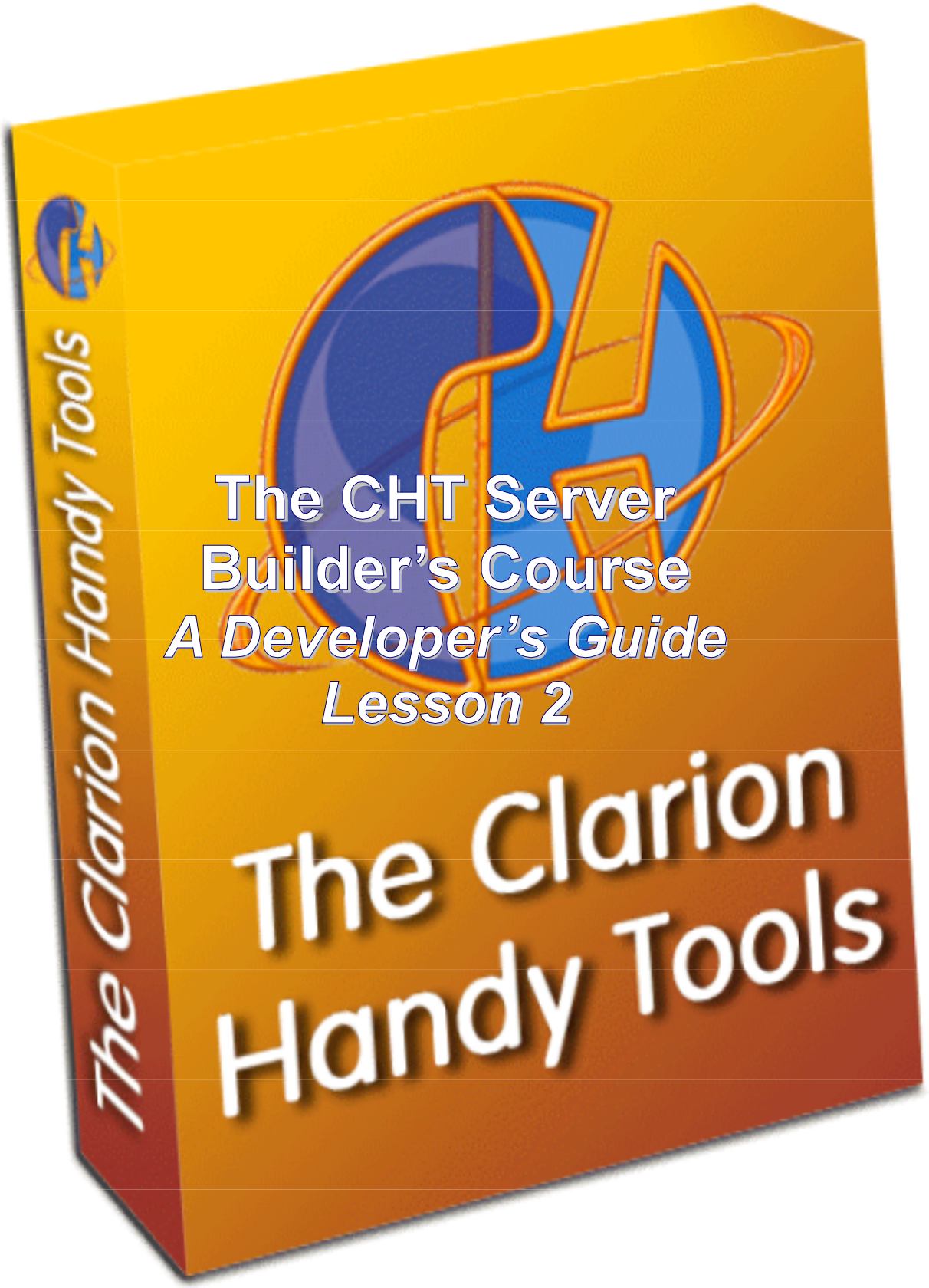
The Clarion Handy Tools
The CHT Server Builder's Course
A Developer's Guide
Lesson 2
The Clarion Handy Tools

INTRODUCTION

Here are somewhat more lengthy responses to Lesson 1 Exercises than the ones we expected from participants. Please take the time to read them and to compare these answers with your own in order to update your information about each topic as necessary.

This lesson deals with the differences between HTML pages constructed at the server end and HTML pages constructed at the browser end, inside the browser, using JavaScript. We'll show you how the HNDMTSNG.EXE Web Group Server bundles JavaScript Data Objects which can be used in browser-based scripts to construct pages. We'll also explain how the scripting tool built into the server is used to construct scripts and how the server gets these scripts to the browser. Finally, we'll explain how data packages come to the browser and how they differ from script packages. The lesson concludes with an overview of server configuration variables, and back end data variables available for script building.

REVIEW OF LESSON 1 EXERCISES

EXERCISE 1:
What is a dynamic web page and how does it differ from a static web page?

Dynamic web pages tend to change each time they're accessed, often even while they're loading. Static pages stay the same. Most web sites today are still relatively static but dynamic pages are definitely the emerging standard. Once a static web page loads, there's very little on it that changes. Dynamic web pages tend to adapt and change to meet visitors' needs and requests.

EXERCISE 2:
Describe briefly, two basic techniques for dynamic web-page creation. (Hint, server end - client end).

Server-Side Page Creation:
Dynamic web pages are processed and assembled by the server.

Client-Side Page Creation:
Dynamic web pages are processed and assembled by the client (the browser).

EXERCISE 3:
Sign into your own, running Web Group Server and click the login menu. No need to log in. Right click the page and select "View Source". Describe briefly what each of these lines does.
3a) Javascript file request hndmtsngcfg.js
3b) Javascript file request hndmtsngsub.js
3c) Javascript file request 7xxxxx_xxxxxxxx_xxxxxxx.js (names will differ)
3d) Javascript file request hndmtsngsvr.js
3e) Style sheet request hndmtsngsty.css

3a) File **hndmtsngcfg.js** contains server configuration information, stored as JavaScript Data Objects which are discussed below. This information originates in the "Server Variables" file. You can take a look for yourself if you look in the server's "run" directory. Server configuration objects included are: **flg**, **button**, **choices**, **file**, **image**, **keyword**, **link**, **menu**, **query**, **system**. There are also a couple of generic functions: **putcookie()** and **getcookie()**. This file is listed first so that the browser reads it first because the scripts that follow have dependencies in these values. That is, the scripts that follow may use variables declared here.

