the user could go to the page outside the current server. Also we could pass a lot of data to the server to save the view. I have used another technique here. It's possible to save the state on the client's side – and if we need to save something on client we use cookies. The cookies are stored in the `document.cookie` property as a string representing each cookie as a `<name>=<value>` pair delimited by semicolons. I have written a function to extract a cookie by name: `readcookie(cn)` (see the sample page).

DOM of the bound table

To manage the table from the script it necessary to understand the structure of the table stored in the DOM (Document Object Model). There are differences in the DOM between a static (or ASP generated) table and the empty table bound to the XML data. A static table has the structure (simplified):

```
<table>
   <tbody>
      <tr>
         <td>col1<>/td><td>col2<>/td>…
      </tr>
      <tr>
         <td>col1<>/td><td>col2<>/td>…
```

```
      </tr>
...
   </tbody>
</table>
```

The rows of this kind of table could be accessed using

```
<TableName>.firstChild.children(index)
```

where

```
index=<tableName>.firstChild.children.length
```

(i.e. the numbe `r of rows`).

For the bound table the structure is different:

```
<table>
   <tbody>
      <tr>
         <td>col1<>/td><td>col2<>/td>…
      </tr>
      <tr>
         <td>col1<>/td><td>col2<>/td>…
      </tr>
   </tbody>
   <tbody>
      <tr>
         <td>col1<>/td><td>col2<>/td>…
      </tr>
      <tr>
         <td>col1<>/td><td>col2<>/td>…
      </tr>
   </tbody>
...
   <tbody>
...
   </tbody>
</table>
```

Note that now there is a separate `TBODY` object for each XML data row. In our case we have 2 table rows per XML data row. The different `TBODY` elements could be accessed this way:

```
 <tableName>.children(index)
```

where

```
 index=<tableName>.children.length
```

After this we could access individual rows with `firstChild and firstChild.nexSibling` properties (or `children(..)` if you like).

Saving the view

The `document.parentWindow.onunload` event handler is used to catch the point when the user is leaving the page, then calls the function written for saving the current view. It simply scans the table and looks for the display property of each detail row saving it to a string of ones and zeroes (representing the state of respective nodes)and afterward to the cookie.

```
var cookieName="ctv"; // let it be external name to simplify example

function SaveViewTree(){
   var i,n,s;
   s="";
   n=   tblVendors.children.length;
   alert(n);
   for(i=0;i<n;i+=1)
   if (tblVendors.children(i).firstChild.nextSibling.style.display=="none")
      s+="1"
   else
```

```
      s+="0";
   document.cookie=cookieName+"="+s;
   return 0;
}
```

These lines show how the functions are linked to the browser events:

```
document.parentWindow.onunload=SaveViewTree;
document.parentWindow.onload=SetTreeView;
```

Restoring the view

The restoring function is similar to the one provided above for saving. It reads the view string from the cookie and updates the display property of each detail row. It processes expanded branches only because all branches initially are set to be collapsed (visibility= "none"). It also sets – to the buttons depending on visibility.

This function is assigned to the `window.onload` event handler:

```
function SetTreeView(){
   var i;
   var mc=readcookie(cookieName);
   var n=tblVendors.children.length;
   var m=mc.length
   for(i=0; i<m; i++){
      if ( (tblVendors.children(i) == null) ) break;
      if (mc.substring(i,i+1)=="0")
      {
      tblVendors.children(i).firstChild.firstChild.innerText="–";
      tblVendors.children(i).firstChild.nextSibling.style.display="";      }
   }
}
```

One more step

We can now leave this page and will see the same picture on returning. (The other page in the download (`products.html`) is empty and is just to illustrate the saving and restoring of the view of the tree–like structure. You can use any other page if you want). Now let's extend the functionality of our catalog. We already have the technique to manage the visibility of this page so we can add buttons to collapse/expand all branches:

```
<input type=button value="Expand all" onclick="ChangeViewTree(0)" class=catbutton>
<input type=button value="Collapse all" onclick="ChangeViewTree(1)" class=catbutton>
```

The function `ChangeViewTree` changes all the display properties to `none` (not visible) or empty (visible) depending on parameter.

Debugging tips

The Document Object Model (DOM) of MS IE 5.0 (and in other browsers too) is really sophisticated. One has to spend a lot of time to obtain full control over the DOM functionality. (I think it's necessary to read a manual about the DOM, but it isn't enough). To make things a little easier I use MS Interdev for code writing and debugging. One really exiting feature of Interdev (like some other debugging tools) is the ability to review objects. For example if I want to access a part of my document (say a particular row or cell of my table) I could write to the watch window an expression like `tblVendors.children.firstChild.nextSibling.style`. Then I can check the previous sibling, or the first child, or the first child of the next/previous sibling and get the value or list of properties for this element or the error if object path is wrong. After getting the desired results it's easy to paste the expression into the code.

Summary

We have gone through the steps required to build a tree–like structure similar to the familiar Windows Explorer. We have included the functionality of persisting the state of the tree so that it is always the same as the way you left it, a boon if you are looking at a large number of nodes.

Click here to download this article's support material.

RATE THIS ARTICLE

Overall
Poor ○ ○ ○ ○ ○ ○ Excellent
User Level
Beginner ○ ○ ○ ○ ○ ○ Expert
Useful
No! ○ ○ ○ ○ ○ ○ Very
Submit

enter the discussion

Related Articles on ASPToday

XML Hierarchical Data Binding
Leveraging FrontPage Navigation Trees with the HTMLHelp Control
Generating a Tree structure (Asynchronously)
Seeing the Forest: Tree Structures in ASP Pages

Related Links

FIX: XML Data Island Does Not Load Correctly in Internet Explorer

---

keyword search

Search

ASPTODAY Diary

S	M	T	W	T	F	S
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
1	2	3	4	5	6	7
8	9	10	11	12	13	14

93 Matches sort current matches by Author Date Rating

40 responses (0-5): Overall (4.28) | User Level (2.98) | Useful (4.15)
October 10, 2000
Creating Website Discussion Forums – Part II
Robert Clark
In Part I of this series Robert Clark built a threaded Discussion Forum based around an MS Access database. This article builds on the forum software created in Part I by adding further functionality to the system. We will see how easy it is to add search functionality, and to allow the user to sort the forums on a field of their choice. Read on >>

43 responses (0-5): Overall (3.4) | User Level (2.74) | Useful (3.28)
October 5, 2000
Bind XML data to a Tree–Like Structure
Ricardas Kunevicus
If you are looking for a way to create a tree structure like Windows Explorer, and want to be able to persist the state of the structure even when you have browsed to a different site before returning, then read this article. Ricardas Kunevicus uses XML and some DHTML features to create his tree. Read on >>

44 responses (0-5): Overall (3.52) | User Level (3.16) | Useful (3.61)
October 4, 2000
Sessionless State Management Techniques with Source Library
Eric Vollman
Until ASP+, with its long overdue robust and scalable state management capabilities becomes fully available, we have to code our own state management techniques, especially if we want to avoid the drain on resources of session variables. Here Eric Vollman presents us with a fully working sessionless state management library. Read on >>

61 responses (0-5): Overall (4.18) | User Level (2.89) | Useful (3.92)
September 28, 2000
SVG - A New Standard in Web Graphics
Jon Frost
In this article Jon Frost explores a new language that is about to revolutionize the Web nearly as much as HTML did in 1992. Scalable Vector Graphics (SVG) is a new open source text–based language for describing two–dimensional graphical objects in XML, and will herald a new Web era by eliminating the layers of production that content originators currently face. This article briefly explores how SVG emerged and quickly moves into analyzing some SVG code samples - future articles will cover a complete working mapping application. Read on >>

27 responses (0-5): Overall (3.67) | User Level (2.85) | Useful (3.26)
September 27, 2000

An Introduction to Windows Management Instrumentation (WMI) with ASP
Srinivasa Sivakumar
Web-Based Enterprise System Management (WBEM) is one of the main areas where many corporations spend tons of money to manage their IT infrastructure. In this article, Srinivasa Sivakumar introduces the Windows Management Instrumentation (WMI), Microsoft's implementation of WBEM, and opens the door to a vast ocean of possibilities for the web developer. Read on >>

56 responses (0-5): Overall (3.27) | User Level (2.5) | Useful (3.39)
September 21, 2000
ADO and SQL Error Trapping in ASP
Pieter Siegers
Error trapping, as we know, is an important part of any application. Peter Siegers takes us through the tricky practice of error handling when using multiple frames in your web application. Read on >>

41 responses (0-5): Overall (2.27) | User Level (2.61) | Useful (2.34)
September 18, 2000
The World's Simplest ASP2HTML Viewer
Dino Esposito
What if you need to check the HTML source code that IIS generates given an ASP page? If you need this only for debugging purposes, then it's a non-issue. Simply navigate to the URL and view the page source code from the browser's menu. There might be other good reasons, though, that lead you to investigating the the possibility of a silent, programmatic server–side tool to convert ASP pages into HTML - as Dino Esposito does in this article. Read on >>

285 responses (0-5): Overall (3.51) | User Level (2.19) | Useful (3.51)
September 6, 2000
Handling ASP VBScript Errors at the Page Level
Cary Marr
Handling errors is probably not the first thing that crosses your mind during application development. However, for a web application that oozes confidence even when it isn't working, gracefully dealing with errors can be the difference between retaining site traffic and losing your customers. Cary Marr describes some neat practices to make your crucial error-handling code easier to maintain and scale. Read on >>

131 responses (0-5): Overall (3.61) | User Level (2.17) | Useful (3.55)
August 28, 2000
Setting Up a Basic Online Store
Alik Levin
When someone mentions the words “internet application…” we usually think ‘online store.’ Although there are many other kinds of application, this is the most common. Here Alik Levin demonstrates the basic process of building a simple yet working application implementing the front end of an online store. Using the methodology and code presented here will give you a head start in building your own web apps.
Read on >>

151 responses (0-5): Overall (3.89) | User Level (2.46) | Useful (3.83)
August 15, 2000
Creating Website Discussion Forums – Part I
Robert Clark
If you have ever used Microsoft's NNTP forum service, you may be aware that, although simple to set up on the server side, the end user needs to create and configure a news server account for your site. Here Robert Clark shows us that creating your own easy to use custom discussion forum need not be too painful, and can bring great benefits to you and your customers. Read on >>

210 responses (0-5): Overall (3.33) | User Level (2.28) | Useful (3.2)
July 14, 2000

Maximize Your ASP Performance

We have discovered that this article has appeared elsewhere and it has therefore been deleted. Read on >>

57 responses (0-5): Overall (2.79) | User Level (2.49) | Useful (2.75)
June 27, 2000
Browser Safe Code – Exposing a Browser's DOM to the Server
Brydon Gilliss
Browser compliance is a necessary evil in this industry and there are plenty of resources available for tackling this problem. They tend to fall into one of two categories: – client–side solutions and server–side solutions. This article discusses a best of both worlds technique that combines the two. Read on >>

223 responses (0-5): Overall (3.25) | User Level (2.42) | Useful (3.04)
June 19, 2000
Using Stored Procedures With ASP, ADO and SQL Server
Jesse Ehrenzweig
By using stored procedures in conjunction with ActiveX Data Objects (ADO), you can exert a greater degree of control over how output from your stored procedure is handled when compared to embedded SQL statements. However, it requires patience to open every stored procedure one wants to interface with, take note of the various parameters to be used and convert this knowledge into ASP code. Enter Microsoft's new ADO Extensions for Data Definitions and Security, also known as ADOX, which gives any ASP developer the ability to programmatically view information about a SQL Server database. As you will soon learn, using a combination of these tools can help save you more than a few gray hairs, and many, many bleary-eyed hours in front of the computer. Read on >>

130 responses (0-5): Overall (2.86) | User Level (2.57) | Useful (2.88)
June 12, 2000
HTML Template Processing using VBScript with ASP
Sharat Devnoor
This article describes a technique that mimics the ability of Webclasses to achieve separation of code and HTML through the use of HTML Template files. Since some ISPs do not allow the use of Webclasses or any COM components for ASP, Sharat Devnoor has developed a couple of VBScript routines to do your very own Template Processing. Using VBScript, the template processing library developed here has proven to be very useful for ASP programming, resulting in much cleaner ASP pages. Read on >>

84 responses (0-5): Overall (4.05) | User Level (3.04) | Useful (4.24)
June 6, 2000
Monitor Your Web Site Performance with the XMLHTTP Component Part 1
Alex Homer
The MSXML component that Microsoft provides with IE5 and Windows 2000 exposes a useful object called XMLHttpRequest. This is used to send an HTTP request to a remote or local Web server, and collect the server's response. Although predominantly intended for use when fetching or posting XML documents, it can also be used to fetch other kinds of documents as well - for example in monitoring a number of websites to check their performance and ensure that they have not 'fallen over' while you weren't looking. Read on >>

104 responses (0-5): Overall (3.68) | User Level (3.03) | Useful (3.51)
May 16, 2000
Using the Bill of Materials Structure
Geoffrey Pennington
There are times when a software developer may come across a client whose business rules are complicated and difficult to model. The Bill of Materials structure is the standard solution when you have one table that has a many-to-many relationship, not with another table, but with itself. Because tables with a many-to-many relationship to themselves are less common, many developers have not seen them. Geoff Pennington explains this relationship and shows how to use it with a simple ASP application. Read on >>

320 responses (0-5): Overall (3.32) | User Level (2.29) | Useful (3.31)
May 12, 2000
Creating a Simple Shopping Cart
Mohammed Mudassir
From the requirements of an ideal cart to the implementation of a first, simple cart, Mohammed Mudassir takes you through the stages to create your own shopping cart. In this, the first of a two-part article, he introduces a neat trick for easy coding of a dynamic array and in the next he will address the remaining requirements of the ideal cart. Read on >>

388 responses (0-5): Overall (3.81) | User Level (2.7) | Useful (3.77)
April 14, 2000
Dynamic Selection Lists
Geoffrey Pennington
There are several ways in which we might get the contents of a selection list to change depending on the selected option in another selection list. The usual solutions depend either on server side techniques that refresh the entire screen when a selection changes or on hard-coded client-side code. In this article, Geoff Pennington develops a solution that uses server-side code to generate browser-side code, thus avoiding wasteful round trips to the server or the inflexibility of hard-coded client side script. Read on >>

116 responses (0-5): Overall (3.34) | User Level (2.85) | Useful (3.34)
March 30, 2000
Language Localization for Enterprise Web Applications
Andrew Krowczyk
Language localization is the process of displaying the user interface in different languages depending on the locale that the application is being run from. Localization for Internet applications has long been a cumbersome and difficult task for developers to tackle. Many approaches have been made to make this functionality easier to implement and less costly to maintain. In this article I will outline an approach to localization that can be used on enterprise class web applications. This approach will reduce code redundancy and total cost of ownership while allowing web developers to cut those long hours of head banging out of their day. Read on >>

165 responses (0-5): Overall (3.88) | User Level (3.13) | Useful (3.75)
March 28, 2000
Using JavaScript's OO Capabilities for Advanced Client-Side Validation
Luke Boucher
Consider form feedback. Much of the HTML form feedback we see at the moment is of the negative variety. Developers are wise to the average user's capacity to input bad data, and have been quick to develop JavaScript-based form checking that ensures numeric fields actually contain numeric values, and that date fields contain real dates. However, many of these implementations force the user to enter values in a fixed and restrictive format, and provide little or no flexibility. In contrast, Luke Boucher takes form validation a step further, showing you how to utilize JavaScript's object-oriented capabilities to give users the type of positive feedback and flexibility that they are accustomed to using desk-top applications such as Access and Excel. Read on >>

532 responses (0-5): Overall (2.97) | User Level (1.37) | Useful (2.8)
March 24, 2000
Query String Basics with ASP
Ken Baumbach
There are four basic ways to pass information from one web page to another: session variables, forms, cookies, and query strings. Focussing on a real world example, Ken Baumbach shows us in this article how to utilize query strings in a way which demonstrates their flexibility and simplicity. Read on >>

527 responses (0-5): Overall (3.77) | User Level (2.78) | Useful (3.46)
March 17, 2000

Displaying a 'Progress Bar' in a Web Page
Alex Homer
A common feature of an ordinary Windows application is the ubiquitous 'progress bar'. However, click on a web page and very often, if the page we've selected is processing some query or is engaged in some other computations, we are provided with scant information about what, if anything, is happening, and how long it's likely to take. In this article, Alex Homer shows us how we might create a display page which tracks the progress being made on the client. Read on >>

1486 responses (0-5): Overall (4.33) | User Level (3.31) | Useful (4.37)
March 16, 2000
File Uploading with ASP and VBScript
Philippe Collignon
Philippe Collignon presents a complete VBScript solution to uploading files. The sample code provided here is pure VBScript and is independent of third party products, that are often expensive and usually unable to be customized. Though VBScript may not be the obvious choice because it is a scripting language designed to work only with datatype variants and it does not provide many built-in functions to manage binary data and byte arrays, it does provide a very light-weight, but powerful solution to your uploading requirements. Read on >>

65 responses (0-5): Overall (3.85) | User Level (2.66) | Useful (3.66)
February 28, 2000
Experimenting With MIME-Types and Content Types - Part II
Alex Homer
Alex Homer concludes his discussion on some of the other MIME-types and content types in action, and builds a simple tool to allow you to experiment in more detail with your own applications and custom file types. Read on >>

185 responses (0-5): Overall (3.81) | User Level (2.5) | Useful (3.59)
February 22, 2000
Understanding MIME Types and Content Types
Alex Homer
Although its not possible to control how the browser or client application handles any file that you send from your Web server, Alex Homer shows us how to "tell" the browser what the file type is, leaving it to the browser and the user to decide what to do with it. Read on >>

195 responses (0-5): Overall (2.88) | User Level (2.07) | Useful (2.66)
February 21, 2000
Building Multi-Column Select Lists Using ASP
Nathan Babb
Didn’t think it was possible to have multi-column list boxes? Well, it’s not quite out-of-the-box functionality, but in this article Nathan Babb shows you how to create multi-column list boxes using just ASP code. Read on >>

