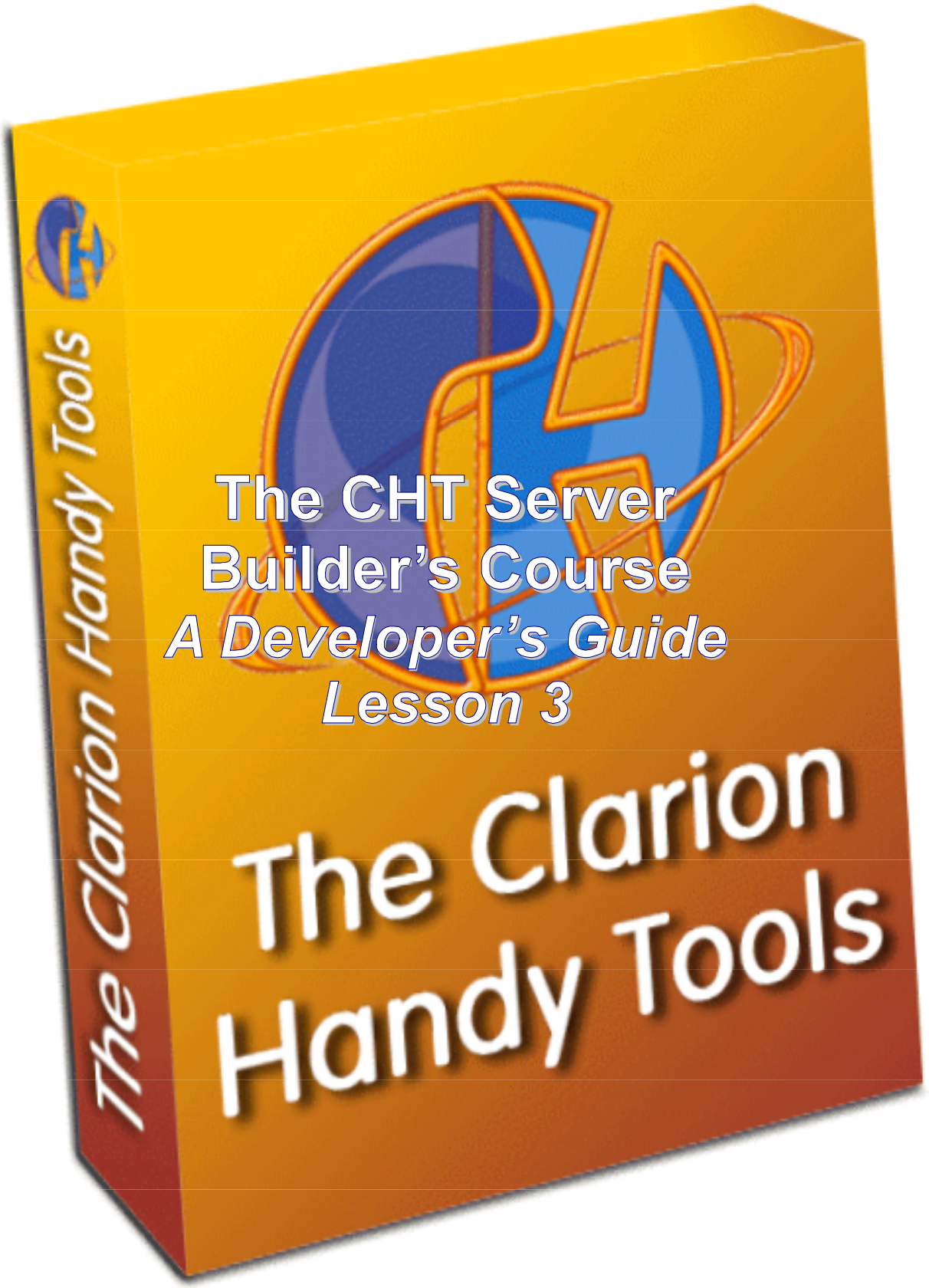

The Clarion Handy Tools
The CHT Server Builder's Course
A Developer's Guide
Lesson 3
The Clarion Handy Tools

INTRODUCTION

Here are somewhat more lengthy responses to Lesson 2 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 covers in considerable detail, every component of a HNDMTSNG.EXE server web page, explaining as we go how these web pages incorporate the various pieces - page scripts, subroutines and style sheets - provided by the server via the server scripts editor. The aim of this lesson is to help you understand that the scripts in HNDMTSNG.EXE are 100% under your control and to encourage you to begin to exercise some control over them. Ultimately this should help you, in upcoming lessons, when you create a back-end server of your own design, to achieve greater control of what happens at the front end, where your scripts are executing.