3b) File **hndmtsngsub.js** contains JavaScript subroutines from the *Config - > Server Scripts -> Select Javascript Items* list. This is a combination of Javascript subroutines subdivided into categories such as buttons, errors, links and so on. The server owner is free to add scripts and modify existing scripts as he/she sees fit. This file is listed second because it makes use of variables declared in the file above and because it contains subroutines used by the files that follow it.

3c) File **7xxxx_xxxxxxxxx_xxxxxxxxxxxx.js (names will differ)** contains back end data sent up from the server in response to the query or menu that caused a particular page to be displayed. This is the only "Dynamic" component in the entire list of files being discussed here. On pre-login pages this contains mostly generic stuff since there is nothing specific known about you. Once you log in, this contains information about the individual logging in. And once you post some kind of a query on a messages query control or members query control, it contains the data that answers your query. From this data then, a browse is built or an update form is built.

The reason for the "funny" name is to ensure that this file will *always be refreshed* from the server and never from the browser's cache. The browser will always request this file unconditionally because it knows from the name that it does not exist in its cache area. The other fixed name files do not change from page to page, unless the server manager changes and regenerate his/her scripts, so the browser requests those conditionally, which is to request the server to send those files only if the dates are newer than the ones the browser now has in its cache.

Consequently, it is *extremely* important that the server hardware's operating system be correctly configured to the time zone and exact time of day for the server's location. All time calculations are calculated back to GMT time based on the computer clock where your server is running.

3d) File **hndmtsngsvr.js** contains the HTML/JavaScript components from the *Config -> Server Scripts -> Select HTML Items* list. These are your HTML pages. Often there is as much JavaScript in here as there is HTML since JavaScript can write HTML which the browser uses to create pages. If you look at the pages that we've supplied with the HNDMTSNG.EXE server you'll see that we tend to use JavaScript in the HTML whenever there are variables to resolve.

The reason for this is obvious.

HTML doesn't really have the capability to do this, but since JavaScript can write HTML, a variable from the back end or from the Config file can be resolved to its contained value via JavaScript and that can then document. write() the necessary HTML to build the page. This file follows the other three because it is dependent on them. It uses Config variables from **hndmtsngcfg.js**. It uses JavaScript subroutines form **hndmtsngsub.js**. It uses back end data and validation subroutines sent up from the server in **7xxxx_xxxxxxxxx_xxxxxxxxxxxx.js**. So those files must be declared first and scanned first by the browser.

4e) File **hndmtsngsty.css** contains the style sheets. These are used during construction of the pages to describe various page components like buttons and entry fields and table layouts. You would think from its position in the list that it should be loaded earlier, before **hndmtsngsvr.js** where pages are described, since those page descriptions contain CSS references. With CSS declarations this is not necessary as long as they are placed somewhere inside the **<head></head>** area of the page.

EXERCISE 4:
Describe briefly how to clear your browser's web cache.

Open your browser and execute menus as follows: *Tools -> Internet Options -> Delete Files -> (Check) delete all offline content*.

EXERCISE 5:
Why is clearing the web cache sometimes important and how does this relate to the CHT Web Group Server?

Because web browsers derive from those primitive early things that were built on slow-as-molasses connections, they try to be as efficient as possible about what they download. If something like an image is used in one page and again in another, the browser is smart enough to know it already has the thing being requested and simply re-displays it from the cache without reference to the server.

In the old days this "smart" behavior made sense and saved a lot of bandwidth. Since pages then were all static, there was no need to do a lot of checking to see if a particular page component had changed or not. But this is today, and dynamic web pages are the new standard. The browser should be configured (see question 7) to cross-check with the server to see if any one page component has changed since last placed in the cache. Hence it sends a conditional request for page components, with a date-time stamp attached.

The server has been programmed to check this stamp and to send the file only if it's date stamp is newer than the one which the browser has in its cache. To see this happening, keep an eye on your server's real-time status log as page requests come in. You'll see the following happening quite often:

```
HNDBrowserServer.SendNoContentHeader()
HTTP/1.1  304 Not Modified
Last-Modified: Thu, 1 MAY 2003 13:06:00 GMT
Date: Tue, 30 SEP 2003 13:04:43 GMT
Filename:IMAGES/DISCUSS.GIF
```

When the server returns **304 Not Modified**, it's simply saying to the browser, "I have nothing newer for you. Use the file you've already got in your cache." In this case the file that the browser is requesting a date check on is **IMAGES/DISCUSS.GIF**. The files in your browser cache accumulate over time, they are not automatically erased.

Therefore it's important that a browser user be aware of how to dispose of them. Without doing this, the cache begins to fill and web access starts to slow down. It's not enough to "Refresh" the page to get new copies of all page components. All that does is repeat the previous page request and that repeat request will again contain conditional requests for the components already in the cache. If you want to guarantee that you have the very latest version of a page, say in a testing environment, you must first clear the web cache.

This behavior is not a CHT invention. It is the normal behavior of all browsers and the expected behavior of servers to interact in this way. You may have seen JavaScript errors on web pages when you're interacting with certain dynamic-page web sites. These errors may, in fact, be the result of less-than-optimum browser settings or bad computer clock settings that cause the "code" files in a dynamic page environment to get out of sync with the "data" files or the subroutines file to get out of sync with the page description files so that calls to non-existent JavaScript subroutines are made from the pages.

If you run a server, this knowledge is *important* and it's important that you know how to explain to your users that their browser must be configured correctly.

EXERCISE 6:
Explain how you go about finding out where your web cache is located.

