

Developer's Corner

Stephan Teodorovich

Getting Started with ASP

The Internet has grown beyond the point where an online brochure will satisfy a typical company's needs for its web presence. If you aim to market yourself as a webmaster these days, you need to have some skill building online applications - websites that users can interact with, whether to get information targeted to their specific needs (i.e. a real-time stock quote), or to interact with other users via chat (i.e. an online community).

In this article, you will be guided through the process of learning one of the most popular frameworks for creating dynamic websites such as these - Active Server Pages (ASP). If you gain a strong knowledge of ASP, as well as some practical experience building websites with it, you should never have trouble getting work as a web developer. A quick search of your favorite online job directory with the keyword 'ASP' should be more than enough to convince you of that.

In this article you'll be introduced to the VBScript programming language and how to use it to write dynamic web pages with ASP. Before that, an explanation about how server-side scripting, and ASP in particular, differs from other Web scripting technologies that you may be familiar with, such as client-side JavaScript will be helpful. This will get you armed with the proper vocabulary and ensure that we're on the same page before jumping into the brave, new world of ASP.

Server-Side Scripting

To understand where ASP fits into the big picture of web development, you need to understand the concept of a server-side scripting language. If you've programmed web pages in Perl, PHP, or JSP before, you can safely skip this section - all of those are server-side scripting languages, and ASP works in much the same way. If you're coming to ASP armed only with knowledge of HTML (and perhaps with some CSS and/or JavaScript experience) then you'll find that server-side scripting is quite a bit different.

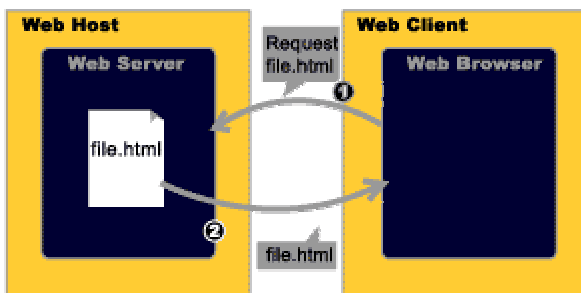


Figure 1: Simple HTTP request and response

If you've ever used any JavaScript in your pages, you know that the requested web page (file.html) can contain, in addition to plain HTML code, small programs written in JavaScript. These programs, or scripts, are read and executed by the web browser. So the browser must understand not only how to read HTML and display text and images, but also be able to run JavaScript programs appearing inside web pages. This

arrangement, where the browser runs the script after receiving it from the server, is called client-side scripting. The name makes sense - all of the script runs on the client-side. The server is completely oblivious to whether the file contains a script or not; it's all up to the browser (the client) to handle execution of the script.

ASP fits into a different category of technologies, called server-side scripting, where the server, not the browser, runs the script. This process is illustrated in Figure 2. As in client-side scripting, the browser requests a file (1). In this case, however, the filename ends with .asp instead of .htm or .html (file.asp, for example), indicating that the file contains an ASP script. The server recognizes this, and instead of sending the requested file directly back to the browser, sends it to the

