Category: ARTICLE
Subject: "Web Apps" 101 - Part 2
Name: Creces, Gus

"Web Apps" 101 - Part 2

Clarion ASP Templates

ASP stands for "Active Server Pages". This is a Microsoft technology that allows you to take ordinary **static**, or unchanging web pages and make them **active** or for want of a better term, **interactive**.

Active Server Pages are Web pages that contain server-side scripts in addition to the usual mixture of text and HTML (Hypertext Markup Language) tags. Server-side scripts are special commands - usually written in VBScript or JScript - that you embed into standard Web pages. These scripts are then executed by IIS (Microsoft's **I**nternet **I**nformation **S**erver) before the pages are sent from your Web Server (IIS) to the visiting Web browser.

When a visitor to your website types a URL in the Address box or clicks a link to your website on any Web page, they're asking a Web server on a computer somewhere to send a file to the client Web browser on your computer. If that file is a normal HTML file, it looks exactly the same when your Web browser receives it as it did before the Web server sent it. That's why it's called a static page. After receiving the file, your Web browser displays or renders its contents as a combination of text, images and sounds.

In the case of an Active Server Page, the process is similar, except there's an extra processing step that takes place just before the Web server sends the file. Before the Web server sends the Active Server Page to the Web browser, it executes all server-side scripts embedded in the page. Some of these scripts display the current date, time, or perform data table access and wrap that information in HTML and Javascript. Others scripts may process information the user has just typed into a form, such as an entry in the Web site's guestbook. To distinguish them from normal HTML pages, Active Server Pages are given the ".ASP" extension.

The latest implementations of ASP (i.e. ASP.NET) allow for the scope of scripting languages embedded in .ASP pages to be broadened to include VB.NET, C#.NET and even C++.NET, in addition to VBScript and JScript. It's important to remember however, even with .ASP pages what your browser gets back is HTML/Javascript/CSS. Natively, without browser extensions and plug-ins, HTML/Javascript/CSS are all that a browser can really interpret. The other-language scripts embedded in ASP pages are meant to be run inside the IIS server before the page is sent back to the browser. Picture it as a sophisticated form of mail-merge where the script portions of .ASP page are replaced inside the IIS server by data produced during the the execution of those scripts. Data produced can be anything from a date-time stamp to a browse or a form built from data in one of your SQL data tables.

It's also important to note that to create a working .ASP page from scratch, or to debug, change, improve .ASP pages, it's mandatory that the developer becomes more than familiar with the following scripting systems:

1) HTML
2) Javascript
3) CSS (Cascading Style Sheets)
4) One or more of the following: VBScript, JScript (not to be confused with Javascript), C#, Visual Basic, C++ and several others

Clarion's ASP Templates come into play here by generating most of the embedded scripts required. To do that they begin by using a generic HTML page template to which they add C#, VB.NET or VBScript to do the work of adding the interactive data to be inserted into your web pages. In effect, Clarion's ASP templates do not generate Clarion code in the usual way, which you then compile to produce an executable file. Instead, they generate .ASP pages. Which, as already outlined above, are HTML pages with one or more of the accepted scripting languages embedded.

It's important to note here the word "generic". The starting point for a web browse or web form produced by these templates is always a *generic* HTML page and a *generic* script, which the developer is free to change via ordinary embedding inside embed points provided by the Clarion ASP templates. Of course, what the developer is embedding now isn't Clarion code. It's HTML/Javascript/CSS and one or more of the accepted scripting languages, VB.NET, C#.NET, C++.NET, VBScript and so on. If the developer hasn't done the personal professional development involved to learn these things they're pretty much stuck with *generic*. If they discover bugs, design limitations or even simple design irritants in any of the generic stuff, they're dead in the water until they learn HTML/Javascript/CSS and at least one of the required scripting languages.

Generated ASP pages, are placed in a special directory on the IIS server machine to be sent to the client browser when someone types the ASP page name into their browser's address area, or links to the ASP page from another web page.

One nice thing about ASP pages, most ISP's are happy to allow them on their web servers. ASP is ubiquitous and probably the gold standard for dynamic web page development, at least in the Windows Server world. Of course, if you have a set of company data tables like your order processing system sitting at company headquarters, you're either going to have to replicate those data tables to the ISP, **or** give the ISP's hardware remote access to your data tables via another IIS connection so that it can pick up raw data from your SQL server. Unless you or your customer are prepared to allow this you're better off hosting the web pages yourself via an IIS server running local to the data tables.

Another ASP advantage, is greater design control over the web pages. Since they're not based on an emulated reproduction of an application's browses and forms as they are in the case of Web Builder, the developer has much greater license to add the sort of embellishments that are only available for screen design in a browser. Of course, the same requirements for greater design control apply as in all web page design, a real working knowledge of HTML, Javascript and CSS.

ASP based web applications, whether built by hand from scratch, or built using Clarion's ASP templates, follow the two-piece principle outlined at the beginning of this series of articles. The **client piece** consists of a set of web pages containing HTML/Javascript/CSS code from which the user interface is rendered inside the browser. The **server piece** is provided by Microsoft's IIS server which takes care of TCP/IP connectivity and sending pages to the browser as well as executing the other-language scripts (VBScript, JScript, C#, VB, C++) embedded in these .ASP pages in order to insert the interactive data from your data tables into the HTML pages. Be aware that the embedded scripting languages *never make it to the client browser* which wouldn't know what to do with most of them (except VBScript and JScript) if they did. These non-HTML, non-Javascript, non-CSS elements are stripped out and data elements are hard-coded into your pages in their place. That's how data elements from your data tables make it to the browser. They're tagged, wrapped and hard-coded into your web pages before they're sent to the visiting browser.