Open your browser, and execute this menu sequence. *Tools -> Internet Options -> Settings*. The path to your web cache appears beside the "Current location:" prompt. To find this location programmatically in your Clarion programs, use the CHT **HNDRegistry** class and request the contents of this entry:

HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders\Cache

EXERCISE 7:
Explain how to configure I.E. to optimally interact with dynamic-page websites like the CHT Web Group Server.

Open your browser, and execute this menu sequence. *Tools -> Internet Options -> Settings -> Check for newer versions of stored pages -> (Check) Every visit to the page*.

EXERCISE 8:
Briefly, describe what each of the following does on a web page and what the acronym stands for (if any).
a) HTML
b) CSS
c) Javascript

8a) HTML (Hypertext Markup Language) is a text formatting description system. It's a way, with words, to describe how a web page should be rendered by the browser. "Language" is appropriate, since HTML does consist of words. But it's a far cry from what experienced computer software developers would call a language. HTML is actually a tiny subset of page description systems used in typesetting machines from a previous era called **SGML** (Standard Generalized Markup Language).

HTML really is inadequate to do what needs to be done these days in a dynamic web page environment. DHTML, XHTML, XML all improve on this somewhat making the language *extensible* to include things that the original designers might not have thought of. In our own server scripts we've tried to adhere to the DHTML standard. This is the best choice for dynamic page creation - the **D** in **DHTML** stands for dynamic. It has stricter rules than plain HTML making it better used in conjunction with JavaScript and the browser DOM (Document Object Model).

8b) CSS (Cascading Style Sheets) are a way to untangle the complexity and repetitive "wordiness" of describing the same things over and over again. If your page buttons, for instance, have seven or eight non-standard characteristics, without CSS you would have to repeat those characteristics every time your page uses a button of that sort. If you have ten buttons on a page, all the same you'd need to describe each one separately, in full detail if it weren't for the convenience of CSS. Style sheets are always declared or described in the <head></head> portion of your page and they look like this:

Below is a single style sheet. It consists of a name followed by a set of properties and their values delimited by a set of curly braces {}. Notice that the property tags (on the left) all end with a colon (**:**) and that the property values (on the right) end in a semi-colon (;). The .CSS file is not a style sheet. It is a file containing one or more style sheets. (**CSS Example Next Page**)

```
.bldr_button_go {
  font-family:                arial;
  font-size:                  7pt;
  font-stretch:               normal;
  font-style:                 normal;
  font-variant:               normal;
  font-weight:                bold;
  background-color:           cornflowerblue;
  border-top-width:           1;
  border-bottom-width:        1;
  border-left-width:          1;
  border-right-width:         1;
  border-style:               outset;
  border-color:               transparent;
  padding-top:                0;
  padding-bottom:             0;
  padding-left:               0;
  padding-right:              0;
  letter-spacing:             normal;
  line-height:                100%;
  color:                      white;
  text-decoration:            none;
  text-indent:                0;
  text-transform:             capitalize;
  vertical-align:             top;
  text-align:                 center;
  width:                      45;
  word-spacing:               normal;
  white-space:                normal;
}
```

In the server scripts this style sheet can be found in the Style Sheet Items dropdown. It is called **(Button) Go Sheet.**

With this style sheet in place, the HTML description of the search control's "Go" button is considerably easier than it would have been without it. All of the characteristics described in style sheet **.bldr_button_go** can be incorporated into the button with the statement **class="bldr_button_go"**.

```
<button type="submit" accesskey="g" name="btnqo" tabindex="3" class="bldr_button_go"
onClick="return jscookie.putlastquerycookie()">Go</button>
```

8c) Javascript is really the "*magic bullet*" that makes CHT-style *client side page creation* possible. Without it you'd be forced into making your users download some kind of browser plug-in to, say, make the browser Java Language aware or you'd have to stick to constructing your pages server-side. Browsers are already capable of executing (or interpreting) JavaScript. That capability is built right in, so there's nothing extra that needs to plug in and go wrong.

JavaScript is also a "safe" language, since it has no file access (other than those little memory files called "cookies") and it cannot execute outside the context of the DOM (Document Object Model) in which it is enveloped. You really couldn't write a virus with the JavaScript language, unless of course you consider web page pop-ups a virus. They may well be that, but at least they can't do harm to your system other than clogging your drive cache with junk. JavaScript just can't take control of your system or see into its internals the way languages like Java, C++, C#, VBScript can do so that JavaScript can't be used against you.

Recently, Microsoft has agreed to incorporate pop-up suppression into its I.E. browser so that even annoying pop-ups will soon be a thing of the past. Of course, pop-ups have legitimate uses too, so that your scripts will need to avoid things like pop-up-window menus.

JavaScript is a valid, useful and powerful computer language. It can make decisions with IF and CASE. It can repeat instructions with FOR loops and much more. It is possible to combine data and code into "Objects" in such a way that makes ***JavaScript Data Objects*** one of the very best ways to package data being sent from a server to a browser. Data packaged this way is considerably more compact, and immediately useful, than data packaged as XML. With **JavaScript Data Objects** it is possible to combine

the data and the subroutines to validate that data in *one neat little package*. **JavaScript Data Objects** are compact and describe the data only, not the look of the data. So data and data validation code is *completely separated* from the code (the HTML + JavaScript) that describes how your data might look in a browser window.

One big bonus for the future, JavaScript has been specified to be added to all mini-browser operating systems like Palm OS and Windows CE. We've tested HNDMTSNG.EXE pages on a Sony CLIE TH55 which uses the Palm OS5 with the NetFront V3.x browser. The scripts work perfectly so that aside from a too-wide page format that forces a lot of left-right scrolling on the CLIE, we can access, browse and interact with the data pages produced by this server. To accommodate the 320 x 480 CLIE screen, conditional script code testing the value returned from the jsutil.getbrowsertype() script can cause narrower page renderings by substituting a different style sheet file designed for that screen format. And of course, that's strictly a script issue, we don't need to touch server code to make that happen.

EXERCISE 9:
Modify the introductory text, on your server's home page, to reflect the theme of your particular NG website using the built in "Server Scripts" editor.

