

A Client-Side Environment for ASP

Dino Esposito, VB2TheMax

Introduction

According to Microsoft's commercial people, more than 800,000 developers worldwide embraced ASP over the past three years and made it the undisputed number one technology for developing web applications, at least on Windows platforms. However, the idea behind ASP also influenced the way in which web applications are developed on other platforms. Java Server Pages, for example, is a Java-based technology that you can use just about everywhere, whose design has been clearly (and admittedly) influenced by ASP.

The goal of this paper is to explain the rationale behind a software tool that generalizes the ASP infrastructure, making it possible for you to exploit it in a server-less environment. ASP, in fact, specifically relies on some IIS capabilities to work. Without IIS (or, alternatively, Personal web Server), it would be impossible to expand an ASP page to the final HTML code. When you open a HTML page from Explorer, the browser loads and graphically displays the source code of the file. When you double-click on an ASP file, instead, in most cases you start up an instance of Visual Interdev that opens the file in edit mode.

No application available out there ever attempted to interpret the ASP source code off the server and translate it into HTML. By design, ASP code must run on an IIS web server and you must point to an ASP page only through a URL rather than through a file system name.

In this talk, I describe the underpinning of a client-side environment that makes up for this structural deficiency. I show you how to write a made-to-measure browser and a set of COM components that know how to process the ASP source code without the help of IIS or PWS. Once this infrastructure has been set up you can write "dual" ASP pages – capable of working both online and offline. Of course, not all ASP developers need dual pages, depending on their main activity. However, a significant number of web sites must be able to port to server-less environments and media (for example, CD or DVD) and, for such sites, dual ASP pages are a viable option.



Dino Esposito is a trainer and consultant based in Rome, Italy. He specializes in Windows and scripting, and has authored *Visual C++ Windows Shell Programming* and *Windows Scripting Host Programmer's Reference*, both from Wrox Press. Dino is also a contributing editor to Microsoft Internet Developer, MSJ and MSDN News. He's the cofounder of www.vb2themax.com, a VB-oriented web site, and works for Artis srl, a Rome-based consulting company active in the system integration and web-based architecture areas.

The Plus of ASP

So what makes ASP this attractive and effective to web programmers? For one thing, it's easy and requires limited skills. HTML and a bit of VBScript coding are in everyone's grasp. That using ASP at its fullest is as hard as any other programming tool or language does not change the key point. The user's perception of ASP is of something easy and within reach.

Plenty of web sites have been created using ASP. In general, you can partition web sites into at least two families: business and informational. Business web sites do any flavor of e-commerce and expose a certain application through the Internet. The perfect example of this class of sites is a home-banking web site.

Informational web sites just deliver information. They're scarcely interactive in the sense that they don't necessarily require users to enter data and obtain results. They have a staff behind which takes care of the content and updates it on a regular basis. The perfect example of this class is the MSDN web site. As a user, you just read and download on your machine. That's the point. What about the possibility of grabbing all the content in a single shot? Instead of a really lengthy download you could buy a series of CDs or a DVD. Once again, that's exactly what MSDN does.

People running informational web sites commonly deliver their content through a variety of media. If this need is a strict requirement already at the time in which you start the project, then you could choose to buy a specialized tool that allows publishing both on the Web and on other media. Most of the time, instead, you just start with an ASP-based web site and along the way you're asked to port it on local contexts. Believe it or not, a large part of your ASP code soon becomes useless. To maintain the same navigational mechanism and preserve as much code as possible you have a very limited range of choices. Either you write a custom pluggable protocol or you come up with some other mechanism to serve up dynamically built pages. Using static HTML pages is highly impractical for the major part of these sites, but if you could afford it, by all means consider this as another potential option.

The MSDN CD viewer utilizes a pluggable protocol. Have you ever run into a mysterious URL prefixed by `mk://` while snooping into MSDN CDs? MSDN just exploits a feature of IE 4.0+ (and the related `webBrowser` control) that supports custom URL protocols implemented as COM objects. What the browser does when it encounters such a custom string – and how it retrieves and processes the page – is up to the protocol itself and it does not necessarily need to go on the net. Learn more on pluggable protocols from the Cutting Edge column on the January 1999 issue of MIND (<http://www.microsoft.com/mind>).