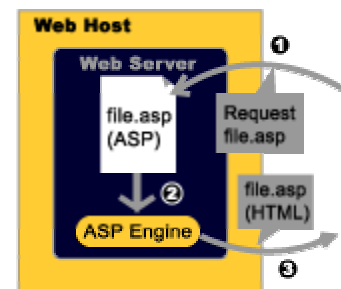


Figure 2: ASP request-response cycle

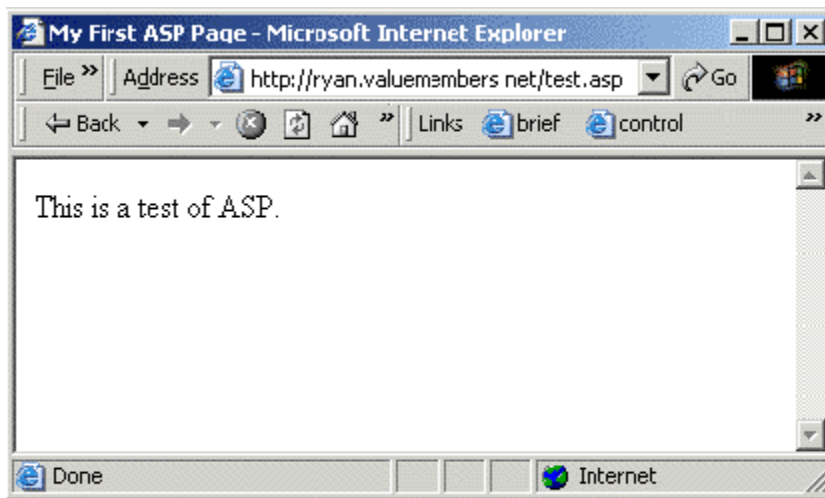
ASP scripting engine (2). The engine is a component of the server software that can interpret ASP scripts and output the results as HTML. The trick here is that any given script can output different HTML each time it is run, so what comes out of the ASP engine can be different for each client (browser) request. That dynamically generated page is then sent to the browser in response to its request (3). The ASP code contained in the page is interpreted and converted to plain HTML by the ASP engine before the browser gets to see it; so as far as the browser is concerned an ASP page looks just like any normal Web page. All the work is done by the server; thus the name server-side scripting.

The server that houses your website must be equipped with an ASP scripting engine to be able to process ASP pages. ValueWeb offers this feature on all of its shared hosting accounts, and uses an application called Chili.Soft ASP to provide the ASP functionality to our Linux servers.

Your First Script

You should begin by making sure your server is operating properly by putting up a simple web page and viewing it using your browser. Once your basic web page is up, you can try your hand at writing and viewing an ASP script. Open Notepad, or whatever text editor you prefer (programs like UltraEdit, HotDog Professional, HomeSite, and Visual Studio all provide convenient support for highlighting of ASP code, among other features), and type the following code:

```
<html>
<head>
<title> My First ASP Page </title>
</head>
<body>
<%
Write out a simple HTML paragraph
Response.Write "<p>This is a test of ASP.</p>"
%>
</body>
</html>
```



If you're thinking this code looks remarkably like an HTML file, you've got the right idea. Save the file as test.asp. In Notepad you'll need to either select "All Files" in the "Save as type" drop-down menu or enclose the filename in double quotes ("test.asp") to avoid having your file renamed to test.asp.txt. Upload the file to your account using FTP, or whatever alternative

method you prefer. Now, try viewing the ASP page in your Web browser by typing its address. This should be something like <http://www.mywebsite.com/test.asp>.

What you should see is a Web page containing the words "This is a test of ASP." much like that shown in the image to the left. The ASP scripting engine in the Web server is supposed to process the ASP code and convert it to plain HTML before sending it to the Web browser. In fact, if everything works like it's supposed to, you should see the following code when you use "View

Source":

```
<html>
<head>
<title> My First ASP Page </title>
</head>
<body>
<p>This is a test of ASP.</p>
</body>
</html>
```

Notice how the ASP code, which began with `<%` and ended with `%>` has been replaced with the HTML code for a simple paragraph of text.

How It Works

Assuming you've got your ASP test page working, you're probably curious to know how it works. As you can see, the first five lines of the file are just plain HTML.

```
<html>
<head>
<title> My First ASP Page </title>
</head>
<body>
```

One of the most convenient aspects of ASP (and, indeed, most server-side scripting languages in use today) is that it lets you intersperse your script code with plain HTML. The script code gets processed, while the plain HTML gets sent as-is (with a few notable exceptions that you'll see later on).

The next line marks the beginning of the actual ASP code in this page.

```
<%
```

Every block of ASP code must begin with this start marker (`<%`). This is what lets the ASP scripting engine know that it should start converting ASP code into HTML. This marker doesn't necessarily have to appear on a line by itself. A block of ASP code, markers and all, can in fact appear right in the middle of a single HTML tag on a single line. In this case, we have placed the marker on a separate line to make it stand out.