The design of the HNDMTSNG.EXE server is such that at the "Jump Start" level of site construction, the amount of interaction with the scripts is minimized. This allows you to get the server up and running out of the box with a minimum of fuss and difficulty - given that we're working in an *extremely* fussy and difficult area of computer software development. When you develop your own server applications, which we will do in the course of this *Web Application Training* exercise, you can opt to use this jump start approach with your servers or not depending, I suppose, on whether only you are going to be interacting with your server or whether you intend to have a lot of different people (who will need training) interacting with your server.

The concept of "Server Variables" makes your web page code more abstract and probably harder to understand. But it makes it far more flexible and less work to maintain - at least up to a point. For instance, the server page code mentions your company name approximately 20-25 times on about 14 different pages. We could have hard-coded the company name into our pages. That would have been easier for us. But instead we used a server variable called **system.systemservercompany** to store the company name so that you can indicate your company name *once* in one convenient place (see server variables editor image below).

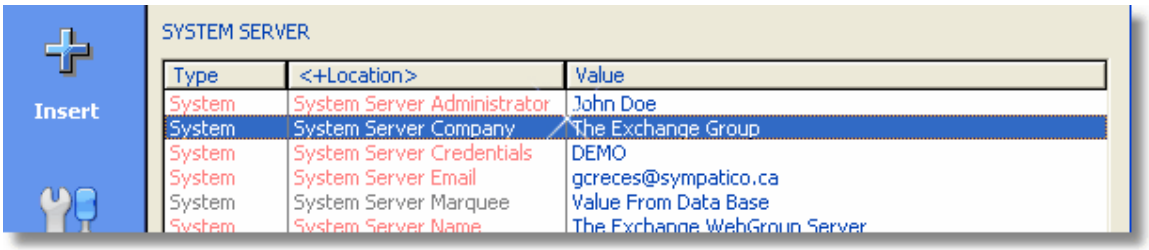

Illustration 2.1

You may have noticed by now that server variables that constitute the configuration data about your server appear in three columns: **Type**, **Location** and **Value**. When you use a server variable in code, you will address it as **type.location**. The variable **system.systemservercompany** is from the type column **system** and from the location column **System Server Company**, with the spaces removed and all lower case.

JavaScript is case sensitive so it's best not to mess around with mixed case variable names or you'll wonder if the variable was defined as System.SystemServerCompany, or System.SystemserverCompany and so on. This way, all lower case there's no guesswork. You can add server variables yourself and add

them into the existing type classes. When you define them by inserting a value, that value will be available on all pages to be inserted as a variable.

The main value of server variables is to define values that are reused often to save you from having to hard code each in 20 different places on 14 different pages - a real *Jump Start* for the new user. We asked you in question 9 to modify home page text in such a way as to reflect the intended use of your server. This text only appears in one place, on the home page, so there would have been little benefit storing it in a server variable.

There are some cool features buried in this server variables browse/form that we'll tell you about later. You may at present get the impression that the build-in script editor is really only useful for use with a Web Group Server like HNDMTSNG.EXE. If you did think this, give your head a shake and take some imagination pills. The entire configuration and script system is a set of data tables. That makes them able to accommodate any kind of server of this type that you write. Consider it your front-end builder for **Client-Side Pages** instead of server-generated pages.

While the HTML scripts provided with HNDMTSNG.EXE are specific to our Web Group Server's intended use, many of the JavaScript subroutines, email scripts and style sheets apply perfectly well to servers with different intended uses. And of course, the script creation system is abstract so that it applies and can be incorporated into any CHT 3rd-generation server.

ABOUT DYNAMIC WEB PAGES

A dynamic web page should meet one or more of the following criteria:

- **Interactivity.** A dynamic website should adapt and react to the visitor's actions as quickly as possible.
- **Synchronicity.** A dynamic website should bring together relevant information and activities from a variety of sources with a minimal amount of searching on the part of the visitor.
- **Flexibility.** A dynamic website should give the visitor a flexible means to find information so that they can choose or determine a search that best suits their informational needs.
- **Adaptability.** A dynamic website adjusts to cater to the individual users needs. Most often this adjustment is done at the client (browser) side via dynamic HTML (DHTML) and JavaScript.

WEB PAGE CREATION TECHNIQUES

Server-Side Page Creation: IIS (Internet Information Server) And Other Generic Web Servers

Normally, dynamic-page creation is done via server-side scripting. Since web servers like IIS are generic, they are not built to do anything specific. They require a script to follow that tells them what to do. An IIS server, among other things, is a run-time interpreter of these scripting languages, either directly or via plug-in DLL.

Run-time server-side scripting (not to be confused with client-side scripting) makes all servers of this type somewhat vulnerable to rogue scripts. And this vulnerability has spawned a huge industry of products to help thwart illegal scripts, things like fire walls and routers and port blockers - though, admittedly these products have other uses too. Someone with malicious intent could conceivably write a script, force it down your IIS server's throat, and exploit your hard disk information or your network.

There is a wide variety of "back-end" scripting methodology, for instance CGI, Perl, ASP, PHP and Java to name a few. All of these scripting systems are similar in one respect. They're designed to tell a generic server like IIS what it should do to construct a web page inside the server.

Client-Side Page Creation: IIS And Other Generic Web Servers

This technique has always been practiced to some extent or other even with IIS and similar servers. If it hadn't come up again and again as a requirement, there wouldn't have been the need to invent JavaScript. Since HTML offers almost no programmatic control of your web page, JavaScript was added to drastically improve the flexibility of page rendering.

JavaScript can also considerably reduce server load, since it enables data validation at the client side, without having to make a trip back to the server just to qualify that a required field has been left blank. Suffice to say, however, that even today, the business of getting information from an IIS-style generic server in order to have something for HTML and JavaScript to work with, still involves primarily back-end scripting. Those back-end scripting mechanisms might produce HTML, XML, DHTML, HTML mixed with JavaScript, and you name it, depending, of course, on the quality of the script and the demands of the web application.

Server-Side Page Creation: CHT Dedicated Servers (Generations 1 And 2)

Server-side web page creation on a CHT server does not involve back end scripting - at least in the sense of a run-time interpreted script such as those mentioned above. Your page creation code is clarion code, found in our HNDHTML classes called into by more Clarion code generated by our templates plus any embed code added by you.