162 responses (0-5): Overall (3.57) | User Level (2.36) | Useful (3.64)
January 27, 2000
Using Embedded Fonts in Web Pages
Steve Smith
Since a client's browser and operating system influence the display of a document, two products have been developed which allow the web publisher to include all of their font information with their web page. Steve Smith assesses and compares these two products. Read on >>

337 responses (0-5): Overall (3.58) | User Level (2.37) | Useful (3.67)
January 20, 2000

Customizing the '404 Not Found' Error
Greg Hinkle
The '404 Not Found'errors indicate a missing file, a bad link, or a crashed server. But, Greg Hinkle shows how, using IIS, a SQL database, and good old ASP, we can take advantage of the 404 message. Read on >>

237 responses (0-5): Overall (3.02) | User Level (1.99) | Useful (2.81)
January 3, 2000
Tip of the Day in ASP
David Schultz
Dave Shultz builds simple tip of the day functionality, which you can import and use on your pages, and shows you how to use and improve it. Read on >>

140 responses (0-5): Overall (3.24) | User Level (2.37) | Useful (3.09)
December 24, 1999
Echo Back - Screen Persistence of a Browser's Forms
Geoffrey Pennington
In this article, Geoff Pennington presents a simple technique he calls "echo back" which introduces "screen persistence", providing one more chance to review entries of a form in your browser. Read on >>

656 responses (0-5): Overall (3.61) | User Level (2.22) | Useful (3.54)
December 14, 1999
Tips for Installing Personal Web Server on Win 9x
John Kauffman
The main strength of PWS is that it provides a convenient and inexpensive development environment for pre Windows 2000 machines, without the need for powerful hardware. John Kaufman leads us through the advantages and installation of PWS. Read on >>

245 responses (0-5): Overall (3.4) | User Level (2.76) | Useful (3.23)
December 9, 1999
Tracking Client Behaviour on Your Website
Dr. GuoQing Hu
Dr. GuoQing Hu discusses how he used ASP to address those sometimes tricky statistical questions that inevitably arise when you have a web site Read on >>

313 responses (0-5): Overall (3.5) | User Level (2.37) | Useful (3.33)
December 8, 1999
Managing Connection Strings with Data Link Files
Laura Granstedt
Still embedding ADO connection strings in your source code? It might be time to consider storing them in Data Link Files: Laura Granstedt tells us how and why. Read on >>

688 responses (0-5): Overall (3.88) | User Level (2.79) | Useful (3.77)
December 7, 1999
The Implications of ASP #include
Eric Helliwell
Ever wondered what exactly is going on with your include files? Eric Helliwell tells you everything you'll ever need to know about them - when to use them, when not too, and how to optimize performance, and what changes in ASP 3.0 Read on >>

628 responses (0-5): Overall (4.23) | User Level (2.95) | Useful (4.13)
November 19, 1999

WAP and ASP - Part II
Luca Passani
Following Monday's article introducing WAP and WML, Luca Passini starts us of in the world of WAP/WML programming with a ticket booking application, and discusses the current pitfalls of this exiting new technology. Read on >>

1074 responses (0-5): Overall (4.03) | User Level (2.44) | Useful (3.89)
November 15, 1999
WAP and ASP - Part I
Luca Passani
WAP is a new protocol that will turn mobile telephones into small Internet browsers. Luca Passani goes into WAP basics and how it can be used with ASP. Read on >>

165 responses (0-5): Overall (3.32) | User Level (2.25) | Useful (3.44)
September 29, 1999
Website Internationalization - using ASP
Andrea Chiarelli
Time to make your site multi-lingual? Andrea Chiarelli shows how to use the Accept-Language request header field in your ASP pages to detect the user's preferred language, and load up the appropriate pages and settings. Read on >>

58 responses (0-5): Overall (3.33) | User Level (2.5) | Useful (3.57)
September 6, 1999
Watch Your Script Engine Version
Alex Homer
Wonderful though your Script Engine might be, if the client machine is using an earlier version it won't do you a lot of good - unless, that is, you've coded for all eventualities. Alex Homer shows how it's done. Read on >>

231 responses (0-5): Overall (2.95) | User Level (2.39) | Useful (2.84)
August 27, 1999
A short-and-sweet implementation of the Likert-Scale based survey
Brian Reagan
A large number of marketing surveys use the Likert-scale (e.g. 1=good, 2=neutral, 3=awful). Brian Reagan explains how to design such a scale for your web site. Read on >>

295 responses (0-5): Overall (3.46) | User Level (2.77) | Useful (3.59)
August 25, 1999
Using Active Server Pages to Build Microsoft Word Documents
Gardiner B. Jones
An increase in dynamic web pages has promoted the creation of form letters from the changed records of a web page table. Gardiner B. Jones shows us how this can be done. Read on >>

393 responses (0-5): Overall (2.87) | User Level (1.27) | Useful (2.67)
August 23, 1999
Displaying Database Query Results
Dan Smith
Dan Smith shows how to display a simple database query using the recordset. Read on >>

65 responses (0-5): Overall (2.95) | User Level (2.69) | Useful (2.74)
August 20, 1999

Maintaining User State with the Active User Object (AUO)
Craig McQueen
Not only does AUO increase the scalability and fault tolerance of your site, but it's also useful if you're curious about the way your site is being used. Craig McQueen explains why the days of the Session object are numbered. Read on >>

162 responses (0-5): Overall (3.8) | User Level (2.57) | Useful (3.93)
August 19, 1999
DIY VB Components for ASP
Darren Gill
Dreading the day your site needs that little bit more? Armed with Darren Gill's step by step guide you'll find writing your first VB component a lot less daunting than you might have feared. Read on >>

29 responses (0-5): Overall (2.52) | User Level (2.28) | Useful (2.52)
August 13, 1999
Creating Recordset using RDS
Ajoy Krishnamoorthy
Want to know how to create an ADODB recordset using RDSServer and insert records? Ajoy Krishnamoorthy explains how. Read on >>

199 responses (0-5): Overall (3.43) | User Level (2.45) | Useful (3.43)
August 12, 1999
The Use of ActiveX DLLs with ASP
Keith Stanislaw
Let ASP create html comboboxes that are populated from the database; all you need is a dll and Keith Stanislaw to explain how. Read on >>

145 responses (0-5): Overall (3.79) | User Level (2.8) | Useful (3.77)
August 4, 1999
Querying Index Server using ADO
Rob Hebron
Index Server 2.0 is a powerful server side tool that alleviates the need for certain components. Rob Hebron investigates some of its lesser-used capabilities, and demonstrates the search engine sophistication it permits. Read on >>

417 responses (0-5): Overall (3.62) | User Level (2.21) | Useful (3.55)
August 3, 1999
Creating a User-Friendly Search Engine For An Access Database
Alexander Haneng
Alexander Haneng creates a search engine for a Virtual CD shop, demonstrating how easy it is to combine DSN-less db searches, query strings, form validation, and recordset paging in one application. Read on >>

50 responses (0-5): Overall (4.08) | User Level (3.18) | Useful (4.04)
July 29, 1999
A Simple Windows Scripting Component (WSC)
Alex Homer
Windows Scripting Components are the updated, XML-oriented, new and improved Scriplets. Combining the usability of the older technology with loads of functionality, they can go anywhere COM can go. Alex Homer starts us off with a simple one. Read on >>

97 responses (0-5): Overall (2.95) | User Level (2.21) | Useful (2.68)
July 28, 1999

Advanced Survey Forms (Wizard Type CGI Forms)
Sunit Katkar
Sunit Katkar creates a CGI form that behaves like Windows Wizard - and explains the behavior of ASP's Checkboxes and Radio buttons for those who, like himself, come from a Perl background Read on >>

69 responses (0-5): Overall (3.26) | User Level (2.54) | Useful (3.45)
July 27, 1999
An Introduction to MSMQ with ASP
Chris Blexrud
If your page takes so long to process that your visitors are leaving in droves, it might be time to investigate MSMQ. Chris Blexrud explains. Read on >>

37 responses (0-5): Overall (3.59) | User Level (3.11) | Useful (3.84)
July 22, 1999
Writing to the Event Log
Rolando López
If your logs files are in a bit of a mess, and decent analysis of some fairly essential error detail takes forever, then Rolando López's thorough examination of Windows NT Event Log and Event Viewer, with code download and ASP VB component, might be just the thing. Read on >>

192 responses (0-5): Overall (3.24) | User Level (1.67) | Useful (3.08)
July 21, 1999
Simple ASP Debugging Tricks - A Beginner's Guide
Steven Schofield
As every programmer knows, a significant amount of coding time will always be spent debugging. Steven Schofield shares a few of his ASP debugging tricks that have stood the test of time. Read on >>

91 responses (0-5): Overall (3.49) | User Level (2.36) | Useful (3.44)
July 20, 1999
Form Driven Queries
Geoffrey Pennington
Constructing the queries to your SQL database on the fly means that you really can get at the information you want and display it as you choose - and it's not even difficult. Geoff Pennington provides code and explanations. Read on >>

174 responses (0-5): Overall (3.76) | User Level (2.72) | Useful (3.84)
July 19, 1999
Pulling images from an Access DB
Dino Esposito
Following Dino Esposito's article 'Pulling Images from Databases', we've had a lot of queries about Access databases. It can be done! Here he shows that it really is relatively simple to retrieve and display images from both Access and FoxPro databases. Read on >>

133 responses (0-5): Overall (3.77) | User Level (2.38) | Useful (3.9)
July 16, 1999
Using Perl in Active Server Pages
kent Tegels
Kent Tegel's explains why using Perl with your ASP can revolutionise your programming experiences, where to use it and how to put it all together. Read on >>

212 responses (0-5): Overall (3.1) | User Level (2.03) | Useful (3)
July 15, 1999

Using Microsoft FrontPage to Create ASP pages
Kevin Spencer
If you can't use Microsoft FrontPage because of the things it does to your code, it may be that all you need is somebody to explain the editor's little idiosyncrasies. Kevin Spencer firmly argues the case for FrontPage. Read on >>

149 responses (0-5): Overall (3.68) | User Level (2.59) | Useful (3.75)
July 14, 1999
ASP Uploading - The Contenders
Steve Smith
There are plenty of HTTP POST uploading options - download, build or buy the component. Steven Smith shares his experiences with the MS Posting Acceptor, Persists Software ASPUpload and Software Artisan's SA-fileUp. Read on >>

275 responses (0-5): Overall (2.95) | User Level (1.51) | Useful (2.86)
July 12, 1999
Basic Password Protection
Ken Baumbach
Password protection isn't always about enormous databases, heavy-handed security and reams of code. Ken Baumbach adds a neat little trick to the database discourse. Read on >>

179 responses (0-5): Overall (2.84) | User Level (2.08) | Useful (2.56)
July 8, 1999
Advanced Form Validation
Andrea Chiarelli
What happens when the forms in your web page are mistakenly resubmitted or partially submitted by the user? If it causes you problems, have a look Andrea Chiarelli's JavaScript solution Read on >>

95 responses (0-5): Overall (3.56) | User Level (2.22) | Useful (3.56)
July 7, 1999
What's in your ASP toolkit?
Rob Docherty
If the code duplication in your applications is generating a lot of clutter, it's probably time to get all those utility routines, database connections, declarations and the rest in your own custom made ASP Toolkit. Rob Docherty organises ASP. Read on >>

18 responses (0-5): Overall (2.72) | User Level (3.33) | Useful (2.83)
July 6, 1999
Bitmasks and Bitwise Operators
Michael Corning
Bitmasks and Bitwise operators don't have to be confusing. Michael Corning introduces and explains the mechanics of bitwise operations and the code, and gets to the nitty-gritty of what the computer actually does with all of it. Read on >>

86 responses (0-5): Overall (3.76) | User Level (2.62) | Useful (3.92)
June 28, 1999
Determining User Attributes In ASP
Kent Tegels
It might be possible to determine user access with User manager, or even by querying the Domain Context, but it can be a whole lot easier in ASP. Kent Tegels explains. Read on >>

60 responses (0-5): Overall (3.55) | User Level (2.45) | Useful (3.48)
June 24, 1999

Easy -To- Maintain Navigation with XML
Paul Spencer
Combining XML and ASP makes for easy web site maintenance and design changes, by separating style from content. In the first of two articles, Paul Spencer introduces XML and looks at its uses for ASP supporting browsers. Read on >>

40 responses (0-5): Overall (3.7) | User Level (3.1) | Useful (3.9)
June 15, 1999
Data Shaping 101
Joe Graf
Data shaping, otherwise known as hierarchical recordsets, allows a column of your master recordset to contain a child recordset of its own. Joe Graf introduces us to the ideas, and guides us through the code. Read on >>

12 responses (0-5): Overall (3.33) | User Level (2.67) | Useful (2.92)
June 14, 1999
Automatic Paged List Operations Using ASP
Alex Homer
Long-winded management or administration pages, which have a tendency to crash and burn during processing, can be considerably improved if you break down the process, keep track of its progress, and get the page to reload automatically. Alex Homer demonstrates. Read on >>

52 responses (0-5): Overall (3.77) | User Level (2.48) | Useful (3.88)
June 10, 1999
Benchmarking Different Approaches to Updating a Database
Geoffrey Pennington
There are many different ways of sending a delimited text file to SQL Server. Geoff Pennington compares and contrasts five of the best. Read on >>

78 responses (0-5): Overall (3.37) | User Level (2.13) | Useful (3.27)
June 4, 1999
Simple Visitor Access Control
Alex Homer
You can choose to keep track of visitor access to your site either through cookies or through ASP Sessions. Alex Homer gives us the pros, the cons, and the code for the both. Read on >>

31 responses (0-5): Overall (3.87) | User Level (2.84) | Useful (4.06)
June 2, 1999
Fifth Generation Serve Side Scripting - Part I
Michael Corning
Michael Corning introduces the new series on scripting server-side XML applications, taking as his basis an XML spreadsheet app wrapping data from the SQL Pubs directory into XML. Read on >>

127 responses (0-5): Overall (3.25) | User Level (2.23) | Useful (3.21)
May 24, 1999
Pulling Images From A Database
Dino Esposito
One perennial question for web developers is exactly how to pull images successfully from a database. Dino Esposito provides the solution. Read on >>

22 responses (0-5): Overall (3.09) | User Level (3) | Useful (3.5)
May 20, 1999

the Dilemma of Implemented interfaces and VBScript
Jason Bock
VB components utilise the COM specification which enables other tools and languages to use these components. This causes the ASP developer problems, because the implemented interfaces are not the default interface. Jason Bock demonstrates a solution. Read on >>

183 responses (0-5): Overall (3.94) | User Level (2.63) | Useful (3.65)
May 14, 1999
A 'Good Enough' Login Component using ASP/VB
Doug Dean
In this artcle, Doug Dean demonstrates how you can readily create a simple, effective and inexpensive user authentication mechanism, using a VB component with the ASP poages on your personal website.
Read on >>

50 responses (0-5): Overall (2.38) | User Level (1.8) | Useful (1.84)
April 27, 1999
Creating Dynamic Pop-up Windows
Alex Homer
Pop-up windows do not often engender users to your site, but with judicious use, they can be an effective marketing tool. Alex Homer demonstrates how to code them. Read on >>

298 responses (0-5): Overall (3.92) | User Level (2.92) | Useful (3.86)
April 22, 1999
Scaling VB COM with MTS
Robert Howard
Robert Howard discusses the shortcomings of the Session object, and how to overcome these by using MTS. Read on >>

82 responses (0-5): Overall (3.52) | User Level (2.49) | Useful (3.8)
April 21, 1999
Easy Debugging Tips and Tools
Steve Smith
No new ASP page is 100% bug free - glitches comes with the territory. As you face the uphill debug struggle, Steven Smith offers some routines and tips to make your debugging run on a more level playing field. Read on >>

147 responses (0-5): Overall (3.01) | User Level (2.05) | Useful (2.94)
April 13, 1999
Using Global.asa Correctly
Jeff Johnson
Getting the desired results from global.asa and having it execute properly is a common problem among ASP developers. Jeff Johnson sheds a light on the surrounding confusion between virtual directories, virtual applications and when global.asa applies where. Read on >>

33 responses (0-5): Overall (3.76) | User Level (2.82) | Useful (4.03)
April 9, 1999
Dealing with Ignored Words in Index Server
Adam Roberts
Adam Roberts shows how to deal with the problems caused by noise words in Index Server searches.
Read on >>

56 responses (0-5): Overall (3.84) | User Level (2.45) | Useful (3.8)
April 1, 1999

XML Hierarchical Data Binding
David Sussman
It's common knowledge that you can bind an XML data source to an HTML table in IE5, but you may not know that you can bind hierarchical XML data into nested tables as well. David Sussman shows you how. Read on >>

103 responses (0-5): Overall (3.84) | User Level (2.69) | Useful (3.78)
March 31, 1999
Persisting Form Control Values with ASP
Alex Homer
Persisting form values both forward to and back from different pages can be a lot of work. Alex Homer demonstrates a reliable way of doing both using a dynamic array within a session variable. Read on >>

65 responses (0-5): Overall (3.62) | User Level (2.75) | Useful (3.51)
March 29, 1999
Server-Side Redirection In IIS5.0
Christian Gross
With IIS5.0 comes the promise of server-side page transfers. But, as Christian Gross discovers, the new methods come with a call for careful coding as well as all the advantages you would imagine. Read on >>

123 responses (0-5): Overall (2.17) | User Level (1.43) | Useful (1.98)
March 22, 1999
A Beginners Guide to Cookie Disabled Sessions
Andrea Chiarelli
ASP acts as a wrapper that introduces state to a stateless protocol. But this functionality is based upon cookies being enabled. Andrea Chiarelli presents a beginner's guide to maintaining session information when cookies are disabled. Read on >>

123 responses (0-5): Overall (3.65) | User Level (2.45) | Useful (3.67)
March 19, 1999
Browsing Directories With ASP
Matt Bullock
A good demonstration of the FileSystemObject object's capabilities is to write a directory browser. While this is easily done, Matt Bullock demonstrates several extensions to this simple idea. Read on >>

