Category:	ARTICLE
Subject:	**"Web Apps" 101 - Part 1**
Name:	Creces, Gus

"Web Apps" 101 - Part 1

This series of articles, of which this is part 1, will attempt to provide a theoretical backdrop to aid in understanding what to use and where to turn when the time comes that you want to build your first "Web" application. It will give me a chance to test our CHT Client Server technology (see below for a definition).

What Is A Web Application?

As a broad definition, at least from the standpoint of Clarion developers, a web app can be defined as an application that lets you browse, edit, delete, process and report your data across the internet. The data tables, (ISAM or SQL) are not local to the user's machine. They're somewhere in "virtual space", perhaps on the Internet perhaps on a company Intranet.

While there are a lot of variations on the theme, "Web App" there really are only two kinds:
1) Web apps where the application runs inside a browser. (For clarity, at CHT we call this a **Browser Server Application**)
2) Web apps where the application is an EXE, that resembles for all intents and purposes an ordinary desktop application (For clarity, at CHT we call this a **Client Server Application**).

All web apps, consist of two parts:
1) A user interface part - the part that the end user interacts with.
2) A server part - the part in the background to which the user-interface part connects and with which it interacts.

Unlike desktop applications which are primarily one-piece, all Web Apps are by their very nature two piece applications. Since the data tables are not on the user's machine, something out there is cyberspace has to act as a pump to package up the data and send it to back to the user so he/she can interact with it. This packaging component is called the **Server**. The user needs something to manipulate, control and display the data. Call this part the user interface or **Client**.

Clients come in all shapes, sizes and colors, but in general, they are either 1) .EXEs or they are 2) SCRIPTS running in a browser.

Servers also come in all shapes, sizes and colors. They're always .EXEs.

In the realm covered by 2) above, since all web pages are essentially scripts running in a browser, one shouldn't confuse "Web Sites" with "Web Apps". For the most part, Web Sites in the usual sense, consist of static pages constructed of various quantities of HTML/JAVASCRIPT/CSS - the **scripts**. A web site only becomes a "Web App" when it gives you the ability to Search, Edit, Change, Delete, Process and Report data via the internet or intranet. Though not all Web Apps give you full privileges to do all of the above, at a bare minimum, they should give you Data Search and perhaps Data Edit capabilities.

Clarion Based Web-App-Building Resources, Then And Now

Clarion Web Builder Templates

This technology has been available to Clarion developers for at least 7-8 years. To some extent, this technology ruined, or at least significantly slowed the natural growth, of web application building for many Clarion developers for that whole time, because it caused developers to misunderstand (and underestimate) the professional development requirements needed (things the developer must learn to do) before building web applications.

In a perverse way, because of the simplicity of getting a rudimentary web app off the ground, this technology sucks you in. But it's capabilities never meet its promise and never even gets close to what you can do with Clarion in the desktop environment. And that failure to deliver led a lot of Clarion developers to snub their noses at Clarion when it came to "Web App" development.

Clarion Web Builder takes an existing desktop application and attempts to make it dual purpose. It becomes a Desktop Application **and** a Web Server at the same time. You do this by dropping templates on the application procedures. That causes these procedures to pump out HTML/Javascript pages - or scripts if you will - that roughly emulate the windows in each of those procedures when those scripts are rendered inside a browser. Looking back at my earlier assertion, this technology, like all Web App technology consists of two parts, a **Server** part - represented by your template-altered application - and a **Client** part - represented by the pages or scripts running in the client browser. This arrangement is exactly in keeping with the broad definition of a "Web App" given above.

To become a web server, your application is made to interact with a Clarion EXE or DLL called the **Broker** which acts as an external-to-itself sending, receiving mechanism for your application. In fact, the broker element is really an http **proxy**. The EXE Broker is a stand-alone web server in its own right. The DLL Broker uses ISAPI (Internet Server Application Programming Interface) and requires you to install IIS (Microsoft Internet Information Server) to perform the sending, receiving duties and the broker DLL acts as a proxy between your app and IIS. IIS handles the HTTP communications over the Web. The closer you look at this the more it becomes obvious that just to keep the server side up and running there are a lot of *black boxes* involved, each of which entails its own learning curve. At this point, the initial excitement of web enabling even a very basic app by dropping a few templates, is quickly overshadowed when the unprepared developer realizes there's a lot more complexity involved than he/she probably ever bargained for.

While a number of developers were able to get Clarion Web Builder apps up and running. This technology comes with a a built-in design limitation that really kills it for all but a few very small-user-count implementations. Every time a new user starts your web app from his browser client, the broker starts a new copy of the application on the server computer. Ten users connected, ten intances of the application running. Fifty users connected, fifty instances of the application running.

It's not hard to see why no ISP (Internet Service Provider) in his right mind would let anyone run one of these Web Builder Applications on his server computers. It would inevitably drag the hardware down to a crawl on busy days and perhaps even cause it to crash unless there is a seriously SMALL limit set on the number of simultaneous users.

There were, and perhaps still are, one or two ISPs who provide a platform for running these Web Builder applications. Though it's a dead end for them. And, in my view, a total waste of time. Whether you opt for an ISP or host your web-enabled application on your own hardware, system resources are a real limiting factor for this kind of approach to web applications. Either you keep the number of simultaneous users to an absolute minimum or you pay the price in speed as memory-caching kicks in when the app-instance count rises. A second major drawback with Web Builder is that the web pages usually look like crap until the developer learns to use HTML, Javascript and CSS and spends the time necessary to inject it into the Web Builder templates. To help you do that, Topspeed introduced a scripting language in Web Builder called TSScript. All that did was give you another language to learn in addition to HTML/Javascript and CSS. In my view if you're going to take the time to learn HTML/Javascript/CSS anyway, there are a lot of better ways to leverage that new skill than Clarion's Web Builder technology.

Recently, SV has released a new ISAPI-based broker that implements threads. This speeds things up a bit in simultaneous user situations because the HTTP requests coming in to the broker from various clients are not cached end-to-end. The broker uses C6's ability to create MS windows threads to improve the broker's listening ability. In other words, http requests can be accepted near-simultaneously. While this is definitely a design improvement, it does nothing at all to change the fundamental design flaw inherent in Web Builder, that of starting multiple instances of the server application each time a user connects.

Clarion Web Builder Templates Summary

Strengths

1. *You can build forward on an existing app*
2. *You can build a rudimentary web interface without knowing much about browsers*
3. *You can use a .TPS file backend data table*

Weaknesses

1. *Not scaleable*
2. [illegible]

2. *Uses way too many system resources when multiple users are connected*
3. *To build a sophisticated, good looking web interface you have to learn the very things Web Builder is supposed to help you avoid having to learn (i.e. HTML/Javascript/CSS)*
4. *Not very robust, prone to breakage, many pieces need to work together, need to know care and feeding of IIS*
5. *Little or no ISP support*
6. *Slow as molassess when more than a few users are connected simultaneously*

Coming up in "Web" Apps 101 Part 2...
Clarion ASP Templates

To obtain a PDF version of this document, .

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

Cheers...
Gus Creces
The Clarion Handy Tools Page