The outcome is a compiled dedicated-purpose application that wraps the data bases designated by you on the CHT **BrowserServerHTMLBuilder** interfaces. Such a server is not capable of executing server side scripts, and is thus not vulnerable to them. This server only delivers the data you designed it to deliver in the way you asked it to be delivered within the limited design scope offered by the templates.

Note that this is what we often refer to in these lessons as CHT Generations 1 and 2 of our web product. This technique is fine, and requires less knowledge of HTML and JavaScript, but it's very limiting from a web page design point of view.

Client-Side Page Creation: CHT Dedicated Servers (Generation 3, Build O8A1.0 Forward)

It's the *limited* nature of server-side page design that led us to build Generation 3, of our Browser Server technology. In fact, building a CHT server in Generation 3 CHT is a lot easier since you're not really forced to think about screen design and all the picky little details during the web server creation stage. You're only concerned about making sure that the data you want to display gets to the client and the data you don't want to display doesn't.

The mechanism for determining who can access the data and who can't remains the same. Restated in another way, when the CHT HTML classes wrapped the pages server-side you kind of had to settle for some functional, but fairly rudimentary screen designs. There was no screen design mechanism server-side that allowed you to modify the web layout to your heart's content. With full, client-side page creation we've turned the problem on its head. Here's what happens now - in Generation 3.

JAVASCRIPT DATA OBJECTS

Requested data is wrapped in the form of JavaScript Data Objects.

A **JavaScript Data Object** is not a page or form design with your data plugged into it. That's what HTML does. This is a JavaScript class that describes your data and makes it available for one hundred percent programmatic control over the data when it gets to the browser.

Below is an example of what one of these things looks like. Since JavaScript can write HTML, DHTML, XML, XHTML, you can begin to understand that when you get one of these packages, it's possible to design a page, inside the browser to display it. What that page looks like is up to the script writer.

What a Generation 3 CHT server does is get you the information in the *single most flexible form possible*. It made no pre-judgments about what the data would look like on the screen.

```
function obj_sig() {
   this.address = "harvey@muskoka.com" ;
   this.emailaddress = "harvey@muskoka.com" ;
   this.first = "Harvey" ;
   this.last = "Strachan" ;
   this.name = "Strachan, Harvey" ;
   this.company = "The Strachan Company" ;
   this.timeslogged = "246" ;
   this.allowsendmail = "Allow" ;
   this.website = "http://www.muskoka.com" ;
   this.loggeddate = "9/26/2003" ;
   this.lastvisit = "9/26/2003" ;
   this.maildate = "8/28/2003" ;
   this.readonly = "0" ;
}
var sig = new obj_sig();
```

Your Page Scripts Use The Server's JavaScript Data Objects As Variables In Page Design.

This particular object is what we call the "Signature" object in our Web Group Server. It's available to your scripts full-time once a user logs in. All of the "Properties" of this object are available as **sig.address**, **sig.emailaddress**, **sig.first**, **sig.last**, **sig.name**, **sig.company**, **sig.timeslogged**, **sig.allowsendmail**, **sig.website**, **sig.loggeddate**, **sig.lastvisit**, **sig.maildate** and **sig.readonly**. What's packaged inside this object was decided on the CHT server design templates, so its fully under developer control when you build your own server.

Notice there's nothing in here like a CSS sheet or an HTML tag to describe what the data should look like on your browser window. It's a pure data package. Here's an example of a server-side script that makes use of this information.

```
jssignature.takesignaturebutton() {
  var snip = document.forms[0].bod_message.value.indexOf(text.snipmarker,0) ;
  var crlf = unescape('%0D')+ unescape('%0A') ;
  var haswebsite = crlf + sig.website ;
  if (sig.company != "" && sig.company != text.notapplicable) {
     var hascompany = crlf + sig.company;
  }
  else
  {
    var hascompany = "";
  }
  if (snip !=-1 && snip !=0) {
     document.forms[0].bod_message.value =
      document.forms[0].bod_message.value.slice(0,snip-1)
     + sig.first + ' ' + sig.last + hascompany + haswebsite + crlf + crlf +
     document.forms[0].bod_message.value.slice(snip,
     document.forms[0].bod_message.value.length);
     return document.forms[0].action ;
  }
  document.forms[0].bod_message.value = document.forms[0].bod_message.value +
  crlf + sig.first + ' ' + sig.last + hascompany + haswebsite + crlf;
  return document.forms[0].action ;
}
```

In the server scripts this JavaScript can be found in the JavaScript Items dropdown. It is called **(JSSIGNATURE) Take Signature Button Script.**

Have you figured it out? Perhaps the function name gave you the necessary clue. This JavaScript function (or in OOP parlance, this method) is hooked to the signature button on the messages edit form. When you click on that button this method inserts your name, company and website into the web edit form somewhere above the <SNIP> marker. Even the <SNIP> marker is stored in a variable called **text.snipmarker** so that it can be configured to be something other than <SNIP> by changing server variable "Snip Marker".

SERVER VARIABLES

Script Design Is Kept As Easy And Intuitive As Possible With Pop Up Selection Lists

In the Script Editor, pop up selection lists are provided to tell the script developer which variables are available for use in his scripts. The menu sequence is:
Insert -> JavaScript Variable -> Back end Data Base Variables -> Signature Variables.

Illustration 2.2

These selection lists are built from the configuration data base. If you write an application that uses different sig variable names, or more or less of them, just add or delete to the Config data base attached to the HNDMTSNG.EXE, **hndmtsngcfg.tps**.