106 responses (0-5): Overall (3.65) | User Level (2.73) | Useful (3.82)
March 12, 1999
Creating an Editable Grid
Geoffrey Pennington
What if you needed to view and update several database records at a time? Geoffrey Pennington rolls his own form-based grid control allowing dynamic access to multiple fields in a database. Read on >>

45 responses (0-5): Overall (2.8) | User Level (2.44) | Useful (2.82)
March 8, 1999
Error Handling In IIS5.0
Christian Gross
With IIS 5.0 around the corner, we will have two ways to handle errors at our disposal. Christian Gross looks at both and asks which to use when. Read on >>

99 responses (0-5): Overall (2.79) | User Level (1.94) | Useful (2.77)
March 2, 1999

Using Dynamic Arrays In ASP
Alex Homer
Multi-dimensional arrays provide a neat way to hold values for all kinds of ASP coding activities. Alex Homer demonstrates the basic ways to use this hold all for persisted form values Read on >>

10 responses (0-5): Overall (3.4) | User Level (2.3) | Useful (3.3)
February 26, 1999
Exploring The Form Manager DTC
Ian Blackburn
Visual Interdev's Form Manager DTC is a great way to manage several different variations (or modes) of an HTML form based on a single page. Ian Blackburn demonstrates Read on >>

67 responses (0-5): Overall (3.04) | User Level (2.34) | Useful (3.07)
February 25, 1999
Frames without the tag
Rob Barker
Frames present an easy layout option for sites but a more awkward housekeeping problem. When frames are not required, tables are used for layout. They can also emulate the frame window system too.. Read on >>

77 responses (0-5): Overall (3.65) | User Level (2.01) | Useful (3.81)
February 19, 1999
Preventing Your Pages From Being Indexed
Jon Duckett
It may be easy to publish private pages people on the web for others to see, given their address, but web spiders and robots may find them too. Jon Duckett looks at how to keep your private pages private. Read on >>

94 responses (0-5): Overall (3.43) | User Level (2.35) | Useful (3.4)
February 18, 1999
Buffering and Redirection in ASP 2.0
Alex Homer
Response.Redirect often backfires for webmasters who publish their webs from behind a firewall or proxy server. Alex Homer discusses why and how to solve the problem. Read on >>

60 responses (0-5): Overall (3.83) | User Level (2.58) | Useful (3.62)
February 16, 1999
Creating a Client-Side Variable Array
Alex Homer
Client side data caching is becoming ever more relevant. Where can you store transient data. Alex Homer looks at one possible solution - client-side variable arrays. Read on >>

104 responses (0-5): Overall (3.6) | User Level (2.32) | Useful (3.67)
February 12, 1999
Parsing Strings With ASP
Alex Homer
Parsing a string into separate values can be a nightmare especially if differing delimiters are used. ASP however alleviates that bad dream if used correctly. Alex Homer shows you how. Read on >>

100 responses (0-5): Overall (2.84) | User Level (1.65) | Useful (2.68)
February 11, 1999
A Simple Page Counter
Shawn Alley
Intranet Administrators may well want to know which pages are the most visited and which could be better positioned. Shawn Alley gives us the means to do so with a simple web page counter. Read on >>

206 responses (0-5): Overall (3.48) | User Level (2.14) | Useful (3.44)
February 10, 1999
Passing Values Between Pages
Alex Homer
Forms are the standard way of keeping values alive between pages. Alex Homer looks at using recursive ASP pages with hidden form elements to turn your forms into wizards Read on >>

30 responses (0-5): Overall (3.73) | User Level (2.73) | Useful (3.7)
February 9, 1999
Using The Page Object DTC
Ian Blackburn
Visual Interdev's Page Object DTC presents an easy way to maintain state information and to execute script remotely. Ian Blackburn shows you how Read on >>

keyword search

Search

Home | Today's Article | Search | Feedback | Write For Us | Suggest an Article | Advertise

Oct 04, 2000

Sessionless State Management Techniques with Source Library

By Eric Vollman

ASP Tricks

enter the discussion

A wealth of information is available to ASP developers explaining the concepts of sessionless state management, why it would be employed, and general concepts for implementation. This article instead takes a "how–to" approach and focuses on actual implementation of a complete sessionless State Management System using an ASP code library (provided in the download). The State Management Library is an Active Server Page include file that I developed out of necessity to support ASP applications deployed in web server–farm environments. The functions and subroutines contained in the include file implement a complete sessionless State Management System with the following capabilities:

- State persistence through query string, hidden form field, or cookie
- Encryption of persisted state information
- Persisted state information tamper detection
- Persisted state information lifetime management

The State Management System embodied in the library masks the drudgery of implementing sessionless state by reducing the effort to a few simple function calls in the ASP page. The functions retrieve state information as it is passed from page–to–page; exposing the state information to page program logic, and persisting state information to make it available for subsequent pages. The developer can concentrate on the task at hand and need only be concerned that the state management functions are called and sequenced properly.

Implementation and Operational Concepts

The library is written entirely in VB Script and employs the intrinsic Scripting Dictionary object as a replacement for the ASP Session object. The Dictionary performs the same role as the Session object and is utilized to implement a **State Collection** containing key and value pairs representing application state. The library provides functions and subroutines to interact with the State Collection items in a manner similar to that of the Session object, however, maintaining state information is vastly different. Unlike the Session object, the State Collection method must be persisted using library functions to preserve the state information between pages. Persistence involves a process termed **State Serialization**.

State Serialization

The State Management System implements a model that serializes all state value and key pairs in a single structure to achieve operational simplicity. Serialization is a process where all of the key and value pairs contained in the State Collection Dictionary are extracted and placed as delimited values in a single string. This delimited string then becomes the value for a specifically named key in a query string, a single form field, or cookie key.

The capabilities provided by the State Management Library enable state persistence of string and numeric data. It cannot serialize and persist objects like an ADO recordset or complex structures like arrays. Objects and complex structures such as those described are simply not suitable for the persistence methods implemented.

Serializing state items in a single string allows library processes to retrieve and persist information without requiring specifics of each and every piece comprising the application state. State information is always persisted using a specific identifier.

ADSI/CDO (13)
ASP Tricks (93)
ASP+ (12)
BackOffice (32)
Components (74)
Data Access (126)
Miscellaneous (35)
Non-MS ASP (10)
Scripting (82)
Security/Admin (44)
Site Design (42)
Site server (13)
XML (63)

free email updates

email address

Register

ASPTODAY Diary

S	M	T	W	T	F	S
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
1	2	3	4	5	6	7
8	9	10	11	12	13	14

Links
Author Page
About ASPToday

State Persistence

Serialized state may be persisted as a query string, hidden form field, or within a cookie, and program logic must specify which method is to be performed. The library processes cannot provide total persistence because state must be passed with every hyperlink and form submission to ensure application state is maintained. Instead, the library provides a function that returns state in a persistence format that program logic can append to hyperlink resources or embed in a form.

- Query String persistence returns a complete query string, including the question mark delimiter that can be appended to the name of a hyperlink resource by a `Response.Write` in the HTML stream.
- Form field persistence returns a complete HTML hidden form field tag to output in the HTML stream in the same manner.
- Cookie persistence is handled completely by the persistence process.

When persisting state, library processes enumerate the state collection, serialize the keys and values it contains and output state information using the same identifier. Depending on the method selected, the identifier becomes a Query String key, the name of a form text box, or the name of a cookie key. Retrieval functions understand that persisted state information will be associated with the identifier regardless of persistence method.

State Retrieval

When retrieving state, library processes do not require knowledge of the state information location – a query string, form submission, or located in a cookie. The retrieval process simply attempts to locate it's specific key in the ASP Request Collection or in the Request Cookie Collection. When library processes retrieve persisted state, the serialized structure is obtained from the ASP Request collection, parsed into key and value pairs, and placed in the State Collection context where they are available to page program logic.

Application State Security and Integrity

Security and application integrity can be an issue because of the open nature of sessionless state management. Significant issues exist for web applications because state information is passed in the open. State information is clearly visible on the URL or within the source of the page sent to the browser. With state information passed in this manner, there is nothing to prevent alteration of values on the query string and requesting the page again with an altered query sting. Cookies provide only slightly better results and bring about their own issues.

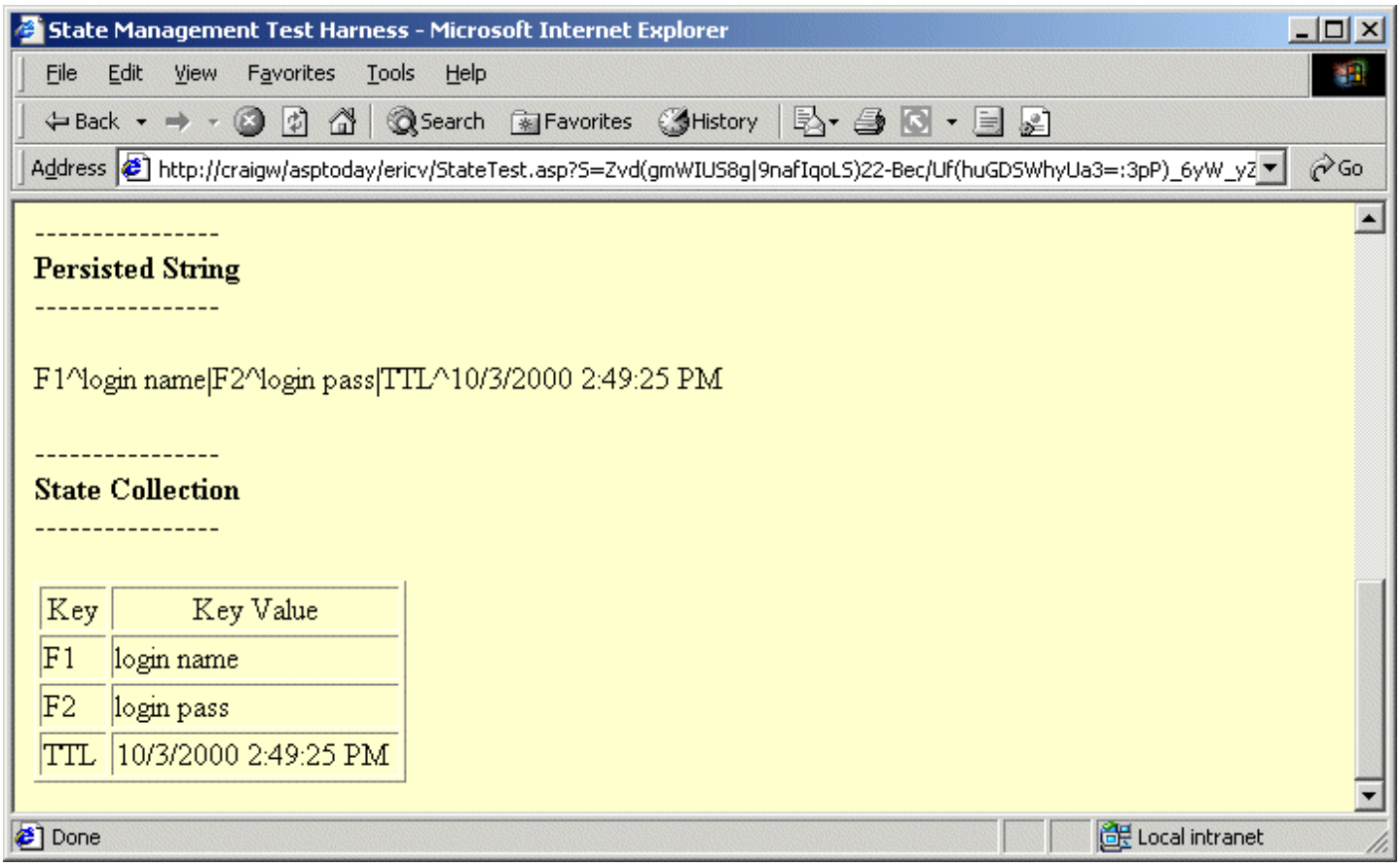
To help reduce the potential risks involved, the persistence processes of the State Management Library encrypt state information providing some measure of security. However, the encryption is meant to only obscure the state information and does not represent a complete security solution. Persistence processes of the State Management Library also embed a checksum within the persisted state to aid in tamper detection. A checksum calculation is performed when state is retrieved and compared to the embedded value. If they are different, an error code is returned to the page program logic to allow respective action. This check is not foolproof, but will detect most tampering attempts.

Application State Lifetime Management

The library provides state lifetime management in a manner similar to the ASP Session timeout method. The persisted information contains a token representing date and time of persistence. State retrieval processes examine the token and determine if the age of the state is older than allowed by the application (default is twenty minutes), returning an error code to the page program logic to act accordingly.

State Management Library Usage

The library is an include file to be referenced within each ASP page of your web application in order to provide the state management capabilities. When employing sessionless state management, it is vital that all pages `include` the library and execute the operations to retrieve state information passed to the page and ensure state is passed to subsequent pages. State integrity is maintained by one page retrieving information passed to it, and in turn, passing the information to pages it references in an unbroken chain. If one page fails to pass state information the chain is broken and state information is lost. I have included some sample `.asp` files for you to try out, as seen below:



`StateManagement.asp` must be copied to a web application folder before it can be included. Including the library in an ASP is illustrated in the following code fragment below. The intent is to maintain state without the intrinsic ASP Session object and the first line includes a directive to disable session management on the IIS web server. This can also be accomplished at the web application scope using the IIS management console.

```
<%@ Language=VBScript EnableSessionState=False%>
<!-- #INCLUDE File="StateManagement.asp" -->
```

The second line instructs the web server to include the contents of `StateManagement.asp` in the current page, making all of the functions and subroutines of the library available for use within that page. This `include` directive assumes that `StateManagement.asp` is present in the root folder of the web application.

Library Implementation Example

Once the library is included, the page can easily implement session management with very little code. Although this example is simple, it implements a broad spectrum of state management operations and tasks.

- State information originating elsewhere in the web application is retrieved and made available in that page.
- State information is accessed and examined to ensure page process requirements were satisfied.
- Items are added to the existing state information for use by other web application processes and pages.
- Application state is persisted ensuring everything is preserved and available for subsequent pages.

```
<%@ Language=VBScript EnableSessionState=False%>
<%
   Option Explicit
   Response.Buffer = True
%>
<!-- #INCLUDE File="StateManagement.asp" -->
<%
   If fnGetState() <> STATE_OK Then
      'State information was not supplied for this page or
      'an error condition occurred. This page requires state
      'to determine if an authenticated user is requesting the
      'page. Just redirect to a default page.
```

```
      CloseState
      Response.Clear
      Response.Redirect "default.asp"
   End If
   'Check the state collection for a key "UID" passed as query string
   'persisted state to validate a logged in user is requesting the page.
   If fnGetStateValue("UID") = "" Then
      'Destroy the state collection and send them back to login.asp
      CloseState
      Response.Clear
      Response.Redirect "Login.asp"
   End If
   'Add items to be included in state from this point forward.
   AddStateItem "KEY1","Test1"
   AddStateItem "KEY2","Test2"
   'Persist the new state information for a URL query string
   'to be passed in hyperlinks.
   strQueryString = fnPersistState(STATE_AS_URL)
   'Destroy the state collection
   CloseState

%>
<HTML>
<HEAD>
<TITLE>Sample State Page 1</TITLE>
</HEAD>
<BODY>

<p><A HREF="sample2.asp<%=strQueryString%>">Link One</a>
<p><A HREF="sample3.asp<%=strQueryString%>">Link Two</a>

</BODY>
</HTML>
```

Retrieving State

The call to function `fnGetState()` initializes state management, retrieves state information passed to the page, and exposes the state information in a collection of Key and Value pairs. The page doesn't need to be concerned how state information was made available to it, only that `fnGetState()` is executed to retrieve and expose state information.

The state retrieval function returns status information about the retrieval process. In the example, the return code was compared against the constant `STATE_OK` to determine that state information was present and available in the state collection. This is one of several return codes for `fnGetState()`. The following example is a more thorough status check:

```
intStateResult = fnGetState()
Select Case intStateResult
   Case STATE_OK
      'State was retrieved and items exist in the state collection.
   Case STATE_NOT_PASSED
      'No state information was found.
   Case STATE_INVALID
      'State information string was malformed.
   Case STATE_CKSUM_ERR
      'Checksum error encountered, something has been altered.
   Case STATE_NOT_PARSED
      'Problem encountered during parsing.
   Case STATE_EXPIRED
      'State information has expired (timed out).
   Case STATE_COLLECTION_ERR
      'Problem encountered while instantiating a Dictionary object.
   Case STATE_NO_ITEMS
      'State was retrieved but no key/value pairs were present
End Select
```

It is up to the page program logic to respond to the various conditions. Program logic in the example was concerned only that state was present. Any status other than `STATE_OK` reflects an error condition for the page and causes a page redirection. Constants used here are defined in the `StateManagement.asp` include file.

Working With the State Collection

All state information is maintained in a collection of keys and values exposed by the `fnGetState()` function. The library provides a full compliment of methods to interact with the State Collection.

Obtaining state information is a simple matter of calling a single function and passing the name of state key of interest.

```
strStateValue = fnGetStateValue("StateKeyName")
```

The variable `strStateValue` will be assigned the value of the state collection key named `StateKeyName`. If the key doesn't exist, `strStateValue` will be assigned `NULL`. When using the returned value, it is important to remember that state values are maintained as variants of subtype String.

State information may be added to the collection or existing key values changed using a single subroutine call:

```
AddStateItem "KEY1","Test1"
```

If the state key `"KEY1"` does not exist in the collection, it will be added with the value containing `"Test1"`. If the key did exist, `"Test1"` will replace the existing value for the key.

State information artifacts may be removed from the collection in a similar manner:

```
RemoveStateItem "KEY1"
```

If the state key `"KEY1"` existed, it would be removed from the collection and no longer persisted as state information.