Writing a Client-side ASP Interpreter

Moving static HTML pages from the Web onto a CD is a simple and easy task. You have only to replicate the same directory structure on the CD and make sure you have only relative links from the pages. That's it.

Moving dynamic ASP pages is a bit more complicated. I said earlier that ASP relies on specific IIS features. More exactly, this means that IIS processes all URLs with an `.asp` extension in a made-to-measure environment. In other words, IIS "does something" before it processes the ASP source code. This "something" is what prevents a browser from locally processing an ASP page. Consequently, a client-side environment for ASP just needs to create this ad-hoc environment.

From where does an ASP page take its dynamism? What makes an ASP page really dynamic? It's the ASP object model. An ASP page has two main characteristics:

- It contains script code
- It supports some built-in objects

An interpreter must be able to set the script code apart from the plain HTML code. In addition, it must be able to run the script code and insert the result in the right place within the page. Distinguishing between plain HTML code and script code is as easy as parsing for `<%...%>` blocks. Running the resulting script code is a bit more difficult. Let's see why.

Running script code from ASP pages

To run script code you must cope with the Windows Script COM interfaces. If you use Visual Basic, this is greatly simplified by the ScriptControl. If you like C++ better, then you'll find a helper component in the MIND archive (<http://www.microsoft.com/mind>) by searching for the August 1997 issue. The ScriptControl has been featured in MIND in the July 1999 issue and can be downloaded with full documentation from the MSDN Scripting web site at <http://msdn.microsoft.com/scripting>.

After figuring out how to execute script code, we're only half way. What about ASP intrinsic objects such as `Response`, `Request`, and the like? Before processing the ASP script code, IIS just instantiates these objects and makes them available to the ScriptControl (or the C++ code) for execution. Technically speaking, this procedure is known as "injecting objects in the ASP scripting namespace". Despite the rather baffling description, believe me when I tell you that it's as simple as passing the object's `IDispatch` pointer to a method.

A client-side environment for ASP pages is a special browser that does a number of things:

- Detects when you're calling an ASP page through a file system path. For example, `c:\foo.asp` instead of `http://server/foo.asp`.
- Initializes the ScriptControl and injects in its namespace instances of the ASP objects.
- Extracts the blocks of script code from the ASP source code.
- Executes this script code and concatenates the results with the rest of page (plain HTML code).
- Creates a temporary HTML file and points the browser to it.

Using the webBrowser control to build this browser is easy with both Visual Basic and C++/ATL/MFC.

Emulating the ASP Object Model

Unfortunately, the ASP object model is part of IIS. Therefore, this is exactly the part of the system we can't rely on! There's only one choice: write your own ASP object model. Thanks to the scripting injection mechanism, the script code can associate an `IDispatch` pointer with a name. `Response`, for the Windows Script interfaces that actually execute the script code, is only an entry in an internal table that points to an `IDispatch` interface, for example. Whether IIS, the browser itself, or a local object provides this interface really doesn't matter.

What you have to do is simply write a bunch of automation COM objects that expose all the methods and properties of the corresponding ASP intrinsic object. For example, `Response` will have methods like `Write` and `End`, and properties such as `Buffer`.

Conclusion

This is a high-level description of what you need to do in order to set up a client-side environment for processing ASP pages. There are lots of additional details that you might want to see discussed. For these, I forward you to a two-part article that appeared in MSDN Magazine.

Links and Resources

In the September and October 2000 issues of MSDN Magazine, you can find two articles covering this topic in all its gory detail. The first article shows you how to set up the browser in VB and implement the `Response` object in C++. The second article continues by creating the `Request` object in VB, and then discusses interesting side topics such as HTML forms, hyperlinks, early binding, and the usage of ASP components.

<http://msdn.microsoft.com/msdnmag/issues/0900/cutting/cutting0900.asp>

<http://msdn.microsoft.com/msdnmag/issues/1000/cutting/cutting1000.asp>