Id	Item Type	Item Location	Item Value	Edit All	m Help In	isplay Fla	Date Modified	Time Modified	Version Id
185	sig	address	Value From Data Base	0	internal	0	73,958	7:52 AM	Alpha 1.01
186	sig	first	Value From Data Base	0	internal	0	73,958	7:52 AM	Alpha 1.01
187	sig	last	Value From Data Base	0	internal	0	73,958	7:52 AM	Alpha 1.01
188	sig	company	Value From Data Base	0	internal	0	73,958	7:52 AM	Alpha 1.01
189	sig	timeslogged	Value From Data Base	0	internal	0	73,958	7:52 AM	Alpha 1.01
190	sig	allowsendmail	Value From Data Base	0	internal	0	73,958	7:52 AM	Alpha 1.01
191	sig	website	Value From Data Base	0	internal	0	73,958	7:52 AM	Alpha 1.01
192	sig	readonly	Value From Data Base	0	internal	0	73,958	7:52 AM	Alpha 1.01
233	sig	name	Value From Data Base	0	internal	0	73,958	7:52 AM	Alpha 1.01
234	sig	maildate	Value From Data Base	0	internal	0	73,958	7:52 AM	Alpha 1.01
235	sig	lastvisit	Value From Data Base	0	internal	0	73,958	7:52 AM	Alpha 1.01
284	sig	emailaddress	Value From Data Base	0	internal	0	73,958	7:52 AM	Alpha 1.01
286	sig	loggeddate	Value From Data Base	0	internal	0	73,958	7:52 AM	Alpha 1.01

Illustration 2.3

There's no automatic relationship between the variable names that you send up from the server and the names in this data base. These names were manually added to match the variable names we programmed the NG server to send in order to act as a pick list, memory assist for the script developer.

Configuration-Related Server Variables May Be Added For Use In Your Scripts

Since you don't have the source application for the HNDMTSNG.EXE there's not a lot you can do to make it send more data fields to you. Unless you build your own server, which we will do before this training course is complete. But server configuration variables can be added, as required, by your page and server designs. Here's how. From the server interface, execute these menu steps:
Config -> Server Variables -> Insert Button (+) -> Select Type -> Enter Location -> Enter Value -> Enter Help Information.

Now, before you click the Ok button to save this and leave the update screen. There's a hidden keystroke you should know about. It's Ctrl-1. This pops up a set of radio buttons:

Illustration 2.4

This internally categorizes your server variable to tell the server how it should be handled.

- **Hide** - makes this an internal (Value From Data Base) variable like the **sig** variables above. This will let you create a server variable that other users can't get at and change or has no local value. This does not make the server send up other data base variables that it is not now sending up. But it does let you add to the variable pick lists available in the editor if you were writing a back end server of your own and were sending up an entirely different set of back end variables.

- **Jump Start** - makes this a variable of higher priority than other server variables.

- **Standard** - standard server variable, normal priority.

- **User** - this is the only flag assigned to server variables that are created by someone who doesn't know about the CTRL-1 keystroke.

This behavior suits our purposes with the HNDMTSNG.EXE server at the moment. In later lessons you will be given your own copy of the HNDMTSNG.APP and can change the behavior to suit your needs.

SUMMARY OF SERVER VARIABLES AVAILABLE FOR USE IN SCRIPTS

Remember as you read this that what is being discussed below is pertinent to the HNDMTSNG.EXE Web Group Server. With a server of your own construction, you can make it present whatever you want to suit your design. Here we'll discuss what's presented for the HNDMTSNG.EXE server since you have a working example of that server to practice with. Some variable names listed below may not agree entirely with what you find in new versions of the demo Web Group Server as it too may evolve somewhat over time.

A) SERVER CONFIGURATION VARIABLES

This set of variables can be seen and listed from the script editor using the menu sequence *Insert -> JavaScript Variables -> Server Configuration Variables*. All of these are either in the **Jump Start**, **User** or **Standard** categories as explained above. The values assigned to these variables are under server manager control via the Server Variables interface. The point of this group of variables is to save you having to hard code certain frequently used values such as your company name into the server pages, as well as to provide configuration settings that are then used in your pages to make decisions about what might be displayed or not displayed under a give condition.

Button Variables

These are button labels used in the scripts. Examples are **button.browsecategorybuttontext** and **button.signaturebuttontext** there are a number of these and you can add your own to this category, or change the value of the existing ones. If you delete any of the one's we've given you, make sure we're not using it somewhere in one of our scripts or you'll break that script until you change it. The script editor has a good Global-Find feature that lets you search the entire script set to see if something is in use or not.

Choices Variables

These are messages popped up when the user has to be asked to make a choice, such as save a record, for example **choices.okaytosavemessage**.

File Variables

These are the .JS and .CSS file names generated via the screen generation component built into the HNDMTSNG.EXE server. The JavaScript configuration file is named in **file.javascriptconfigfile** as hndmtsngcfg.js. You have the power to change this file name or to reference the file name in one of your scripts.

Help Variables

The HNDMTSNG.EXE server does not have any of these in it's data base but there's nothing stopping you from adding them in if you see fit.

Image Variables

These are the names of image files used often in the server scripts. The main header image for instance is, **image.headerimage** and its dimensions are **image.headerimageheight** and **image.headerimagewidth**.

Keyword Variables

These represent the search keywords (like CONTAINS, STARTSWITH) that are used in web queries as well as the ORDER BY keyword.

Examples are **keyword.defaultcontainskeyword** and **keyword.defaultstartswithkeyword**. This lets the server manager change the query language used on the server, to say French or Dutch by changing the definitions of these variables. The server looks after binding these newly translated keywords to the correct back-end function so that the HNDMTSNG.EXE server can understand any language you throw at it. (NOTE: Keyword definitions must not contain spaces with the exception of the ORDER BY keyword).

Link Variables

Link variables represent URLS or web links that you're likely to use at least more than once in your scripts. If a link then changes after you've written your scripts, you can change it in one place without having to go looking for it in a dozen places. Examples are **link.serverurl** and **link.contactlink**.

Mail Variables

Mail variables are email addresses defined in your server variables. Or mail-related bits of information such as the name of the currently defined default registration email message.

Examples are **mail.mailfromaddress** and **mail.registrationemailplaintext**.

Menu Variables

These are the text labels of menus that appear in your server pages. Examples are:
menu.backpagemenu, **menu.homepagemenu**.

Prompt Variables

Prompt variables contain the labels of prompts used in your update and entry forms. Examples are:
prompt.companyname, prompt.emailaddress.