Query String State Persistence

The persistence function, `fnPersistState()`, does not actually perform the task of persistence, but returns a serialized state string that is to be appended to a hyperlink page resource as the query string.

```
<%
   'Persist the state information as a query string for HREF URLS
   'strQueryString will contain serialized state for the page HREF's
   '
   strQueryString = fnPersistState(STATE_AS_URL)
   CloseState
%>

<HTML>
<HEAD>
<TITLE>Sample State Page 1</TITLE>
</HEAD>
<BODY>

<p><A HREF="sample2.asp<%=strQueryString%>">Link One</a>
<p><A HREF="sample3.asp<%=strQueryString%>">Link Two</a>

</BODY>
</HTML>
```

The persistence function is called once to pass the constant `STATE_AS_URL` and specify the type of persistence structure needed. The value returned is assigned to the variable `strQueryString` and appended to all hyperlink resources in the page.

Hidden Form Field State Persistence

The same persistence function from above is used and differs only in argument. Passing the constant `STATE_AS_FORM` as the argument returns a string representing an HTML hidden text field. The text field contained in the string retains the serialized state as the field value.

```
<HTML>
<HEAD>
<TITLE>Sample State Page 1</TITLE>
</HEAD>
```

```
<BODY>
<FORM method="post" action="formhandler.asp">
   <p><input type="text" name="field1">
   <p><input type="text" name="field2">
   <%
      'Persist the state information as a hidden form field to be passed
      'in a form submit
      '
      Response.Write fnPersistState(STATE_AS_FORM)
      CloseState
   %>
   <p><input type="submit" name="Submit">
</FORM>
</BODY>
</HTML>
```

The `Response.Write fnPersistState(STATE_AS_FORM)` statement outputs the hidden form field in the HTML output stream. If more than one form is present in the page, it is more efficient to call `fnPersistState()` once and assign the return to a variable to be output multiple times.

Cookie State Persistence

In this case, `fnPersistState()` actually performs the persistence operation to a cookie and returns a `NULL` value:

```
<%
   'Persist the state information as a key within a cookie
   'strResult will simply be set NULL.
   '
   strResult = fnPersistState(STATE_AS_COOKIE)
   CloseState
%>
```

The state information exists in a cookie known to the `fnGetState()` state retrieval process. When using cookie persistence, `fnPersistState(STATE_AS_COOKIE)` must be called early and before all HTTP headers have been sent.

Combination State Persistence

Persistence methods may be combined in content consisting of both hyperlinks and forms. Both must pass state to ensure nothing is lost.

```
<%
   'Persist the state information as a query string for HREF URLS
   'strQueryString will contain serialized state for the page HREF's
   '
   strQueryString = fnPersistState(STATE_AS_URL)
   '
   'Persist the state information in another variable as a hidden form
   'field to be passed in a form submit

   strFormField = fnPersistState(STATE_AS_FORM)
   CloseState
%>

<HTML>
<HEAD>
<TITLE>Sample State Page 1</TITLE>
</HEAD>
<BODY>

<p><A HREF="sample2.asp<%=strQueryString%>">Link One</a>
<p><A HREF="sample3.asp<%=strQueryString%>">Link Two</a>
<p>
<FORM method="post" action="formhandler1.asp">
   <p><input type="text" name="field1">
   <p><input type="text" name="field2">
```

```
   <%=strFormField%>
   <p><input type="submit" name="Submit">
</FORM>
<p>
<FORM method="post" action="formhandler2.asp">
   <p><input type="text" name="field3">
   <p><input type="text" name="field4">
   <%=strFormField%>
   <p><input type="submit" name="Submit">
</FORM>
</BODY>
</HTML>
```

In the above example, persistence function is called twice to obtain:

- A query string formatted state
- state formatted as an HTML form field.

Library Configuration

The library uses several constants to affect and control operation. Several of the constant values can be changed to alter aspects of operation including:

- Enable/ Disable encryption of state information
- Enable/ Disable state lifetime management
- Enable/ Disable checksum generation and verification
- Name of the key used for state information
- Name of the cookie used for persistence
- Delimiters used during serialization

The library download includes documentation for the constants.

Usage Guidelines

The library does not limit the number of items that can be maintained and persisted as application state, but this does not suggest that no constraints exist. Aside from physical length constraints of URLs and cookies, keeping state requirements to a minimum is sound application design. Consider the following when implementing state:

- Keep the state requirements to a small number of keys and values. The fewer the better.
- Use short key names of three characters or less. Instead of SessionID use something like SID.
- Key values must be kept short. If state requirements include long text strings, reduce them to a token that will equate to a string in code or some other alternative.
- Query strings are URL Encoded by the persistence function and can result in a much longer query string.
- The library also includes a checksum and age token that increases the size of the persisted state information.
- Even though light encryption is employed, never persist anything that would pose a security or confidentiality risk.
- Use cookie persistence with caution. Cookies have an affinity to the domain from which they were created and state can be lost in load–balanced environments as activity occurs on different servers. The library does not set the cookie domain or path parameters.

If the application must maintain a significant amount of state information, then persist it within a database and maintain only a row key with the library functions. The key can then be used with a query to retrieve the bulk of the state from the database. The majority of the applications I develop do just that using a variant of the State Management Library presented here.

I find it best to disable encryption during development to make troubleshooting and monitoring easier. The library also contains a development utility call named `DumpState()` that will output the contents of the state collection, original persistence string, and configuration values to the HTML stream.

I have used `Response.Redirect` in the examples for compatibility reasons as the means to handle state and page process errors. If the web application is deployed under IIS 5.0 and a recent ASP version, use the `Server.Transfer` method for efficiency as long as the redirection is to a resource on the same server.

Conclusion

The library provides a complete Sessionless State Management System for ASP web applications. The article download contains all of the library source code, further documentation, and sample ASP pages. ASP+ will provide robust and scalable state management capabilities that are long overdue. Until then, we'll have to make do with code solutions like this one or employ other techniques.

Click here to download this article's support material.

Related Articles on ASPToday

Considerations for Scalable and High-Availability Web Applications
Maintaining User State with the Active User Object (AUO)
Serialize Your Session with a Homegrown State Management Component for ASP
ASP session management in a load balanced environment
Serialize Your Session, Part II - Extending the Serializer component
A Summary of the New Features in ASP+

Related Links

Advanced Basics - Working on a Web Farm
Design Strategies for Scalable Active Server Applications
The Lost QueryString: Maintaining State with Active Server Pages
Architecting Your Web Applications
A Blueprint for Building Web Sites Using the Microsoft Windows DNA Platform
The Unofficial Cookie FAQ

Sep 28, 2000

SVG - A New Standard in Web Graphics

By Jon Frost

ASP Tricks

Scripting

XML

enter the discussion

In this article we will explore a new language that is about to revolutionize the Web nearly as much as HTML did in 1992. Scalable Vector Graphics (SVG) is a new open source text–based language for describing two–dimensional graphical objects in XML. SVG will herald a new era in the history of the Web by eliminating layers of production that content originators currently face. In this article we will briefly explore how SVG emerged and quickly move into analyzing some SVG code samples. In future articles I will cover a complete working mapping application.

The Emergence of SVG

SVG was developed by the World Wide Web Consortium (W3C), an open–standards international industry consortium that was formed to lead the Web to its full potential. In 1998 the W3C was presented with two proposals for new graphics formats. Both Precision Graphics Markup Language (PGML) and Vector Markup Language (VML) are XML vector languages that use CSS. The W3C decided to combine the best aspects of both of the PGML and VML languages into a new language called SVG. The W3C envisioned an open–standard extensible language that satisfied Web developers increasing demand for dynamic, scalable, and platform–independent presentation and interaction.

The Power of SVG

There has been a lot of excitement generated by this new graphics format. There are over twenty members of the SVG Working Group including: Sun Microsystems, IBM, Adobe Systems, Macromedia, Hewlett–Packard, Microsoft, and AOL/Netscape. Adobe Systems led the industry in its support of SVG by releasing a downloadable plug–in required for Internet Explorer and Netscape, which supports much of the SVG Recommendation, and by promising full support of SVG in all of their major graphics applications. Sun Microsystems and IBM have also released some significant applications and viewers for SVG. On August 2nd the W3C finally released the SVG Candidate Recommendation and urged developers to begin implementing SVG. Chris Lilley, the Activity leader and chair of the SVG Working Group said that he expects the final W3C SVG Recommendation to come by the end of the year. This combination of industry support and near completion of the SVG Recommendation means that now is a great time for web programmers to begin harnessing the power of SVG.

Several key advantages of SVG are:

- Scalable vector graphics – do not degrade upon panning and zooming which is ideal for charting, mapping, and Web site navigation graphics
- Increased color accuracy – over 16 million colors which means what is seen on the screen will look exactly the same when printed
- Compatibility with XML, HTML4, XHTML as well as conformance to CSS, XSL, and the DOM – this means that SVG is extensible, styleable, scriptable, and integrates easily.
- Smaller file sizes – decreased download times due to the effectiveness of SVG's vocabulary

ASPTODAY Diary

S	M	T	W	T	F	S
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
1	2	3	4	5	6	7
8	9	10	11	12	13	14

Implementations of SVG

SVG's greatest power lies in its ease of use. Just like HTML and XML, SVG is composed of text marked–up by tags which are made up of elements and attributes. There really is nothing new about the structure of the language. The genius of SVG lies in its vocabulary for describing vector graphic shapes, images, and text as well as in SVG's compatibility with the XML 1.0 Recommendation, Namespaces in XML Recommendation, HTML4, and XHTML as well as conformance to the Cascading Style Sheet (CSS) level 2 specification, XSL Transformations (XSLT) Version 1.0, and much of the Document Object Model (DOM) level 2 specification. In the following example we will see the structure of a basic, complete SVG document:

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "–//W3C//DTD SVG 20000802//EN"
   "http://www.w3.org/TR/2000/CR–SVG–20000802/DTD/svg–20000802.dtd">
<svg width="100%" height="100%">
   <desc>Example 1 – One rectangle</desc>
   <rect x="0.75cm" y="0.5cm" width="2.5cm" height="3cm"
   style="fill:purple; stroke:gray; stroke–width:0.1cm" />
</svg>
```

This code creates the following shape:

In the first three lines of Example 1 we see that this SVG document uses the public SVG DTD from the W3C Web Site and is in fact comprised of XML syntax. Line four specifies the height and width of the SVG display area. Theoretically, with SVG we will finally have complete control over the position of every pixel of content that we want to display or print. There are still some formatting and positioning issues that the W3C must address which we will cover at the end of this article. In the next section we are going to get our hands dirty with some coding SVG.

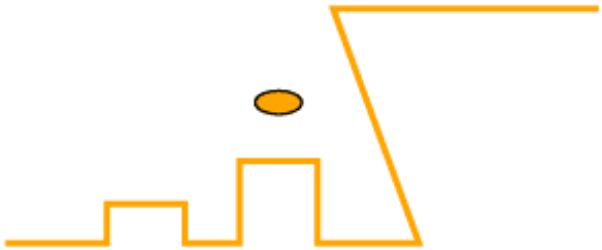
Shapes

As mentioned earlier, SVG offers three graphical objects, which are vector graphic shapes (e.g., paths consisting of straight lines and curves), imported images, and text. Example 1 makes use of the `<rect>` element, which is one of six predefined basic shape elements that are included in the SVG DTD. Rectangles, circles, ellipses, lines, polylines, and polygons make up the basic shape elements that are actually mathematically equivalent and are afforded the same properties as the `<path>` element. In Example 2 we take a look at the grammar involved in describing an ellipse, polygon, and polyline path shapes.

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "–//W3C//DTD SVG 20000802//EN"
  "http://www.w3.org/TR/2000/CR–SVG–20000802/DTD/svg–20000802.dtd">
<svg width="8cm" height="4cm" viewBox="0 0 800 400">
  <desc>Example 2 – One Polyline and one ellipse</desc>
  <ellipse cx="370" cy="175" rx="30" ry="15"
   style="fill:orange; stroke:#000000; stroke–width:3; opacity=25%"/>
```

```
   <polyline style="fill:none; stroke:orange; stroke-width:8"
      points="20,355
          150,355 150,305 250,305 250,355
          320,355 320,250 420,250 420,355
          550,355 440,55 780,55" />
</svg>
```

This code creates the following shape:



Notice that the ellipse element takes the `cx`, `cy`, `rx`, and `ry` attributes that define the x and y axis center point and x and y axis radii while the polyline (and polygon) takes only the `points` attribute. Even with this simple example you can begin to see how inefficient it is to hand–code the `points` element data. For example, imagine how many `points` coordinates would be needed for a decent SVG map. Indeed, SVG will rarely be hand–coded; a slew of graphics applications and coding techniques are currently being developed which will generate 99% of our SVG documents (visit http://www.w3.org/Graphics/SVG/SVG-Implementations for a current list of SVG applications, tools, and viewers). Before we move on to images in SVG I want to point out the `viewBox` attribute of the `svg` element. This `viewBox` takes an `x`, `y`, `width`, and `height` value, which define the dimensions of the SVG document, and depending on the value of the `preserveAspectRatio` attribute can allow for all the content defined within the `svg` element to scale automatically. This trait of SVG is significant in that now our content can truly be platform independent. For instance, my company will not have to create separate documents for our questionnaire tool's charting feature for our users who browse our site on their wireless devices, and for me that means "SVG Rules!"

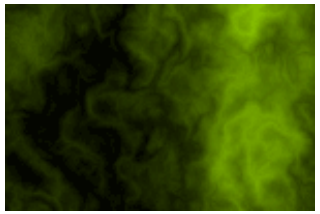
Images, Text, and CSS

In the next example we will find out how raster images fit into the SVG quilt that is unfolding before us. JPEG and PNG raster images excel at displaying lots of colors and therefore will remain important formats for detailed images. The "image" element that you will see in Example 3 is actually used to display the contents of a complete file, which can be a raster image or another SVG document.

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20000802//EN"
   "http://www.w3.org/TR/2000/CR-SVG-20000802/DTD/svg-20000802.dtd">
<svg width="12cm" height="3cm">
   <defs>
      <style type="text/css">
      <![CDATA[
         rect {stroke:orange;stroke-width:1 fill:black}
         .clipstyle
{fill:none;fill-rule:nonzero;stroke:#333333;stroke-width:2;font-family:Verdana,Arial;font-size:80;font-weight:bold}
      ]]>
      </style>
   </defs>
   <desc>Example 3 - One image</desc>
```

```
    <rect x="1" y="1" width="375" height="80" rx="5" ry="5" />
    <g id="clipelement">
    <g>
       <clipPath id="clip1">
       <text x="5" y="70" class="clipstyle">IMAGINE</text>
       </clipPath>
    </g>
    </g>
    <g id="rasterfill">
       <image x="10" y="0" width="180" height="100" xlink:href="imagination.png" style="clip-path:url(#clip1)"
/>
       <image x="190" y="0" width="180" height="100" xlink:href="imagination.png" style="clip-path:url(#clip1)"
/>
  </g>
</svg>
```

The code above takes the file `imagination.png` shown below:



and renders it with SVG text to create the following image:

Notice the `image` element's `x`, `y`, `width`, `height`, and `xlink:href` attributes. The `x`, `y`, `width`, and `height` attributes define the dimensions of the image where `x` and `y` refer to the coordinates of SVG into which the image file is placed. The `xlink:href` attribute is an URI reference to the specified file location value (`imagination.png`). A clipping area has been defined which makes the raster image effectively a background fill effect on the `IMAGINE` text element. And finally, notice that the text element contains a class attribute value of `"clipstyle"` that references the CSS style properties that are internally defined.

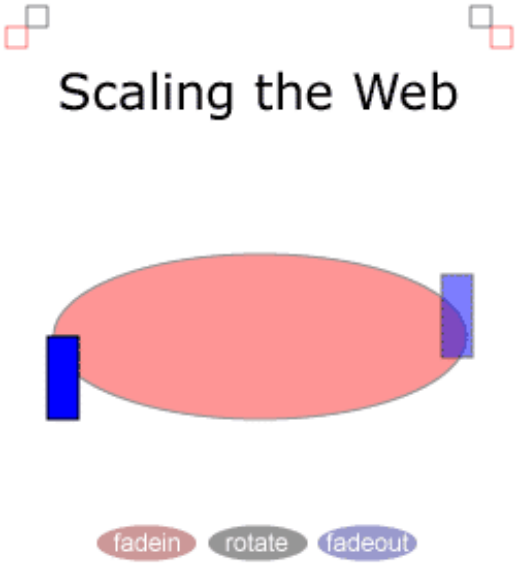
Animation

In this last example we will use much of what we have learned so far and extend this into a whole new realm using SVG's animation vocabulary. I have included only key portions of the `example4.svg` document. Notice that the `"controls"` graphics use `xlink:href` to reference the animation properties and values of the `ellipse` element.

```
<g id="body-content">
 <g id="anim">
   <rect x="20" y="40" width="10" height="10" style="fill: none; opacity:50; stroke: black; stroke-width:0.4;"
/>
   <rect x="235" y="40" width="10" height="10" style="fill: none; opacity:50; stroke: black;
stroke-width:0.4;" />
   <rect x="10" y="50" width="10" height="10" style="fill: none; stroke: red; stroke-width:0.4;" />
```

```
    <rect x="245" y="50" width="10" height="10" style="fill: none; stroke: red; stroke–width:0.4;" />

    <ellipse id="pink" cx="133" cy="200" rx="100" ry="40">
       <animate id="fadein" attributeName="opacity" attributeType="CSS" from="0" to="1"
       begin="indefinite" dur="2s" fill="freeze"/>
    <animate id="rotate" attributeName="rx" attributeType="XML" from="–100" to="100"
       begin="indefinite" dur="2s" fill="freeze"/>
    <animate id="fadeout" attributeName="opacity" attributeType="CSS" from="1" to="0"
       begin="indefinite" dur="2s" fill="freeze"/>
    </ellipse>
  </g>

  . . .