That brings up a nasty aspect of ASP and Internet Explorer that I'm sure Microsoft would sooner have us forget. ASP has been around for quite some time in various versions since the introduction of IIS 2, if not earlier. In those early days, Javascript was just being invented by Netscape in order to give browsers some programming capacity that just didn't exist in HTML, and still doesn't. Microsoft introduced "Browser Plug-ins" in the form of downloadable Activex controls that extended their browser's normal capabilities. They also gave their browser the ability execute VBScript and JScript.

What's so nasty about this if it improves the browser's capabilities? The reason is that these Activex controls and VBScript are too powerful, in that they give the script writer too much power to cause side effects on your computer. Do you really want to allow a script to reach you through your browser that can modify data and run executables on your computer that can have detrimental side effects? Of course you don't. These extra capabilities provided to Internet Explorer via Activex and unrestricted client-side scripting by Microsoft came back a number of years later to seriously bite them.

Javascript has none of these detrimental side effects. You cannot really write a script in Javascript to do any harm to the computer executing the script, since Javascript has no ability to create, change or run files on your computer. The only thing it can do, file wise, is create cookies which are small text files for temporarily storing "state" information. These are not executable files and they are not harmful. While cookies can be abused, by data scavenging software, the fact that they're abused is a side-effect, not the direct cause of Javascript. Like leaving the key to your house under the front door mat, it's not the key's fault, the lock's fault or the mat's fault when someone finds the key and steals your hi-def TV. The sole interest of Javascript combined with HTML and CSS is to increase the screen rendering capability of your browser. And to do it in as safe a manner as possible. All modern browsers are for all intents and purposes HTML/Javascript/CSS compatible. It was not possible several years ago to write a single web page and expect it would render more or less the same in all browsers without having separate versions for each browser. That is no longer the case. All modern web pages and web apps *can* restrict their page content to HTML/Javascript/CSS and expect a fairly similar outcome in the browser.

Granted, ASP can produce safe web pages too, as long as it avoids including Activexs and confines page content to data, HTML, Javascript and CSS.

A real advantage of the ASP approach to web app building is the fact that the IIS server can handle multiple client browser connections all without starting your "server" app multiple times the way that Web Builder does. In fact there is no "server" app in the background except IIS. With Web Builder your template-doctored application generates the web page ultimately being sent to the browser. In an ASP setting those same pages are generated from generic, pre-built HTML pages containing server-side scripts which are pre-processed, server-side in order to produce the final pages sent to the browser. Clarion's ASP templates help you create those server-side scripts.

If there is a downside to ASP it's in the fact that it does not really separate your data from the code that renders it in the browser.

So when a page is requested, as I've already stated, your data elements are hard-coded into the page each time it's requested and the whole page is sent to the browser each time as well. If you compare how that works in light of how a desk top app works its equivalent to the data table sending you a picture of what the screen should look like when a browse is requested. That's not how web pages *have* to work and it's certainly not how data tables and SQL servers work. In desktop apps, data elements are sent to the application and the application's screen design determines how the data are displayed. Changing your screen design, to move the location of a column or field has no direct impact on the data base unless you're adding a field or column that wasn't there before. That's because data componentry and rendering code are entirely separate. For the most part, ASP does not work this way. Because it's designed in principle on the mail-merge model (though far more advanced), pages built with ASP tend to have page data elements intermingled with code and screen rendering elements. It's the nature of the beast. Keep this fact in mind later in this series of articles when we begin to discuss how CHT servers handle data elements and screen rendering scripts entirely separately.

Clarion ASP Templates Summary

Strengths

1. *You can build a rudimentary web interface without knowing much about browsers*
2. *You can build your ASP browses and forms in the context of the ever-familiar Clarion Template environment*
3. *Very scaleable since the server is IIS alone and not an app that must run separately for each connected client browser*
4. *Good ISP support as long as you're willing to have them also host your data tables and/or connect to them over the internet*

Weaknesses

1. *To build sophisticated, fully-under-your control web interfaces you have to learn the very things ASP is supposed to help you avoid having to learn (HTML/Javascript/CSS) as well as at least one other server-side scripting language like C#, VB, C++, VBScript, JScript*
2. *Data intermingled with screen rendering code*
3. *Reliance on IIS forces you to give access to your data tables to the ISP or if hosting IIS yourself, to learn the care and feeding of an IIS server*
4. *Many generic ASP systems rely heavily on utilizing Activexs and/or VBScript at the client side, introducing the possibility of mistrust by visitors to your site and the possibility that your web-pages won't work correctly when the client browser has these capabilities disabled*
5. *Can't build forward from an existing Clarion .APP, only the existing .DCT*

Coming up in "Web" Apps 101 Part 3...

Clarion PHP Templates

To obtain a PDF version of this document, .

Message posted via HNDMTSCL.APP (Client Application) interacting with HNDMTSSV.APP (Server Application).

Cheers...

Gus M. Creces
The Clarion Handy Tools Page
http://www.cwhandy.com