Query Variables

These contain short forms for query components such as TODAY()-7, TODAY(). This is defined in the HNDMTSNG.EXE server variables file as THISWEEK and is used in a query such as DATE RANGE THISWEEK. When the query variable THISWEEK is used in any query the server expands it to it's real value as defined in the server variables **Value** field. You may add as many of these as you wish, as long as they're defined as type **Query** the server will expand them correctly for you. Examples are **query.now** and **query.thismonth**.

System Variables

This is a catch-all category of configuration values such as **system.messagecategoryreplydefault** the default value used in the category field of a message when it's a reply. Most of these would be used in a JavaScript IF or CASE statement to help you make some kind of branching decision.

Text Variables

Again a catch all for text strings used often enough in your pages to warrant being defined. For example **text.snipmarker** defined as <SNIP> to act as a separator between the initiating message and your reply to it.

A) BACKEND DATA

This set of variables can be seen and listed from the script editor using the menu sequence:
Insert -> JavaScript Variables -> Backend Data Base Variables.

All of these belong to the **Hide** category as explained above. That means they are hidden from the server variables browse and update form, since the values for these cannot be assigned there. You can, however, make them appear in the browse for editing purposes, again using the Ctrl-1 keystroke. The values delivered by these variables come from the server's data bases.

The point of having the names available is to provide you with pick lists when writing scripts so that the script writer doesn't need to strain to remember what they are. We remind you here that you can create and hide these variables from the server variables interface. You can't force the server to send you a specific variable from the back end if it isn't already doing so. *That's not to say this kind of feature could not be introduced by some smart CHT server builder.*

Message Browse Data (brwmsg[datarow]):

Message browse data comes up as a Javascript Data Object like all other data. It has the added characteristic of being stored in an array, one array slot for each row in your browse. The size of this array - the number of rows in your browse - is determined by the query entered by the user, up to a maximum number of rows in a server variable called **system.maximumqueryresultset**. The first data row number is defined as **ndx.msgstart** the last row a **ndx.msgcount**.

These **ndx** values come up as part of the data package describing the query result set. To discourage users from dragging down your server's performance with stupid queries that ask for the whole data base, keep the **system.maximumqueryresultset** value set below 150. (Google keeps its search results to 100 matches.) That will keep user's brains sharp and your headaches minimal.

If you look at the script called, **(PAGE) 11. Messages Browse Page HTML,** you will see a FOR loop that iterates through the messages browse data array to populate the messages browse data to that page. Here the code is truncated somewhat for brevity. You can always examine the full browse code more closely in the server scripts editor. *(Code may vary in later version servers.)*

```
for (var datarow = ndx.msgstart; datarow <= ndx.msgcount; datarow++) {
   if (datarow % 2 == 0) {
      thisrow = '<tr class="bldr_greenbar">' ;
   } else {
      thisrow = '<tr class="bldr_row_container">' ;
   }
   document.write(thisrow)
   document.write('<td class="bldr_browse_messages_date">' +
     unescape(brwmsg[datarow].boddatelogged)   + '</td>');
   document.write('<td class="bldr_browse_messages_date">' +
     unescape(brwmsg[datarow].bodtimelogged)   + '</td>');
   document.write('<td class="bldr_browse_messages_size">' +
     unescape(brwmsg[datarow].bodmsgsize)      + '</td>');
   document.write('<td class="bldr_browse_messages_name">' +
     unescape(brwmsg[datarow].bodname)         + '</td>');
   document.write('<td class="bldr_browse_messages_category">' +
     unescape(brwmsg[datarow].bodcategory) + '</td>');
   document.write('<td class="bldr_browse_messages_subject">'  +
     unescape(brwmsg[datarow].bodsubject)  + '</td>');
   document.write(vieweditreplybutton) ;
   document.write(threadbutton) ;
   document.write('</tr>') ;
}
```

In the server scripts this code can be found in the HTML Items dropdown. It is called **(PAGE) 11. Messages Browse Page HTML.**

Members Browse Data (brwmem[datarow]):

Members browse data comes up as a Javascript Data Object like all other data. And because it's used to construct a browse, it too uses an array, one array slot for each row in your browse.

When you look at the script called, **(PAGE) 12. Members Browse Page HTML,** you will see a FOR loop that iterates through the members browse data array to populate the browse data to that page. Here the code is truncated somewhat for brevity. You can always examine the full browse code more closely in the server scripts editor. Notice that separate style sheets (class="xxx") are being applied to each of the columns allowing for maximum flexibility in describing the width, color, font and so forth, on any column entity.

```
for (var datarow = ndx.memstart; datarow <= ndx.memcount; datarow++) {
  /* ALTERNATES BROWSE ROW COLORS EVERY SECOND RECORD */
  if (datarow % 2 == 0) {
     thisrow = '<tr class="bldr_greenbar">' ;
  } else {
     thisrow = '<tr class="bldr_row_container">' ;
  }
  document.write(thisrow);
  document.write('<td class="bldr_browse_members_status">'   +
    unescape(brwmem[datarow].regstatus)        + '</td>');
  document.write('<td class="bldr_browse_members_date">'     +
    unescape(brwmem[datarow].regdatelogged)    + '</td>');
  document.write('<td class="bldr_browse_members_date">'     +
    unescape(brwmem[datarow].regtimelogged)    + '</td>');
  document.write('<td class="bldr_browse_members_name">'     +
    unescape(brwmem[datarow].regname)          + '</td>');
  document.write('<td class="bldr_browse_members_company">'  +
    unescape(brwmem[datarow].regcompany)       + '</td>');
  document.write('<td class="bldr_browse_members_website">'  +
    unescape(brwmem[datarow].regwebsite)       + '</td>');
  document.write('<td class="bldr_browse_members_visits">'   +
    unescape(brwmem[datarow].regvisits)        + '</td>');
  document.write('<td class="bldr_browse_members_allow">'    +
    unescape(brwmem[datarow].regallowsendmail) + '</td>');
  document.write(vieweditmailbutton) ;
  document.write('</tr>') ;
}
```

In the server scripts this code can be found in the HTML Items dropdown. It is called **(PAGE) 12. Members Browse Page HTML.**

Message Body Data (bod.):

The individual fields of the message body are available on the message update form. While the message body information provided includes ten fields in all, some fields are provided solely for informational and programmatic control reasons. Three fields only (shown in orange), are accepted by the server for update from the web message insert/update form.