  <g id="controls">
    <a xlink:href="#fadein">
       <ellipse cx="78" cy="300" rx="24" ry="8.5" style="fill: #CC9999"/>
       <text x="62" y="304">fadein</text>
    </a>
    <a xlink:href="#rotate">
       <ellipse cx="132" cy="300" rx="24" ry="8.5" style="fill: #999999"/>
       <text x="116" y="304">rotate</text>
    </a>
    <a xlink:href="#fadeout">
       <ellipse cx="185" cy="300" rx="24" ry="8.5" style="fill: #9999CC"/>
       <text x="165" y="304">fadeout</text>
    </a>
  </g>
    <rect x="221" y="170" width="15" height="40" style="fill: blue; opacity:0.5; stroke: black" />
</g>
```

The above code produces the following image:



If you have installed the Adobe SVG viewer (and set a new MIME type on your web server to image/svg-xml), you can click on this image to view the animation (note that the rotate button won't work due to a bug in the current version of the viewer).

I am still amazed at how small our SVG document is compared to how much use we get out of it. This is due in part to SVG's partial incorporation of the Synchronized Multimedia Integration Language (SMIL) 1.0 Specification and also because of the fact that a concerted effort was given to making the SVG document file sizes as small as possible. SVG authors can make use of filters and symbols while SVG viewers can make use of HTTP 1.1 compression and progressive rendering methods to drastically improve user viewing and interaction time. Visit http://www.w3.org/TR/SVG/minimize.html for the most current information about minimizing SVG file size.

Mathematical graphing, dynamic charting, online demonstrations, interactive mapping, and educational games will become much more prevalent on the Web. And keep in mind SVG's extensibility, which means that we will continue to be astonished by implementations of this new language as the W3C, industry leaders, and developers take extensibility to its extreme.

If you are ready to dive deeper into the world of SVG then I would recommend visiting the W3C Test Suite and SVGElves.com which comprise a fairly comprehensive and growing set of sample implementations of SVG. Another cutting edge SVG sites can be found at BurningPixel, which demonstrates the beauty of SVG's dynamism and interactivity.

The Future of SVG

The SVG 1 Recommendation will be an historic moment in the development of more attractive and interactive user interfaces, but some of us will have to wait until SVG 2+ before our dreams are more fully realized. SVG's adoption depends on a combination of three key things: vision from the W3C and developers, demand from large companies as well as developers, and support from authoring environments and browsers.

With the Release of the SVG 1 Recommendation, Web developers will find content formatting solutions in a combination of XHTML, SVG, CSS2, and XSL. The beauty of extensible languages however is found in the potential ease of implementing new functionality. Currently there are several key issues that are being addressed (e.g., text formatting and viewer caching of SVG content), many of which are expected to be resolved soon. When browsers and authoring environments implement these new features in the SVG 2 Recommendation then Web developers could see another significant change in the "modus operandi" of Web development. There has been talk of the tremendous potential for synergy between SVG and other XML languages. Possibilities include incorporating XFORM, FO–XML (the document and printing standard) and possibly even the emerging VoiceXML standard, but only time will tell.

Conclusion

So far we have looked at a new language called SVG and have explored how to use some of SVG's core elements and attributes. In the next part of this series I hope to cover a practical implementation of SVG, by developing a dynamic mapping application.

Click here to download this article's support material.

RATE THIS ARTICLE

Overall
Poor ○ ○ ○ ○ ○ Excellent
User Level
Beginner ○ ○ ○ ○ ○ Expert
Useful
No! ○ ○ ○ ○ ○ Very
Submit

enter the discussion

Related Articles on ASPToday

Simple, Yet Effective Graphics with VML
Comparison of VML and SGC

Related Links

Adobe SVG Viewer
Jasc Trajectory Pro
SVG Viewers and Editors
SVG 1.0 Specification
SVG Elves
Example of SVG in Action
More Examples of SVG in Action

---

keyword search

Search

Home | Today's Article | Search | Feedback | Write For Us | Suggest an Article | Advertise

Aug 28, 2000

Setting Up a Basic Online Store

By Alik Levin

ASP Tricks

Site Design

enter the discussion

The first thing you probably think when someone mentions the words "internet application…" is 'online store.' Although there are many other kinds of application this is the most common. In addition, it is interesting to develop because it incorporates technologies in all tiers: presentation, business, and data. Using ASP as an integrating technology is easy and even fun when building interactive web sites such as an online store. The problem is that sometimes we just don't know where to start, how to combine all those ASP possibilities into a working web application. This article will demonstrate the basic process of building a simple yet working application that implements the front end of an online store. Using the methodology and code presented here will give you a head start in building your own web apps.

Before your start coding from scratch

Web applications must be thoroughly planned and analyzed (just like any other serious work) prior to writing actual code. A detailed methodology of planning software applications and web ones in particular is out of this article scope, but there are four main steps to take:

- **1.** Write a detailed description of your future application from the specs provided. This description should include target users, provided functionality, knowledge required, available infrastructure, development tools, expected problems and limitations, and possible solutions.
- **2.** When you have it all written down it is easy to visualize the application if you use a flow chart. Draw the chart reading your description. Here you need to identify all the functionality modules and relationships between them in schematic way.
- **3.** Now the data flow should be obvious in your application. Translate it into a site map based on the flow chart. Draw another picture that represents all web pages in your application, their functionality, and relationships between them. A relationship could be a hyperlink, the result of a form submittal, or execution of the `Response.Redirect` method.
- **4.** Design your database on paper. This chart should include all tables required, fields to build up the tables, and relationships between tables. I am not going to argue with those who may say that this step could be (or even should be) taken as second one.

Getting down to business (step by step)

Step 1. We intend to build a web application that implements an online storefront. This store does not require registration. In the store, there are three different departments offering three products types (say, TV-Video, Stereo, and Kitchen Accessories). The user would be able to browse the departments and view product details (e.g. description, manufacturer, price). At any given time, the user would be able to pick any product and place it in his/her shopping cart. The shopping cart may be viewed at any time presenting its current state (products and total cost). The user should be able to remove any particular item from the cart. Once a user decides to place an order, the application should accept user credentials (e.g. first name, last name, credit card authority, credit card number, address to deliver to), associate it with items in the user's shopping cart, and register it in a database. After the order is placed, the shopping cart is emptied. We won't concentrate on form validation to keep the article size down (For an article on form validation, see Advanced Form Validation). The application was created on a Windows 95 platform using MS Personal Web Server, ASP technology, and MS Access for data storage, but can easily be ported to a more robust platform. The application is targeted to a wide Internet audience so there won't be use of DHTML to ensure consistency in functionality no matter what browser is being used.

ASPTODAY Diary

S	M	T	W	T	F	S
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
1	2	3	4	5	6	7
8	9	10	11	12	13	14

Links
Author Page
About ASPToday

Design decisions to face:

The first is that of storing and accessing product information.

The possible solutions are:

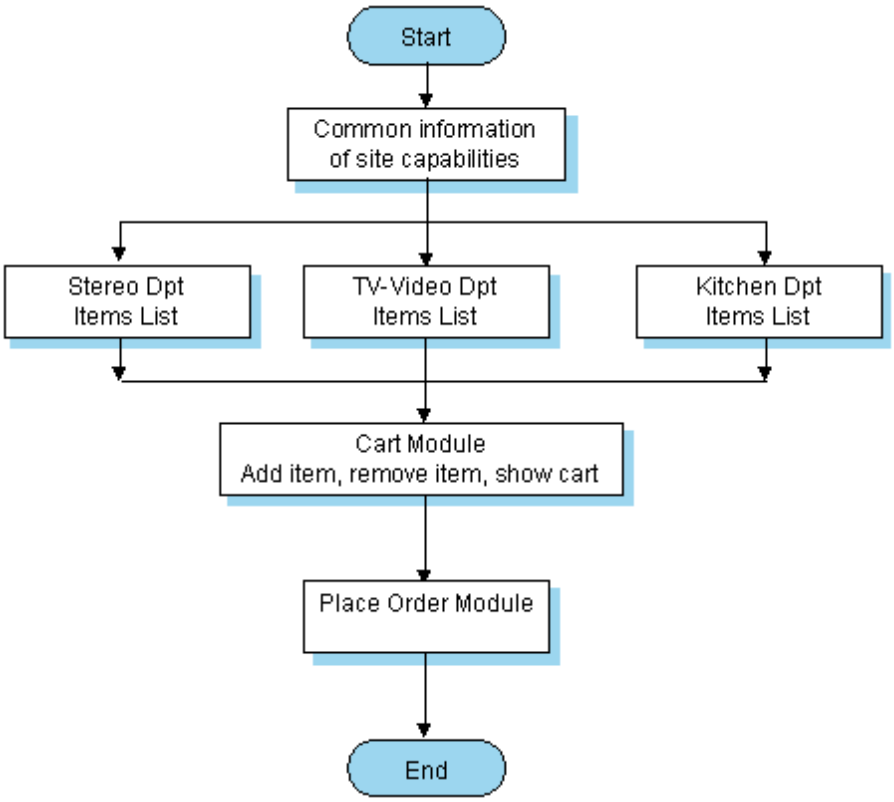
- using static HTML pages
- storing data in files and accessing it through the `FileSystemObject` object; with limited stored data query options
- storing data in an XML format and accessing it through the XML parser
- storing data in database and accessing it through ADO.

In this case we will choose to store data in a database and access it through ADO. With this option we can separate business logic from low level data management, and ADO also has an easy–to–use API to connect to databases and run SQL statements against it.

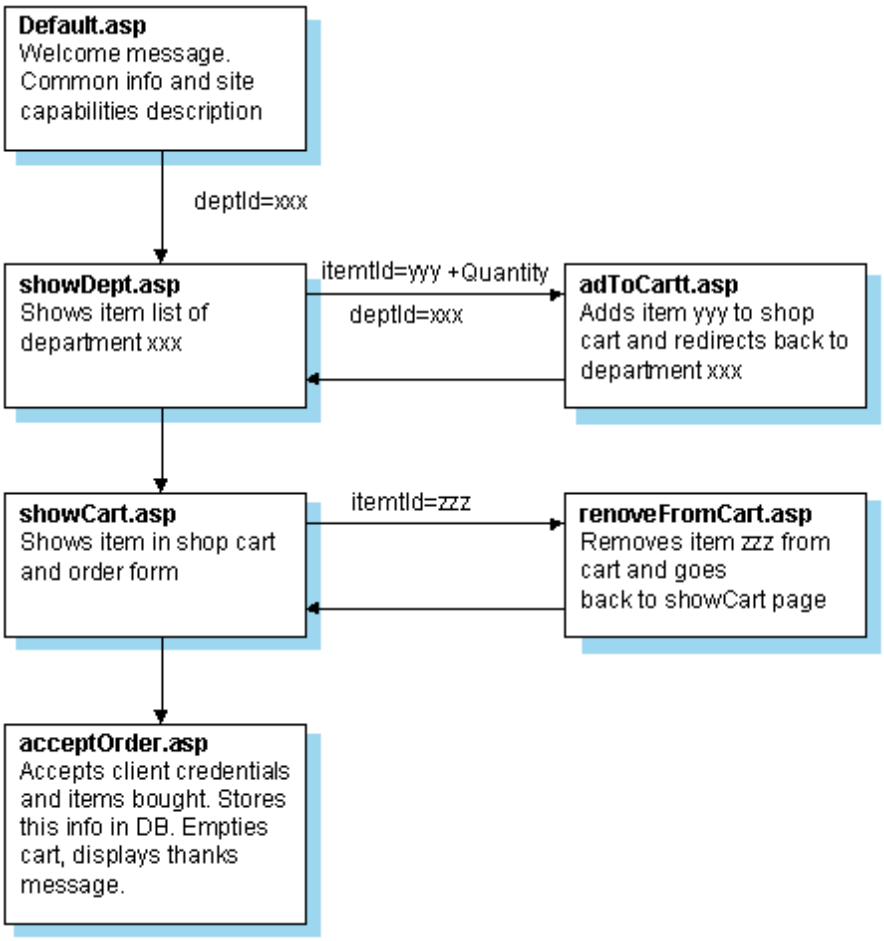
The next decision is that of shopping cart implementation. We could implement it in the client tier (using cookies or static frame), implement it in middle tier (using Session Object or MTS Shared Property Manager), or implement it in data tier (using temporary tables and a hidden field).

The chosen solution is to implement it in client tier using cookies. I believe that if there is the opportunity to do processing on the client, thus freeing server resources, then it should be done.

Step 2. The flow chart:



Step 3.The site map:



Step 4. The database Diagram:

TBL_ITEMS

	Field Name	Field Type	Sample Data
1	ItemId	Number	1845
	ItemDesc	String	TV Set
	Manufacturer	String	SONY
	Price	Currency	$250

TBL_ORDER_DETAILS

	Field Name	Field Type	Sample Data
m	rowNum	Number	1845
	ItemId	Number	2457
m	OrderId	Number	544
	Quantity	Number	3

TBL_ORDERS

	Field Name	Field Type	Sample Data
1	OrderId	Number	844
	OrderDate	date	1.1.2000
	ClientFName	String	JOHN
	ClientLName	String	SMITH
	CCAthority	String	VISA
	CCNumber	Number	1234567893216546
	AddressToDeliver	String	Park Ave., 123

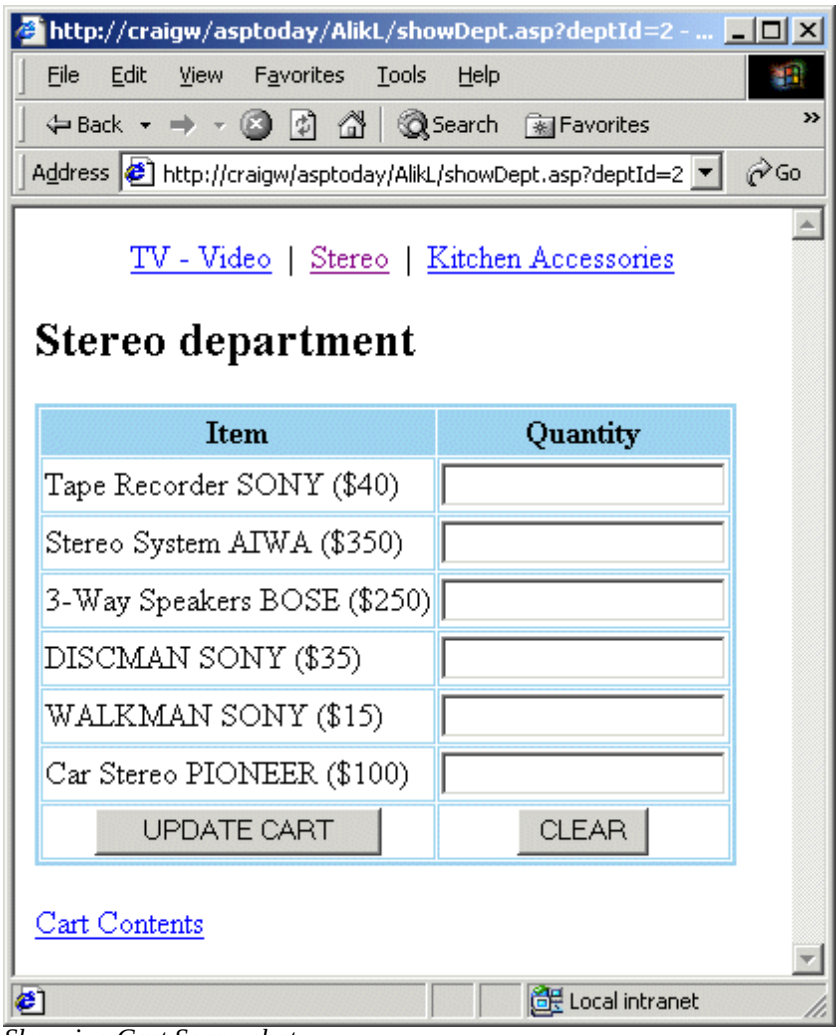
The code

We are now ready to begin coding. It would be nice to set up the database first. For our application, I am using MS Access but the data can easily be ported to SQL Server for a commercial application. After the data tier is set, let's consider the client presentation. Our first page is `Default.asp` which contains common info and links to our three departments: TV – Video, Stereo, and Kitchen Accessories:

Each hyperlink points to the same page, `showDept.asp`, but each has a different query string consisting of a `deptId` variable. This indicates to `showDept.asp` it should list items of that specific department:

```
<A HREF="showDept.asp?deptId=1">TV - Video</A>
 | 
<A HREF="showDept.asp?deptId=2">Stereo</A>
 | 
<A HREF="showDept.asp?deptId=3">Kitchen Accessories</A>
```

The `showDept.asp` page has to complete several tasks. First, it retrieves the `deptId` value sent in the query string to define what department was requested. Second, it builds a SQL statement to pass to the database for retrieval of items. Next, it accesses the database to retrieve records from it according to the SQL statement, and finally it sends retrieved records to our client in a manner the client can view, type in desired quantity, and add to the shopping cart:



Shopping Cart Screenshot

Let's look at the code. First, retrieve requested department ID:

```
dptId = Request.QueryString("deptId")
```

Let's translate the `dptId` value (1,2,3) to the department name to be displayed on the page. It will help our client to navigate easily through our online store:

```
arrDPT =  Array("TV - Video","Stereo","Kitchen")
dptName = arrDPT(dptId - 1)
```

Build the SQL statement:

```
strSQL = "SELECT ITEMID,ITEMDESC,MANUFACTURER,PRICE FROM " _
          &"TBL_ITEMS WHERE DEPTID=" & dptId
```

Create ADO objects and access the database:

```
Set cn = Server.CreateObject("ADODB.Connection")
Set rs = Server.CreateObject("ADODB.Recordset")
cn.Open "Provider=Microsoft.Jet.OLEDB.4.0;" _
&"Persist Security Info=False;Data Source='d:\on_line\onlinestore.mdb'"
rs.Open strSQL,cn
rs.MoveFirst
```

Now build a dynamic table navigating through the Recordset:

```
<FORM action="addToCart.asp" method=POST name=form1>
<TABLE BORDER=2 CELLSPACING=1 CELLPADDING=1>
<TR>
   <TH>Item</TH>
   <TH>Quantity</TH>
</TR>
<%
```