The next line actually doesn't do anything:

```
' Write out a simple HTML paragraph
```

This line is what programmers call a comment. The apostrophe at the start of the line marks the rest of the line as something for the ASP scripting engine to ignore. If you removed it or changed it in any way (except for the apostrophe) the script would not be affected at all. Nevertheless, it is good programming practice to intersperse little explanatory remarks such as this in your code for the benefit of anyone who needs to read and understand your code later on (including yourself). A comment need not appear on a line by itself. As soon as it sees an apostrophe, the ASP engine will ignore the remainder of the line on which it appears, so you can just as well place a comment on the end of a line, following the command to which it pertains.

The next line is the first and only ASP command in the script:

```
Response.Write "<p>This is a test of ASP.</p>"
```

In simple terms, an ASP command, or statement, is a piece of code that tells the ASP scripting engine to do something. By listing a series of statements one after another in the order you want them to be completed, you can write an ASP script that does any number of things in the specified order. In this case, however, we only needed the script to do one thing: write out the HTML code necessary to produce a paragraph of text that reads, "This is a test of ASP."

The code **Response.Write** is ASP's way of saying, "write the following into the Web page." The HTML code that you want the ASP engine to write is enclosed by the double quotation marks (any portion of text in ASP is called a text string).

Having finished telling ASP what to do, we ended the ASP code block with the ending marker, `%>`. Like the starting marker (`<%`), the ending marker serves to indicate the boundary between normal HTML code and ASP script commands. When it sees an ending marker, the ASP scripting engine goes back to just sending the HTML code as typed.

```
%>
```

To finish off the page, we added the HTML closing tags:

```
</body>  
</html>
```

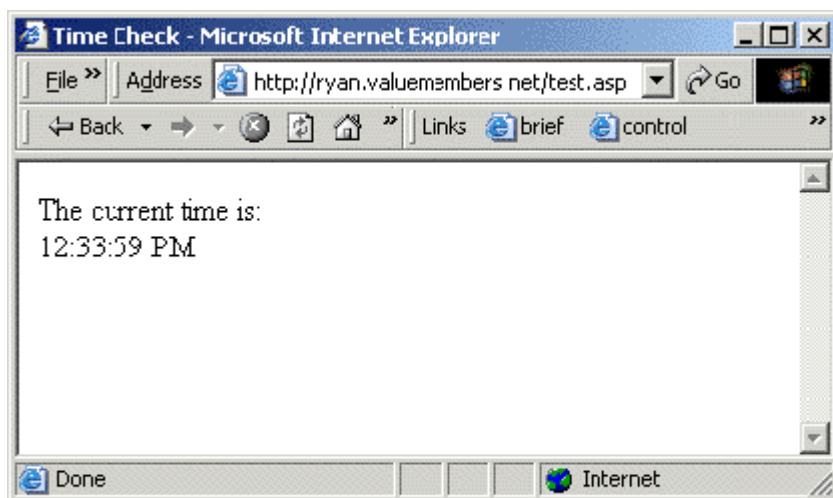
You should now be able to point to each line in your test.asp file and understand what it does (or, in the case of the comment line, what it doesn't do).

Although the test.asp page just produced static HTML, it is important to remember that it is created dynamically every time a browser requests the page. The strength of ASP is that it creates dynamic pages, which can produce different results with every page request. Alter the code of your test.asp file to read as follows:

```
<html>  
<head>  
<title> Time Check </title>  
</head>  
<body>  
<%  
Response.Write "The current time on the server is:<br>"  
Response.Write Time  
>%>  
</body>  
</html>
```

Save the changes, and then place the updated file on your server. When you view the page in your browser now, you should see something like this:

Notice that when you refresh your browser, the time is updated to match the time according to the computer on which the server is running. Notice also that when you use the "View Source"



command in your browser, all you see is plain HTML code. ASP writes the time into the page before sending it to the browser. The result is a simple, but dynamic page.

Comparing the original test.asp script to the modified one, you'll notice that there are two **Response.Write** commands, and that the second does not contain quotation marks, simply the word **Time**. In this script, **Time** tells the ASP engine to insert a text script representing the current time (i.e. "12:33:59 PM"), and that is what the Response.Write command prints out. Also notice that the two **Response.Write** commands are executed in the order they appear, from top to bottom, producing the desired HTML code.