ASP and ASP.NET Technical Overview

Introduced as the hot new feature of IIS 3.0, Microsoft's Active Server Pages (ASP) technology changed the face of Web development. Prior to ASP, dynamic Web pages were usually created by applications written to the Common Gateway Interface (CGI) specification. CGI describes how applications can receive data from a Web server, which receives the data from users' Web browsers, often through HTML forms users fill out. CGI applications can process the data and create a dynamic Web page on-the-fly, passing the Web page back to the Web server for transmission to the user. CGI works well but is labor intensive. Any changes require developers to recompile the application, and CGI programming is fairly complex.

ASP leverages Microsoft's experience with scripting languages such as Visual Basic Script. Rather than writing a Web page and CGI application separately, ASP combines the two into a single file. Essentially, the CGI components are built into IIS, and a script within the Web page tells the CGI components what to do. These special Web pages are given their own filename extension—`ASP`—and are referred to as ASP pages. A typical ASP page might look like this:

```
<HTML>
<BODY>
<P>Hello and welcome to our Web page.
Today is <% Response.Write Date() %>.</P>
</BODY>
</HTML>
```

ASP pages can contain HTML code, just like regular Web pages can. But ASP pages also contain server-side script, in any scripting language the server understands (IIS understands both JavaScript and VBScript). The script is enclosed between `<%` and `%>` tags, allowing the Web server to easily identify the code. When a Web user requests the ASP page (by using her Web browser), the Web server reads the file from disk. Because the file has an `.ASP` filename extension, the server scans it for the all-important `<%` and `%>` tags. When it finds them, the server executes the code between the tags.

In the previous example, the server is directed to write out the current date. The key concept in ASP is that the server doesn't output the actual script code; it outputs only data the code tells it to. The final HTML sent to the requesting user's Web browser will have the server-side code stripped out, with only the results remaining:

```
<HTML>
<BODY>
<P>Hello and welcome to our Web page.
Today is Monday August 23, 2002.</P>
</BODY>
</HTML>
```

The user's Web browser interprets and displays the HTML normally, and the user never has access to the original server-side code. ASP has changed the way developers think about Web development, and a host of similar technologies—such as Java Server Pages (JSP)—have been released for other Web server products. ASP also addresses a common Web developer complaint, which is the difficulty of persisting session information.

Web Sessions

First, some background: In Web terminology, a *session* is the period of time that a particular user visits a Web site. The session includes all the Web pages the user views during that period of time. When the user stops visiting the Web site for 20 minutes or so, his session is said to *end* and the assumption is that he has gone off to surf another Web site instead. The problem is that Web servers don't inherently deal with sessions. Instead, Web servers deal with *page hits*: A user requests a Web page, the Web server delivers it, and the relationship between user and server is considered complete. When the user requests a second Web page, the server doesn't remember him from the last time and treats the transaction as a whole new relationship. In other words, the server has no way of tying individual Web page hits together into a session. That's a big problem for Web applications: Imagine if you visited an e-commerce site and placed an item in your online shopping cart, only to have the Web server immediately forget about you! The capability to tie page hits together into a cohesive session, and to maintain, or *persist*, information throughout that session, is key to using the Web for interactive applications.

CGI programmers came up with several different solutions, all of which involve quite a bit of coding. ASP simplifies session management by providing developers with a special `Session` object. ASP developers can attach information to the `Session` object, and ASP will make sure that the information persists across the session. Even if a hundred users request the same Web page at once, ASP can maintain a unique `Session` object for each user, enabling developers to personalize Web pages, construct shopping carts, and create other types of dynamic Web applications.

ASP made Web development so much easier that it was a mega-hit, easily as popular with Web developers as Visual Basic was with traditional developers. Unfortunately, as ASP was used in ever-larger applications, several weaknesses became apparent:

- ASP maintains `Session` objects in the Web server's memory. This technique is incompatible with Web farms, which seek to distribute incoming user requests across a number of identically configured Web servers. Web servers can't share `Session` objects with each other, so users have an inconsistent experience because subsequent page hits in their sessions are handled by different servers in the Web farm. This weakness limits ASP's scalability to a single server or requires developers to ignore the otherwise-useful `Session` object in favor of techniques that are compatible with Web farms.

- Complex ASP pages can become a miasmic of HTML and server script, making the pages very difficult to debug and maintain. Even the most careful developers could easily create pages that were incomprehensible a year later, costing additional time and money as they tried to remember what they were thinking when they originally created the pages.
- ASP's performance can be less than stellar because IIS has to process one line of server code at a time. The code is never compiled into a form that can execute more efficiently, and ASP has no real way of caching information to make commonly requested pages execute more efficiently.

The Fix for ASP

Microsoft recognized these problems and set out to create a whole new form of ASP, called ASPX (which was later renamed ASP.NET). Actually written in the .NET Framework's C# language, ASP.NET keeps the strengths of ASP while fixing the weaknesses.

First, Microsoft addressed the scalability problem by enabling developers to rip out the native `Session` object and replace it with their own. Microsoft even provides alternative `Session` objects that store session information in a back-end database server, where the data remains equally accessible to all the servers in a farm. This new technique provides an easy-to-use `Session` variable that can scale across the largest Web farms.

Second, Microsoft completely redesigned the way ASP works. Rather than mixing static HTML and server-side code, Microsoft created extensions to HTML that allow the HTML itself to act as server-side code. The result is that ASP.NET pages basically use HTML to define how the page will look and, in a separate section, provide all the code that adds interactivity to that HTML. ASP.NET pages are thus easier to maintain and even easier to write because even the most complex pages still have a simple, straightforward structure.

Finally, Microsoft handled the performance problem by making ASP.NET part of the .NET Framework. ASP.NET pages compile to the .NET Intermediate Language (IL) and execute on the Common Language Runtime (CLR), just like any other .NET application. Compiled pages are retained for future executions, making ASP.NET much faster than the original ASP.

ASP.NET Eases Development

A side effect of ASP.NET addresses another common ASP complaint: the lack of good development tools. Microsoft's ASP development solution, Visual InterDev, is (in many developers' opinion) an abomination, making it difficult to work with complex pages and nearly impossible to create good-looking Web pages because of a lack of decent design and layout capabilities. Many ASP developers, in fact, prefer to develop in a tool they jokingly refer to as "Visual Notepad," which is simply the basic Notepad text editor included with every version of

Windows. As a .NET Framework application, however, ASP.NET can be used from within the powerful Visual Studio .NET development environment. ASP.NET developers can use any .NET language within their ASP.NET pages, although Visual Basic .NET and C# are probably the most popular choices.

ASP.NET even lets developers create ASP-like pages that mix HTML and server-side code. By not forcing developers into the new ASP.NET coding model, Microsoft has enabled ASP veterans to quickly and easily switch to ASP.NET without having to learn all its new features. The open model also makes converting ASP pages to ASP.NET without having to completely rewrite them from scratch easier. Finally, Microsoft very wisely assigned a different filename extension, `ASPX`, to ASP.NET pages. The new extension enables IIS to distinguish between ASP and ASP.NET pages and enables both ASP and ASP.NET to run simultaneously on the same server. Thus, you can begin to deploy ASP.NET applications on a Web server that also needs to support older ASP applications, with no fear of compatibility issues.