```
   Do Until rs.EOF
%>
<TR>
   <TD><%=rs(1) & " " &rs(2)&"($"&rs(3)&")"%></TD>
   <TD><INPUT name=txt<%=rs(0)%> value=<%=Request.Cookies("txt" & rs(0))%>></TD>
</TR>
<%
   rs.MoveNext
   Loop
%>
<TR>
   <TD><INPUT type="submit" value="UPDATE CART" name=submit></TD>
   <TD><INPUT type="reset" value="CLEAR" name=clear></TD>
</TR>
</TABLE>
<INPUT type="hidden" name= deptIdvalue=<%=dptId%>>
</FORM>
<%
   rs.Close
   cn.Close
   Set cn = Nothing
   Set rs = Nothing
%>
```

I would like us to pay special attention to the highlighted line. In this line we dynamically mark text boxes with the item ID it represents by concatenating some string (here "`txt`") with the item ID retrieved from the database. We are going to use it in the `addToCart.asp` page to store the item picked and the quantity in an appropriate cookie. If a cookie of that name already exists we retrieve it to see what items and at which quantity have already been picked. We also use the `hidden` input type to pass department ID so that `addToCart.asp` will know where to return to after it has finished the cart update.

Shopping cart implementation

I have chosen to use cookies for a number of reasons. They are easy to use and code, and I believe that if there is any opportunity to pass functionality to the client it should be done, freeing resources at the server. They do have their limitations – they can only hold a small amount of data, and are easily changed by the user should they wish to do so for some reason. The shopping cart module includes three pages which each implement their own self–explanatory functionality: `addToCart.asp`, `showCart.asp`, `removeFromCart.asp`.

`addToCart.asp` accepts the form submitted from `showDept.asp`, retrieves the relevant data of item IDs and quantities that customer has chosen, stores this data in a cookie, and redirects the customer back to the same department. At this point for debugging purposes `addToCart.asp` will only show us a table which contains two columns: Item ID and Quantity:

```
<%@ Language=VBScript %>
<HTML>
<HEAD>
</HEAD>
<BODY>
<TABLE BORDER=1 CELLSPACING=1 CELLPADDING=1>
<TR>
   <TH>Item ID</TH>
   <TH>Quantity</TH>
</TR>
```

Navigate through the `Request.Form` collection to retrieve relevant data:

```
<%
   For Each item In Request.Form
```

Exclude values of the submit and reset buttons, and the hidden field:

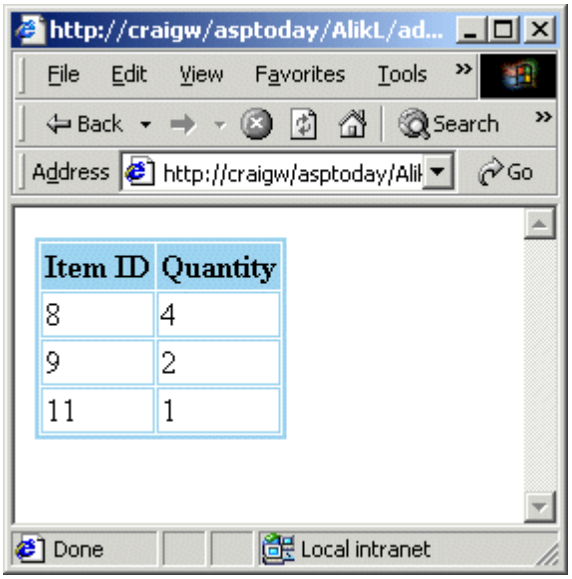
```
  If item <> "submit" And item <> "clear" And item <> "deptId"
```

Then process only those inputs that contain actual data:

```
     If Request.Form(item) <> "" Then
%>
<TR>
```

```
   <TD><%=mid(item,4)%></TD>
   <TD><%=Request.Form(item)%></TD>
</TR>
<%
      End If
   End If
Next
%>
</TABLE>
</BODY>
</HTML>
```

The result of submitting the form that we filled in the Stereo department will look like this:



Let's put it in cookies instead of presenting it in browser, after making sure the data was passed and retrieved in a desirable way. The redesigned and final version of the `addToCart.asp` page will look as follows:

```
<%
   dptID = Request.Form("deptId")
   For Each item In Request.Form
      If item <> "submit" And item <> "clear" And item <> "deptId" Then
         If Request.Form(item) <> "" Then
            Response.Cookies(item) = Request.Form(item)
         End If
      End If
   Next
   Response.Redirect "showDept.asp?deptId="&dptID
%>
```

Now let's provide the ability to view the cart contents by building the `showCart.asp` page. To do so we need to navigate through the `Request.Cookies` collection and retrieve all relevant cookies. Relevant cookies are those that have names starting with `txt` (we could give any other name, like "basket" for example):

```
<TABLE BORDER=1 CELLSPACING=1 CELLPADDING=1>
<TR>
   <TH>Item ID</TH>
   <TH>Quantity</TH>
   <TH>Remove</TH>
</TR>
<%
```

Navigate through Request.Cookies collection:

```
  For Each item In Request.Cookies
```

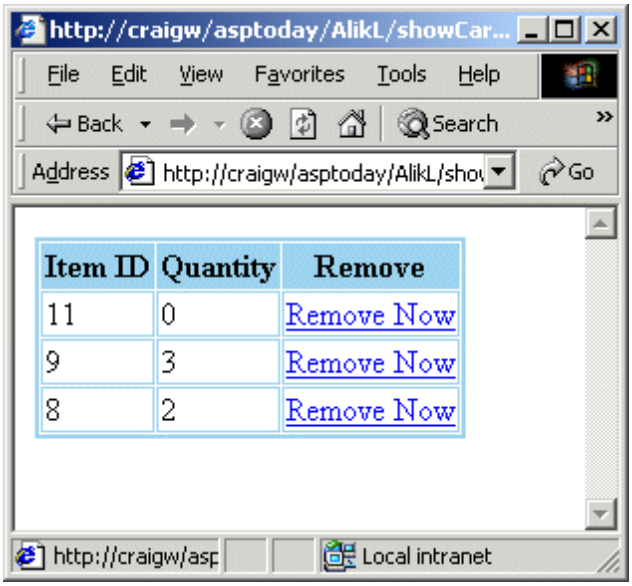
Recognize relevant cookie:

```
  If Left(item,3) = "txt"
```

Then retrieve the actual item ID:

```
  itemId=Mid(item,4)%><TR>
   <TD><%=itemId%></TD>
   <TD><%=Request.Cookies(item)%></TD><TD><A
HREF="removeFromCart.asp?cook=<%=item%>">Remove Now</A></TD>
</TR>
<%
End If
Next
%>
</TABLE>
```

As you can see, each row in the table is built up of the item ID, its quantity, and a hyperlink to the `removeFromCart.asp` page which passes the cookie name (txt15, for example) in a query string to remove it from the shopping cart:



Works fine, but item ID won't say much to our customer. It does link however to the TBL_ITEMS table which contains all the item information we need. So in order to present to the customer a comprehensible representation of their shopping cart we need to access the database to retrieve the information about items in their cart. Let's redesign `showCart.asp`. First we build the SQL statement to retrieve the data:

```
<%
   strSQL = "SELECT * FROM TBL_ITEMS WHERE ITEM_ID IN("
   For Each item In Request.Cookies
      If Left(item,3) = "txt" Then
         itemId = Mid(item,4)
         strSQL = strSQL & itemId & ","
      End If
   Next
   strSQL = strSQL & "-1)"
%>
```

When the SQL statement is ready to run we create the ADO objects and retrieve data into the Recordset Object, and then just show it to a client navigating through the records:

```
<%
   Set cn = Server.CreateObject("ADODB.Connection")
   Set rs = Server.CreateObject("ADODB.Recordset")
   cn.Open "Provider=Microsoft.Jet.OLEDB.4.0;" _
       &"Persist Security Info=False;Data Source='d:\on_line\onlinestore.mdb'"
   rs.Open strSQL,cn
```

It could be very helpful to check if there is anything in the cart, if not display an appropriate message to client:

```
  If rs.BOF And rs.EOF Then
%>
<H1>Your Cart Is Empty</H1>
```

```
<%
      rs.Close
      cn.Close
      Set cn = Nothing
      Set rs = Nothing
      Response.End
   End If
```

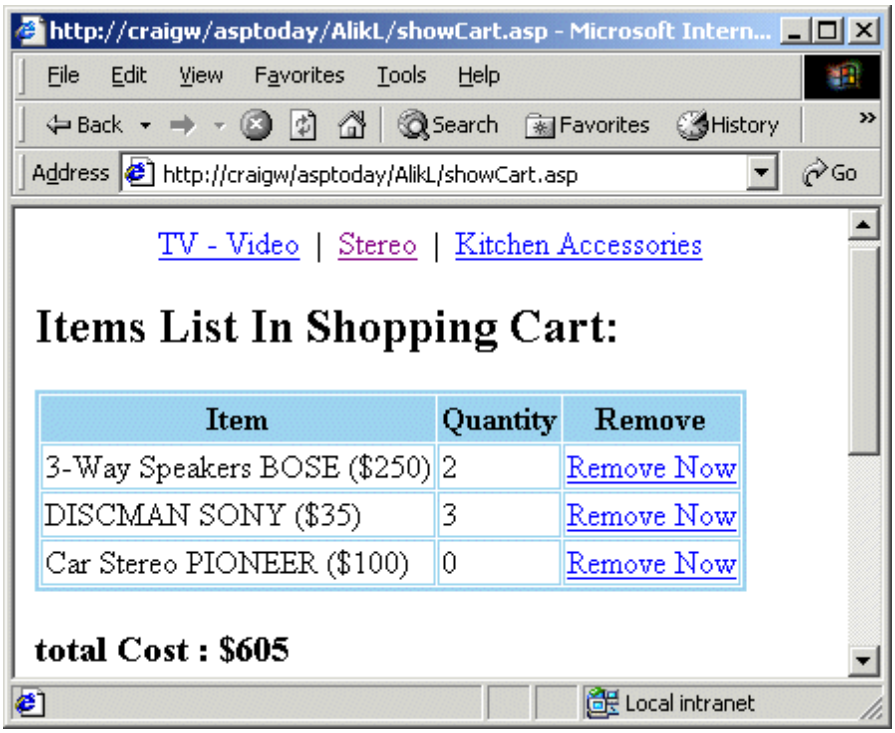
If the cart is not empty, show its contents:

```
  rs.MoveFirst
%>
<H2>Items List In Shopping Cart:</H2>
<TABLE BORDER=1 CELLSPACING=1 CELLPADDING=1>
<TR>
   <TH>Item</TH>
   <TH>Quantity</TH>
   <TH>Remove</TH>
</TR>
<%
   Do Until rs.EOF
      item = "txt" & rs(0)
      quantity = Request.Cookies(item)
      totalCost = totalCost + rs(3) * quantity
%>
<TR>
   <TD><% = rs(1) & " " & rs(2) & " ($" & rs(3) & ")"%></TD>
   <TD><% = quantity%></TD>
```

Providing a hyperlink to `removeFromCart.asp` makes it possible to remove specific items from the cart. This hyperlink should also pass the name of the item to be removed using a query string:

```
  <TD>
   <A \HREF="removeFromCart.asp?cook=<%=item%>">RemoveNow</A>
   </TD>
</TR>
<%
      rs.MoveNext
   Loop
   rs.Close cn.Close Set
   cn =  Nothing Set
   rs = Nothing
%>
</TABLE>
<H3>total Cost :$<%=totalCost%></H3>
```

The new look of our shopping cart is much nicer than before:



The last thing to do in the shopping cart module is to build a `removeFromCart.asp` page. This page accepts a cookie name representing the item to be removed from the shopping cart. All we need to do is retrieve this name from query string, set the expiration date to a time in the past for this cookie (which deletes the cookie), and return to the shopping cart:

```
<%
   cookName = Request.QueryString("cook")
   Response.Cookies(cookName).Expires = Now()-1000
   Response.Redirect "showCart.asp"
%>
```

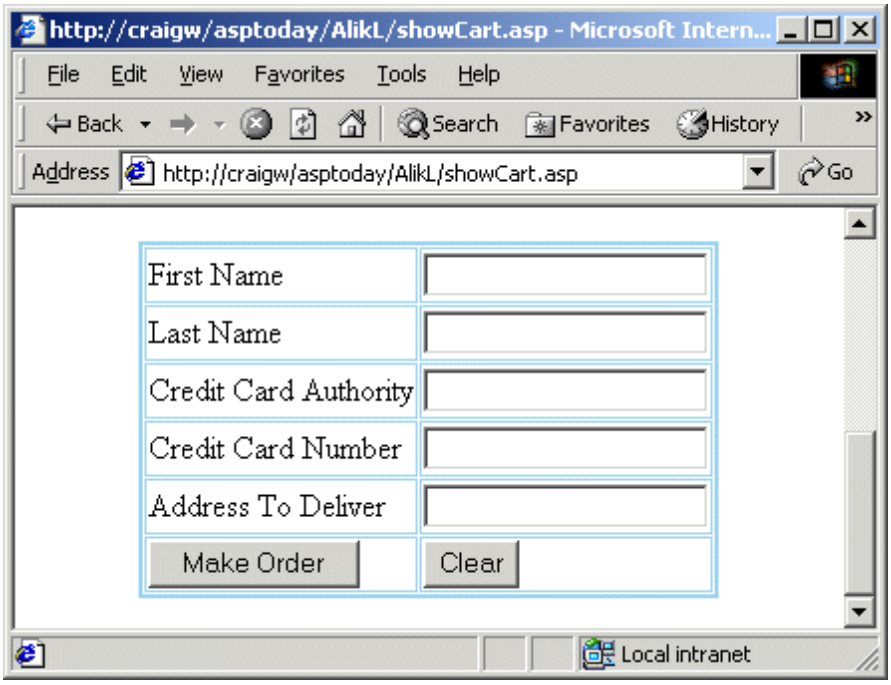
Accepting the order

One last thing left to do is to accept the order. For that purpose, we are going to provide a form in `showCart.asp` to accept the client data. This form will be processed by the `acceptOrder.asp` page. Here is the additional code for `showCart.asp` that displays the order form:

```
<P align=center>
<FORM action="acceptOrder.asp" method=POST name=form1>
<TABLE BORDER=1 CELLSPACING=1 CELLPADDING=1>
<TR>
   <TD>First Name</TD>
   <TD><INPUT name=txtFName></TD>
</TR>
<TR>
   <TD>Last Name</TD>
   <TD><INPUT name=txtLName></TD>
</TR>
<TR>
   <TD>CreditCard Authority</TD>
   <TD><INPUT name=txtCCAuth>
   </TD>
</TR>
<TR>
   <TD>Credit CardNumber</TD>
   <TD><INPUT name=txtCCNum></TD>
</TR>
<TR>
   <TD>AddressTo Deliver</TD>
   <TD><INPUT name=txtAddr></TD>
</TR>
```

```
<TR>
   <TD><INPUT type="submit" value = "Make Order"></TD>
   <TD><INPUT type="reset" value="Clear"></TD>
</TR>
</TABLE>
</FORM>
</P>
```

The code above results in the following presentation:



The `acceptOrder.asp` page accepts the data sent by the form and stores it in the `TBL_ORDERS` table, assigning a unique ID to the `ORDERID` field of the new record. After that, it stores all shopping cart items in `TBL_ORDER_DETAILS` assigning to the `ORDERID` field the very same unique ID. The question is, where do we get this unique ID? The simplest answer seems to be to manage a single row, single field table that will store this unique ID. Every time that we will need a new unique value, we'll just update the current value by increasing it by one. The following is the `acceptOrder.asp` page:

Retrieve Form data submitted by user:

```
<%
   fn = Request.Form("txtFName")
   ln = Request.Form("txtLName")
   cca = Request.Form("txtCCAuth")
   ccn = Request.Form("txtCCNum")
   addr = Request.Form("txtAddr")
```

Build SQL statement to retrieve unique ID for the order:

```
  strSQLUniqueValue = "SELECT * FROM TBL_COUNTER"
```

Create ADO objects, retrieve unique ID, and increase it by one:

```
  Set cn = Server.CreateObject("ADODB.Connection")
   Set rs = Server.CreateObject("ADODB.Recordset")
   cn.Open "Provider=Microsoft.Jet.OLEDB.4.0;Persist Security Info=False;" & _
      "Data Source='d:\on_line\onlinestore.mdb'"
```

Don't forget to set the recordset lock type property to 2 (pessimistic lock type) so it will be updateable:

```
  rs.Open strSQLUniqueValue,cn,,2
   thisOrderNumber = rs(0)
   rs(0) = thisOrderNumber+1
   rs.Update
   rs.Close
   Set rs = Nothing
```

Build SQL statement to make new record in TBL_ORDERS table and execute it:

```
  strSQLNewOrder = "INSERT INTO TBL_ORDERS " _
      &"(ORDERID,ORDERDATE,CLIENTFNAME," _
      &"CLIENTLNAME,CCAUTHORITY,CCNUMBER,ADDRESSTODELIVER)" _
      &"VALUES(" & thisOrderNumber & ",#" & Now() & "#,'"_
      & fn & "','" & ln & "','" & cca & "'," & ccn & ",'" & addr & "')"
   cn.Execute strSQLNewOrder
```

Now, navigate through the Request.Cookies collection using a For Each loop. Inside the loop build a SQL statement for each item in the shopping cart and execute it to insert it into the TBL_ORDER_DETAILS table:

```
  For Each item In Request.Cookies
      If Left(item,3) = "txt" Then
         itemId = Mid(item,4)
         quantity = Request.Cookies(item)
         strSQLOrderDetails="INSERT INTO TBL_ORDER_DETAILS " _
            &"(ITEMID,ORDERID,QUANTITY)VALUES(" _
            & itemId & "," & thisOrderNumber & "," & quantity & ")"
         cn.Execute strSQLOrderDetails
```

Empty each item from the cart after updating the database:

```
        Response.Cookies(item).Expires = Now()-1000
      End If
   Next
   cn.Close
   Set cn = Nothing
%>
```

Display a friendly message to client:

```
<HTML>
<HEAD></HEAD>
<BODY>
<H2 align = center> Your Order Was  Successfully Accepted. Order  Number is:
<%=thisOrderNumber%></H2>
</BODY></HTML>
```

Conclusion

I have covered a simplified development process for a data driven web commerce application utilizing all three tiers: presentation, business, and data. In addition, we've seen one of the many possible realizations of a shopping cart based on cookies. It has its pros: it is easy to implement, it utilizes client resources, and it is fast. It also has its cons: the browser accepts a limited number of cookies or it may not accept cookies at all. This can be checked by ASP code and if it does not we can display a friendly message like: "Our site requires cookies to be enabled. Please enable them, and you will be able to enjoy our huge discounts…" There are still a lot things to take care of like form validation, credit card verification, preventing duplicated orders, providing transactional support, security, etc. that we haven't covered here – it is not possible to learn how to build an entire commercial application from just one article! This sample application should be taken more as a starting point.

Click here to download this article's support material.

Related Articles on ASPToday

Double Authentication
Advanced Form Validation
Workflow Application Encryption with Certificate Server - Part I
Creating a Shopping Cart ASP Component
An Online Shopping Cart WAP Application using WML and ASP
Creating a Simple Shopping Cart

Related Links

Example Cart
Another Example Cart
Web Commerce Methodology

---

keyword search

Search

Home | Today's Article | Search | Feedback | Write For Us | Suggest an Article | Advertise

Jun 12, 2000

HTML Template Processing using VBScript with ASP

By Sharat Devnoor

ASP Tricks

enter the discussion

When I started developing web applications using Active Server Pages, the intermingling of script with HTML in ASP pages always troubled me. There was no clean separation of user-interface from the application logic, making it hard to separate the process of designing the application's user interface from writing, testing, and debugging its code. Also, when editing I had to contend with two syntaxes at the same time, VBScript and HTML, and I was unable to use Template expansion constructs (also known as AutoComplete Features) and syntax coloring features of my favorite editor – reason enough to look for an alternative approach.

Solving the Problem

Then I learned about WebClasses, and their ability to achieve separation of code and HTML through the use of HTML Template files. Since some ISPs do not allow the use of Webclasses or any COM components for ASP, I developed a couple of VBScript routines to do my own Template Processing. I decided to use VBScript because I wanted a solution that would be easier to use by any ASP programmer, in any environment capable of supporting ASP. Although I developed this solution for use in ASP pages, it can be used to process files of any content, and, with some care, binary files.

Template Processing Model

In order to make things clearer, here are the definitions of some key concepts in this template processing model:

HTML Template File — This is an HTML page designed to act as a template. Templates differ from regular HTML pages only in that they often contain named replacement areas (the area inside the `<wc@..>` tags) that are replaced with actual ASP content before sending the page to the browser, allowing you to customize your response. In the example below, I have created my own custom tags and any data they call will go between them:

```
<HTML>
<HEAD>
<TITLE>A Sample Template File</TITLE>
</HEAD>
<BODY>
<H1>Current Time is <wc@CurrentTime>Sample Data</wc@CurrentTime></H1>
</BODY>
</HTML>
```

The output of the above template file would look like this:

Custom tags — Note the `<wc@CurrentTime>` tag. This is a user-defined tag identified by tag-name `CurrentTime` that I created in order to tell my routine that I want to insert the current time in this area of the page. The tag can have any name you wish, as long as you have the `wc@` prefix. You can use any other prefix if you want, provided you use the same prefix for all the tags. All the tags require a corresponding closing tag with an optional tag-content enclosed between them. Both tags and the tag-contents are replaced at run-time with custom content provided by you. If you want to perform the same replacement in several locations, use the same tag-name.

Template-data — The process of performing text replacements involves scanning a template file for tags and replacing them and their contents with custom content. Since it is very time consuming to scan the template file every time you need to replace the tags with custom

ADSI/CDO (13)
ASP Tricks (93)
ASP+ (12)
BackOffice (32)
Components (74)
Data Access (126)
Miscellaneous (35)
Non-MS ASP (10)
Scripting (82)
Security/Admin (44)
Site Design (42)
Site server (13)
XML (63)

free email updates

email address

Register

ASPTODAY Diary

S	M	T	W	T	F	S
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
1	2	3	4	5	6	7
8	9	10	11	12	13	14

Links
Author Page
About ASPToday

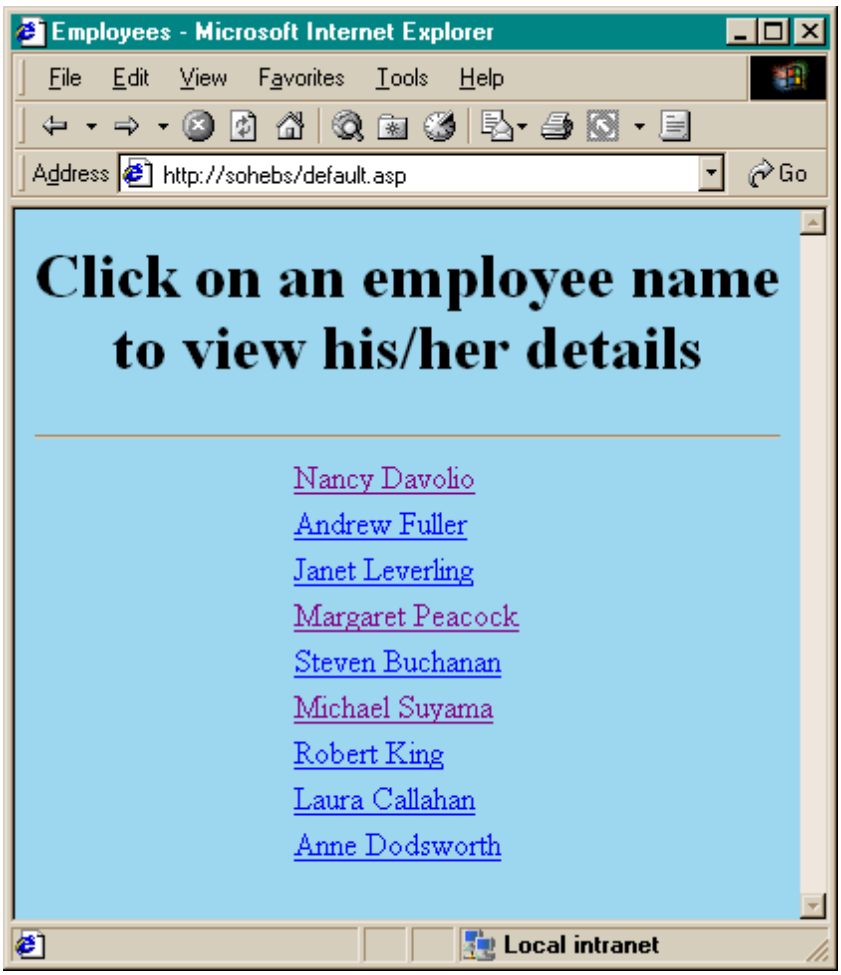
content, I decided to speed up the process by storing the information about the tags (name, exact location and length of the tag) in an array that I call "`template-data` ".

Tag-dictionary — This Dictionary object contains all the tags and tag-contents, with tag-prefix + tag-name used as the key. This object is used to assign custom content to the tags.

Now that we have defined the terms, it's time to discuss the template processing model in detail, which consists of three stages:

- Parsing — the HTML template file is parsed to create `template-data`. This `template-data` is then cached in an application object. This is done once only for each template file.
- Value Mapping — the actual content is mapped to their respective tags making use of the tag-dictionary object.
- Formatting — In this stage the tags are replaced with actual content and the template is written to the output making use of the `Response` object.

Now let's discuss each one of them in detail. All code is included in the download at the end of this article. Our application uses the Northwind sample database. Be sure to read the `read-me` file, and make the appropriate changes for your machine. `Default.asp` displays a list of the employees of Northwind – by clicking on any name, the details of that employee is displayed in a table that results from our template processing.



Parsing

The `ParseTemplate` function performs the bulk of the template processing during the parsing stage. This function accepts the full path to the template file and the tag-prefix used in that file. If an empty string is provided for the tag-prefix, `wc@` is used as a default. This function scans the file, looking for user-defined tags prefixed with the given tag-prefix. It then populates `template-data` with the information about all the tags that it locates, and returns this data to its caller. This function does not support nested tags and expects a closing tag to accompany each tag.

```
Function ParseTemplate(ByVal sFileName, ByVal sTagPrefix)

   ' parses the given "sFileName" and returns the TemplateData
   ' for that file.
   '
   ' sFileName  - required. name of the template file.
```

```
' sTagPrefix - required. the tag prefix used in the template file
'              default is "wc@"

On Error Resume Next

Dim sTemplate
sTemplate = ReadTemplate_(sFileName)
If Err.Number <> 0 Then Exit Function

If Len(sTagPrefix) = 0 Then sTagPrefix = "wc@"

Dim vaData, nDataItems
ReDim vaData(1, 10)
nDataItems = 0
InsertData_ vaData, nDataItems, sFileName, Null

Dim vaTags, nTagItems
ReDim vaTags(1, 10)
nTagItems = 0

Dim nStart, nPos, sTagName, sTagContents, nTagStart
nStart = 1

Do
   nPos = InStr(nStart, sTemplate, "<" & sTagPrefix, vbTextCompare)
   If nPos = 0 Then Exit Do

   ' insert this block data into "vaData"
   InsertData_ vaData, nDataItems, OP_READ, CLng(nPos - nStart)
   nTagStart = nPos

   Do
      nStart = nPos + 1
      nPos = InStr(nStart, sTemplate, ">", vbTextCompare)
      If nPos = 0 Then Exit Do

      sTagName = Mid(sTemplate, nStart, nPos - nStart)
      nStart = nPos + 1

      ' now search for the closing tag-prefix
      nPos = InStr(nStart, sTemplate, "</" & sTagName & ">", _
                   vbTextCompare)
      If nPos = 0 Then Exit Do

      sTagContents = Mid(sTemplate, nStart, nPos - nStart)
      nStart = nPos + Len(sTagName) + 3 ' 3 = Len("</>")

      ' insert this TagName and TagContents into vaTags & vaData.
      If Len(sTagName) > 0 Then
        InsertData_ vaTags, nTagItems, Trim(sTagName), Trim(sTagContents)
        InsertData_ vaData, nDataItems, OP_TAG, Trim(sTagName)
      End If

      Exit Do
   Loop

   InsertData_ vaData, nDataItems, OP_SKIP, CLng(nStart - nTagStart)
Loop

' insert the remaining block.
InsertData_ vaData, nDataItems, OP_READ, _
                                      CLng(Len(sTemplate) - nStart + 1)
```

```
   ' pack all the information into vaData
   ReDim Preserve vaData(1, nDataItems - 1)
   If nTagItems > 0 Then
      ReDim Preserve vaTags(1, nTagItems - 1)
      vaData(1, 0) = vaTags
   End If

   ParseTemplate = vaData

End Function
```

Although `ParseTemplate` does the bulk of the work, you would never use it directly as it would cause the page to be parsed every time the page was accessed. Instead we use the `InitTemplateData` function, which is called in `empdat.asp`.

```
<% Option Explicit %>
<!-- #include file="includes\tpm_decl.asp" -->
<!-- #include file="includes\tpm_init.asp" -->

<%
On Error Resume Next

Call InitTemplateData("emp_details", Server.Mappath("empdat_pi.htm"), "wc@")
If Err.Number <> 0 Then
   Err.Raise 999, "Unable to process employee details template"
End If
%>
```

This function, defined in `tpm_init.asp`, checks to see if the template data has already been created for a particular page. If not, it uses the `ParseTemplate` function to get the template data, and stores it in the Application object using a key supplied by you:

```
Function InitTemplateData(ByVal sAppKey, ByVal sFileName, ByVal sTagPrefix)

   ' this function tries to get the template-data from the
   ' application object with "sFileName" as key, if found
   ' it returns the template-data. if not found then this
   ' function parses the given template file, stores the
   ' template-data in application with "sFileName" as key,
   ' and returns the template-data.
   '
   ' sFileName  - required. name of the template file.
   ' sTagPrefix - required. the tag prefix used in the
   '              template file, default is "wc@"

   On Error Resume Next

   Dim vTemplateData
   If Not IsArray(Application(sAppKey)) Then
      vTemplateData = ParseTemplate(sFileName, sTagPrefix)
      If Err.Number <> 0 Then Exit Function

      Application.Lock
      If Not IsArray(Application(sAppKey)) Then
         Application(sAppKey) = vTemplateData
      End If
      Application.UnLock
   Else
      vTemplateData = Application(sAppKey)
   End If

   InitTemplateData = vTemplateData

End Function
```

You can initialize the template data in `Application_OnStart` event handler in `Global.asa`, so that the parsing is done only once for each template, resulting in increased speed while processing the templates. You also get the added benefit of template data being always present when you need it:

```
Sub Application_OnStart()

   Dim sInitErr: sInitErr = ""

   Dim vTemplateData
   vTemplateData = InitTemplateData("emp_details", _
                                    Server.Mappath("empdat_pi.htm"), "wc@")
   If Not IsArray(vTemplateData) then
      sInitErr = "Error processing employee detail template file"
   End If

   If Len(sInitErr) > 0 Then
      Application("OnStart_Errors") = sInitErr
   End If
End Sub
```

Value Mapping

The tag dictionary object plays a very important role in this stage. You first get the tag dictionary object from a template data by using `GetTagDictionary` function. Then you use this object to supply the actual content that will replace the tags using the appropriate tag name as the key (see `empdet.asp`):

```
<%
...

' Get the template-data
Dim vTemplatedata
vTemplatedata = Application("emp_details")

' Get the Tag-Dictionary for the template-data
Dim ObjTagDict
Set ObjTagDict = GetTagDictionary(vTemplatedata)

' Supply the actual content for the tags
objTagDict("wc@FirstName") = objRs("FirstName").Value
objTagDict("wc@LastName") = objRs("LastName").Value
objTagDict("wc@TitleOfCourtesy") = objRs("TitleOfCourtesy").Value
objTagDict("wc@BirthDate") = objRs("BirthDate").Value
objTagDict("wc@HomePhone") = objRs("HomePhone").Value
objTagDict("wc@Address") = Replace(objRs("Address").Value, Chr(13) _
                           & Chr(10), "<br>")
objTagDict("wc@City") = objRs("City").Value
objTagDict("wc@Region") = objRs("Region").Value
objTagDict("wc@PostalCode") = objRs("PostalCode").Value
objTagDict("wc@Country") = objRs("Country").Value
objTagDict("wc@Notes") = objRs("Notes").Value
%>
```

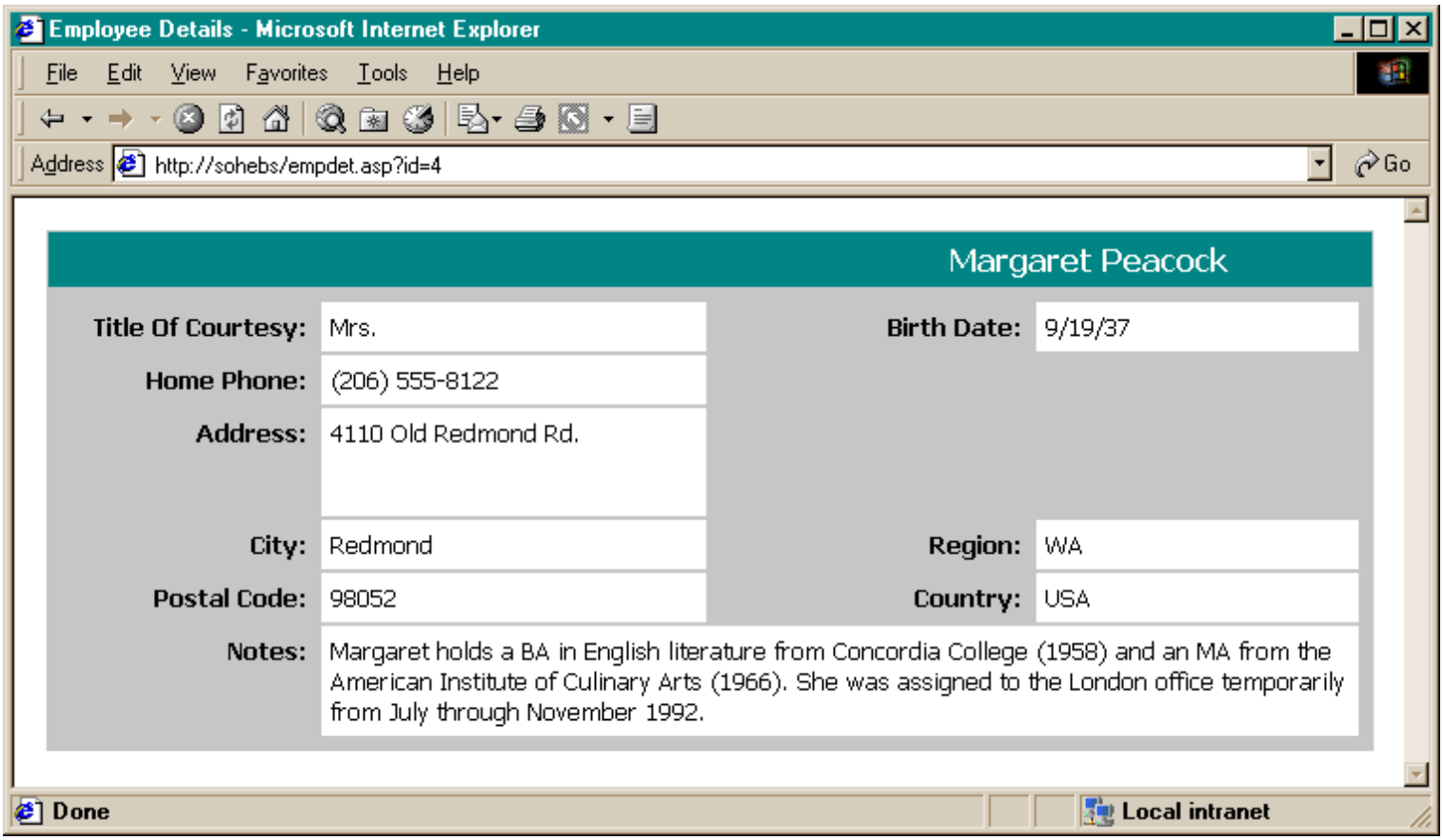
Formatting

The `WriteTemplate` method plays the central role in this stage of template processing. This method accepts two parameters; the template data and the tag dictionary with the actual content assigned to the respective tags. This method then reads the template file, uses the template data to find and replaces all the tags with the actual content from tag dictionary and writes the resulting content to the output stream through the `Response` object. This process is very fast as the template data contains the exact position and length of all the tags in the template.