Available fields are: **bod.ngmemberid, bod.updated, bod.id, bod.ngthreadid, bod.datelogged, bod.timelogged, bod.msgsize, bod.category.value, bod.subject.value, bod.message.value**.

In another lesson we will go into more detail about how the data object describing this data is structured. Editable fields (the orange ones) come up with considerably more information attached to them including some field-specific validation information constructed by the server-side object builder from your Clarion data dictionary.

Registration/Member Body Data (reg.):

The individual fields of the registrant/member record are available on the member update form. While the member record information provided includes eleven fields in all, some fields are provided solely for informational and programmatic control reasons.

Six fields only (shown in orange), are accepted by the server for update from the web member update form. Available fields are **reg.first.value, reg.last.value, reg.company.value, reg.website.value, reg.allowsendmail.value, reg.email.value, reg.present, reg.datelogged, reg.timelogged, reg.name, reg.timeslogged**.

In another lesson we will go into greater detail about how the data object describing this information is structured. Editable fields (the orange ones) come up with considerably more information attached to them, including some field-specific validation information that's passed up by the server-side object builder from your Clarion dictionary.

Server Flags (flg.):

Some of these flag variables are used in commands sent to the server to indicate certain actions should be taken by the server when a data query is sent to it.

The available flags are: **flg.ask, flg.noask, flg.actionbrowse, flg.actionedit, flg.actionchange, flg.actioninsert, flg.actiondelete, flg.actionreply, flg.actionemail, flg.actionrecycle, flg.actionprint, flg.askokay**. In another lesson, we will go into greater detail what these flags are for and how they're used in your page scripts.

Email Send Data (ml.):

From the registrants/members browse, members are allowed access only to their own member data. They can edit and change the **reg** fields shown above in orange, when the member record they are accessing is their own. When another member's record is accessed, the server presents an email send form.

The fields available when this email form appears are: **ml.subject.value, ml.body.value**. Any other information about the person receiving the email is presented on the email form as not editable, and comes up as a standard **reg** record described above. Both orange **ml** fields are editable. The server instruction to put these records back includes a **flg.actionemail**. The server-supplied function that comes included with this email data, called **emailrecord()** looks after adding this flag for you.

```
function emailrecord() {
  if (document.forms[0] != null) {
   if ((flg.askok == flg.noask) || (confirm("Okay to send this message.\n"))){ ;
      document.forms[0].editaction.value = flg.actionemail   ;
      return true ;
   }
  }
  return false ;
}
```

In another lesson, we will go into greater detail about what the server provides by way of data packages as well as functions for data validation and data update. We'll deal with these first as "consumers" of the available information. The server administrator or the individual writing scripts can be defined as a consumer, since he/she uses what's provided by the server in order to create scripts that display and interact with the information.

Then, when you have a good grounding in how the server packages information and support functions for that information, we'll dig into the internals of a server application that you'll build, to see how, using the

CHT templates, you are able to build these data packages like the ones now being sent up by the HNDMTSNG.EXE Web Group Server.

Signature Data (sig.):

Earlier in this lesson we discussed signature or **sig** data in the context of explaining what a JavaScript Data Object looks like. This is purely "read" data not "write" data and is supplied by the server on every page once a member logs in. You cannot on the home page, for instance, greet the individual visiting that page, with a "Hi, Bob. Welcome to my website..." since you don't yet know who he/she is, until they log in.

Once they have logged in, then **sig** information is available to directly address information about the logged-in individual.

Fields supplied are: **sig.address, sig.emailaddress, sig.first, sig.last, sig.name, sig.company, sig.timeslogged, sig.allowsendmail, sig.website, sig.loggeddate, sig.lastvisit, sig.maildate, sig.readonly**.

LESSON 2 - SUMMATION

In this second lesson, we explained the difference between HTML pages constructed at the server end and HTML pages constructed at the browser end, inside the browser, using JavaScript. We showed you how the HNDMTSNG.EXE Web Group Server bundles JavaScript Data Objects which can be used in browser-based scripts to construct pages.

We also explained how the built-in script editor is used to construct scripts and how the server gets these scripts to the browser. We explained how data packages come to the browser and how they differ from script packages. Finally we gave you an overview of the server configuration variables, and back end data variables available for script building.

LESSON 2 - EXERCISES

1) Explain the mechanism your I.E. browser uses to tell the server that it has a particular file or image already in the its cache and to send it only if it has changed from the one the browser already has there. Copy a piece of the server log into your answer to indicate how the server responded in this situation.

2) Create your own style sheet using the "Server Scripts" editor in your HNDMTSNG.EXE server. Give it a name of your choosing and paste that style sheet into your answer for us to see. Then use your style sheet on some field, button, menu or text in your web server and explain in your answer what page this modification is on again for us to see. Lastly, explain what steps you used to create your own style sheet in the "Server Scripts" editor. Including an explanation of how the pop up assists work when you're in the style sheet portion of the editor.

3) Add two server variables of your own, a **system** variable (make it a "jump start" variable so that it appears pink in the server variables list) and a **link** variable. Add some code to your server home page script **((PAGE) 01. Home Page Text HTML)** to display the information stored in these two server variables. In your answer, provide the names of these variables **system.yournamehere** and **link.yournamehere** and the data they contain so that we can see them in place on your home page.

4) Log onto your own HNDMTSNG.EXE server and right click the first page that comes up after you log in (the messages query page). Right click that page and select "View Source" and find the name of the server data file that accompanies that page. Find this file in your web cache, open it and isolate the **sig** data object. Copy the sig object to the clipboard and paste it in with your answer. When you've done that successfully, explain the steps you took in order to get to that sig data package in your web cache.

That's all for this lesson.

Have fun!

Gus M. Creces
The Clarion Handy Tools Page
support@cwhandy.com