```
<%
' write the template to the output
Call WriteTemplate(vTemplateData, objTagDict)
%>
```

Since `WriteTemplate` accepts only one template data and one tag dictionary object you might ask – what if I want a tag in a template file to be replaced by the contents of another template file? For example, say that I have a template file in which the `<wc@HeaderFile>` tag is to be replaced by the contents of header file, which is itself a template file. It is possible to do this, albeit in a round about fashion. The code below shows how to solve the problem, with comments explaining the details:

```
<%
Dim vTemplateData, vHeaderTemplateData
vTemplateData = Application("Template_file")
vHeaderTemplateData = Application("Header_file")

Dim objTagDict, objTagDictForHeader
Set objTagDict = GetTagDictionary(vTemplateData)
Set objTagDictForHeader = GetTagDictionary(vHeaderTemplateData)

' now we want the "wc@HeaderFile" to be replaced by the contents
' of the header file, to achieve that first map all the
' actual content for header file through 'objTagDictForHeader'

objTagDictForHeader("wc@tagA") = "ABC…"
..
..

' now assign the contents of the header file to "wc@HeaderFile" in
' template_file by assigning an array containing the template-data
' and tag-dictionary for header file to the tag-dictionary of
' template_file. This is to done in order to overcome the limitations
' of WriteTemplate function.

objTagDict1("wc@HeaderFile") = Array(vHeaderTemplateData, objTagDictForHeader)

' now write the template_file to the output
call WriteTemplate(vtempalteData, objTagDict)
%>
```



General Considerations when using HTML

- Run your HTML files through an HTML syntax checker before adding them to your application. If you add a template that contains badly formatted HTML, you may see errors in the processed output.
- Use relative URLs to images and related files. Your application and its HTML pages can be deployed onto a Web server under a different parent directory than the one on the development computer. Because of this, it is best to use relative URLs in your HTML pages rather than absolute URLs.
- Use dynamically generated URLs whenever possible to move to other pages, rather than typing a manual URL into your templates or code.

Conclusion

The template processing library has proven to be very useful for my ASP programming, resulting in much cleaner ASP pages. I hope that you will find it as useful as I do – happy programming!

Click here to download this article's support material.

Related Articles on ASPToday

Writing Database content to Static HTML Pages with ASP and Javascript
Pictures at an Exhibition: Template–based Integration of Graphics and Back–End Code

---

Aug 25, 1999

Using Active Server Pages to Build Microsoft Word Documents

By Gardiner B. Jones

ASP Tricks

Data Access

enter the discussion

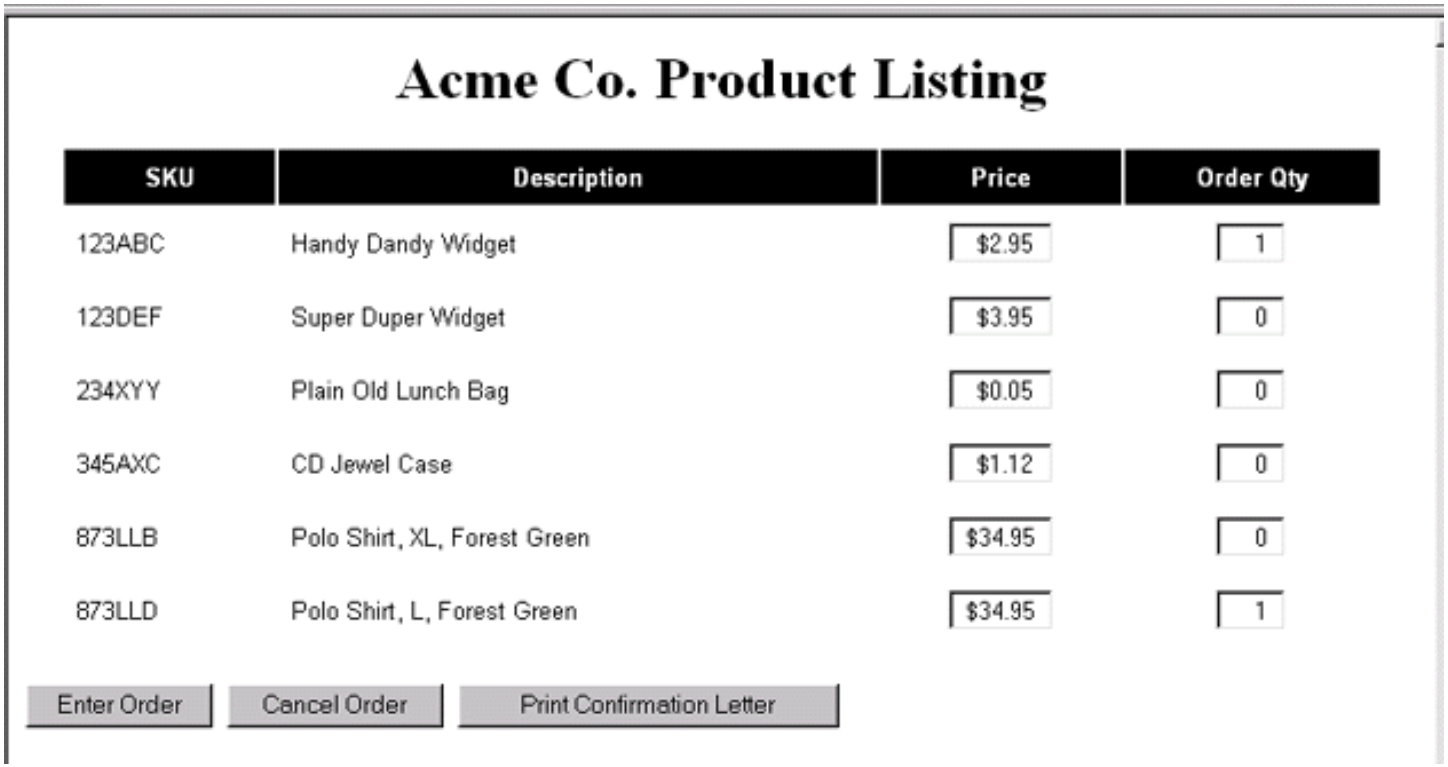
Background

`BuildDoc.asp` is an Active Server Page (ASP) that reads the output of a Web page form, and creates as output a Microsoft Word document containing a table of changed data within the form. Forms are no longer limited to containing static information. With database connectivity, the increasing use of Dynamic HTML (DHTML), and the growing interest in XML, it has become common practice in business Web pages for the data contained in them to be dynamic. That is, what is shown in the form may change based on user interaction (see the sample input form below).

ASPTODAY Diary

S	M	T	W	T	F	S
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
1	2	3	4	5	6	7
8	9	10	11	12	13	14

Links
Author Page
About ASPToday



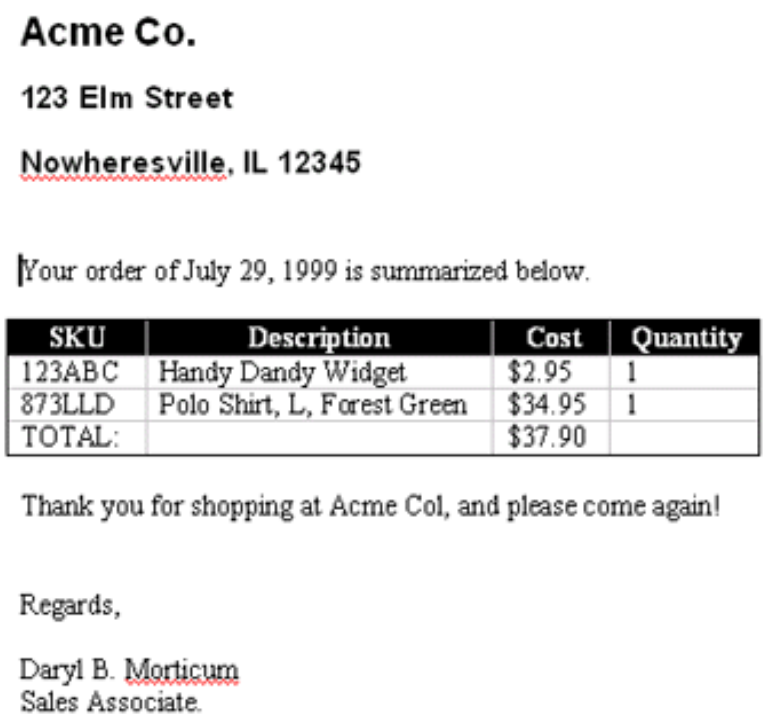
Acme Co. Product Listing

SKU	Description	Price	Order Qty
123ABC	Handy Dandy Widget	$2.95	1
123DEF	Super Duper Widget	$3.95	0
234XYY	Plain Old Lunch Bag	$0.05	0
345AXC	CD Jewel Case	$1.12	0
873LLB	Polo Shirt, XL, Forest Green	$34.95	0
873LLD	Polo Shirt, L, Forest Green	$34.95	1

Enter Order | Cancel Order | Print Confirmation Letter

The business need filled by `BuildDoc` is to enable sales associates to create form letters from the changed records of a Web page table. Only the data modified by the sales person is sent to Word, where it is formatted into a table. Obviously, all samples here are fictitious.

`BuildDoc` will read all of the information on the form, identifying which rows have been changed, and then creates the Microsoft Word document using only the information contained within the changed rows (see the sample output document below). `BuildDoc` uses a template file (buildDoc.dot) that contains the address header, and some preformatted text. It then writes a table into the document that has a row for each modified row from the Web page form.



Acme Co.

123 Elm Street

Nowheresville, IL 12345

Your order of July 29, 1999 is summarized below.

SKU	Description	Cost	Quantity
123ABC	Handy Dandy Widget	$2.95	1
873LLD	Polo Shirt, L, Forest Green	$34.95	1
TOTAL:		$37.90	

Thank you for shopping at Acme Col, and please come again!

Regards,

Daryl B. Morticum
Sales Associate.

How To Do It

We start by reading all of the Web page form fields into hidden form fields on the receiving Web page. In the source code below, note the " `onLoad` " call in the body tag. It calls the buildDoc VBScript subroutine, passing three parameters to it: the contents of the page's form (all the hidden fields), the location of the Word template file, and the number of rows received from the input form. The input form fields are all read and then, when the page loads, it calls the buildDoc subroutine. For the sake of brevity, we will assume that all variables have been first declared before use.

The code for the loading of the input form fields into buildDoc.asp is thus: -

```
<!DOCTYPE HTML PUBLIC "-//W3C/DTD HTML 3.2 Final//EN">
<HEAD>
        <TITLE>Build Document</TITLE>
        <META HTTP-EQUIV="Refresh" CONTENT="30;URL='orderForm.asp'">
</HEAD>
<%
        dotLocation="'\\servername\directory\theTemplate.dot'"
```

```
        intRowCount = Request.Form("rowCount")   'initialize a row counter
%>
<BODY Language="VBScript" onLoad="buildDoc document.theForm,
<%=dotLocation%>,intRowCount>
<FORM NAME="theForm">
<%
        itemCount = 0                            'set field counter to zero
        For Each Item in Request.Form            'count up the form fields
        itemCount = itemCount + 1             'using For..Next loop
%>
        <INPUT TYPE="hidden" NAME="<%=Item%>" VALUE="<%=Request(Item)%>">
<% Next %>
        <INPUT TYPE="hidden" NAME="numbRows"   VALUE="<%=intRowCount%>">
        <INPUT TYPE="hidden" NAME="fieldCount" VALUE="<%=itemCount%>">
</FORM>
</BODY>
</HTML>
```

We create an instance of the Word Document object, using the sample code immediately below. Note that in Internet Explorer 4+ this will fail unless the browser security is set to Low, or Custom with the appropriate setting to run programs.

```
<%
Set objWordDoc = CreateObject("Word.Document")
ObjWordDoc.Application.Documents.Add theTemplate, False
ObjWordDoc.Application.Visible=True
%>
```

We re-dimension our array so that it is the same size as the number of rows that are contained in the Web page's form. In this case, we set the Y-axis to a constant value of four because that is the number of columns we need in the output document. The X-axis contains the number of received rows from the form.

```
<%
Redim Preserve theArray(4,intTableRows)
%>
```

Now we are ready to examine all of the form rows. We do this by looping through all the input Web page form fields to collect each form field name and corresponding value. We test each to determine which array element to put it into, and then we put it there. The SELECT CASE statement in the code sample below is important. It is where we determine in which column the form field belongs. We used hard coded CASE options here for expediency.

```
<%
For intCount = 0 to frmData.fieldCount.value
   strOkay = "Y"
   strSearch = frmData.elements(intCount).name   'load the field name
   strValue = frmData.elements(intCount).value   'load the field value
   strPosition = Instr(1,strSearch,"_")          'get pos val of "_"
   intStringLen=strPosition-1
   If intStrLen > 0 Then
      strLeft = Left(strSearch,intStringLen)
      strRight = Right(strSearch,(Len(strSearch)-Len(strLeft)-1))
      Select Case strLeft
         Case "SKU"                          intArrayY=0
         Case "description"                  intArrayY=1
         Case "price"                        intArrayY=2
         Case "quantity"                     intArrayY=3
      End Select
      IntArrayX = strRight
      If strOkay <> "N" Then
         TheArray(intArrayY, intArrayX) = strValue
      End If
   End If
Next
%>
```

Now we are ready to begin creating the document. We start by setting the Microsoft Word Document object RANGE using our variable, `rngCurrent` , to the active document (just in case the user has a different document also open). Then we specify the table size by specifying its location (`rngCurrent`) and the number of rows and columns it needs.

```
<%
        Set rngCurrent = objWordDoc.Application.ActiveDocument.Content
        Set tabCurrent =  ObjWordDoc.Application.ActiveDocument.Tables.Add
        rngCurrent,intNumrows,4)
%>
```

Having created the document with the table, we now begin populating the table with data. First we point to the first row (`tabRow=1`), then begin a loop that will run through each row. We insert a line feed [`Chr(10)`] at the end of each row to put some white space between rows. Finally, we increment our row counter, output the dollar values with "FormatCurrency" to ensure use of dollar signs, commas and decimal places. Right justification of dollar

amounts is handled by setting the column in question to " `ParagraphAlignment=2` ". I won't tell you how much of a pain it was to discover how to do that! Suffice it to say that it is easier and better documented in VBA, which is not at all like what is required in VBScript.

```
<%
For j = 1 to intTableRows

ObjWordDoc.Application.ActiveDocument.Tables(1).Rows(tabRow).Borders.Enable=False

objWordDoc.Application.ActiveDocument.Tables(1).Rows(tabRow).Cells(1).Range.InsertAfter theArray(1,j)

objWordDoc.Application.ActiveDocument.Tables(1).Rows(tabRow).Cells(2).Range.InsertAfter theArray(2,j)

objWordDoc.Application.ActiveDocument.Tables(1).Rows(tabRow).Cells(3).Range.InsertAfter
FormatCurrency(theArray(3,j))

objWordDoc.Application.ActiveDocument.Tables(1).Rows(tabRow).Cells(4).Range.InsertAfter theArray(4,j)

objWordDoc.Application.ActiveDocument.Tables(1).Rows(tabRow).Cells(4).Range.InsertAfter Chr(10)

objWordDoc.Applicatoin.ActiveDocument.Tables(1).Rows(tabRow).Cells(3).Range.ParagraphFormat.alignment=2

tabRow = tabRow + 1

Next
%>
```

Finally, we finish up our document with some closing text and specifying the template's location, and then our subroutine.

```
<%
objWordDoc.Application.ActiveDocument.Paragraph.Add.Range.InsertAfter("Thank you for shopping at Acme C
please come again!")
objWordDoc.Application.ActiveDocument.Paragraph.Add.Range.InsertAfter(" ")
objWordDoc.Application.ActiveDocument.Paragraph.Add.Range.InsertAfter(" ")
objWordDoc.Application.ActiveDocument.Paragraph.Add.Range.InsertAfter("Regards,")
objWordDoc.Application.ActiveDocument.Paragraph.Add.Range.InsertAfter(" ")
objWordDoc.Application.ActiveDocument.Paragraph.Add.Range.InsertAfter("Daryl B. Morticum")
objWordDoc.Application.ActiveDocument.Paragraph.Add.Range.InsertAfter("Sales Associate")
```

```
End Sub
%>
```

Hopefully this will get your gears spinning for ways you can do something similar. We are sure that we aren't the only people who have had a need to create a document from a Web page's form. This is how we did it. If you have a better way, or an improvement on or method, we would love to hear from you.

This article was written as a joint collaboration between the following co-employees of the Education Networks of America: -

Gardiner Jones, MCP+I, MCSE+I, MCT (principal author)

Cal Evans (the right alignment genius)

Sandie Mountz (input form author)

Click here to download this articles support material.

RATE THIS ARTICLE

Overall
Poor ○ ○ ○ ○ ○ ○ Excellent
User Level
Beginner ○ ○ ○ ○ ○ ○ Expert
Useful
No! ○ ○ ○ ○ ○ ○ Very
Submit

enter the discussion

Related Articles on ASPToday

What's in your ASP toolkit?
Advanced ASP Charting Using ASPdb2000
Exporting ASP Reports to Excel Format
Dynamic Word Document Generation Using COM and ASP

Related Links

Office Development - Sample Scenarios and Solutions
Using the Microsoft Office Assistants to Create Web Content
Web Word Wizard™ from Document Automation
Book: Word 2000 VBA Programmers Reference

If you would like to contribute to ASPToday, then please get in touch with us by clicking here.