REVIEW LESSON 2 EXERCISES

EXERCISE 1: Explain the mechanism your I.E. browser uses to tell the server that it has a particular file or image already in the it's 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.

When the browser has been configured to **"Check for newer versions of stored pages -> Every visit to the page"** it makes what is, in effect, a conditional file request by indicating to the server to send a new copy of any given file only if it has one that's newer than the one it already has in its cache. In fact, the browser could go ahead and display the one it has, say, an image or a JavaScript subroutines file, but before using it, it checks with the server to see if there's perhaps a newer copy of this file at the server end. If the server does not have a newer copy, it responds with **"HTTP/1.1 304 Not Modified"** to tell the browser that there's no new copy of the file and to go ahead and display the one already in it's cache.

Without this conditional mechanism, the browser would need to download every aspect of every page, completely each time a new page is displayed, even though a lot of elements in any new page are probably repeats of what was displayed in the previous page. On the other hand, if the browser unconditionally displayed all same-name items from its cache, instead of bothering to download them again, that would be far more efficient, but would make dynamic-page websites impossible. Changes would not be reflected until the browser cache was cleared. This mechanism is discussed further in the section below called, *Anatomy Of A Conditional Browser File Request*.

The following section copied from the server's visible log illustrates a conditional request that resulted in a "304 Not Modified" response.

```
HTTP/1.1 304 Not Modified
Last-Modified: Tue, 16 MAR 2004 10:57:22 GMT
Date: Thu, 1 APR 2004 23:34:58 GMT
Host: http://64.229.207.254
Filename:IMAGES\HTMH2.JPG
Socket-Number: 704
Connection: Close
Content-Type: image/jpeg
Content-Length: 0
Server: CHT DESKTOP SERVERS O8A1.4
```

This information appears in the server's visible log when the Status control is on (pushed in).

EXERCISE 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.

The easiest way to create a new style sheet is to find one in the existing set that resembles what you want your style sheet to do and then "copy" it. Here's how:

- Select any style sheet from the style sheets drop down. This could be the one you want to copy but it doesn't have to be.
- Pull down the edit menu and select "Insert Local"
- From the list of available style sheets presented, pick the name of the one you want to copy.
- A dialog appears "Okay to insert new copy of: (Class) Style Sheet Name?
- Click "Yes".
- You now have a new style sheet with identical contents to the one copied. It's "Item Title:" and "Item Name:" end in "(Copy)". To avoid a name clash. Give this new item a proper title and name and make the modifications that you want to see in the style sheet body.

If you created a style sheet called say, **".bldr_title_containerextra"** you could use it as an alternative design for titles.

```
<table class="bldr_title_containerextra">
  <tr>
    <td>
     This Is My Custom Title Text...
    </td>
  </tr>
</table>
```

Note that while most style sheet names start with a dot, the dot is not used when you actually apply the style sheet in your scripts.

The style sheet editor has a complete set of standard style sheet values available for insertion, which can save you the trouble of trying to learn or remember the possible settings for most style sheet elements. To change the setting of an existing element, for instance **font-family:**, double click font-family so that it becomes highlighted. Then right click for a list of possible settings. In the case of some style sheets where the settings are numeric values or percentages, the values given are *examples* only.

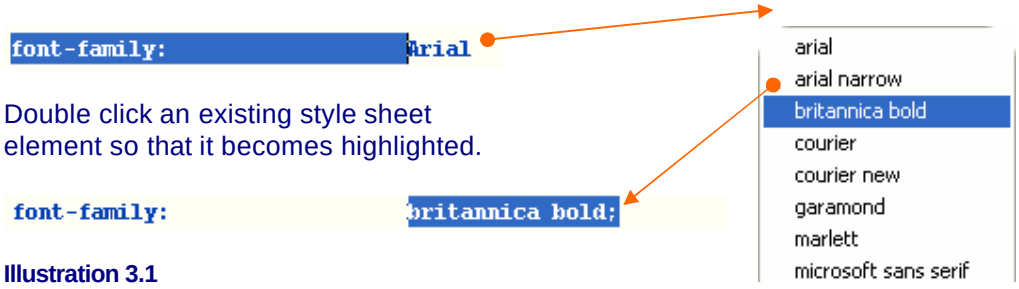


Illustration 3.1

Right click the highlighted element to obtain a list of alternative choices. When you select a new item, the results of your selection are inserted into your style sheet as illustrated above.

To insert a new style sheet element from scratch into your style sheet, create a blank line by striking the enter key at least once as illustrated below:

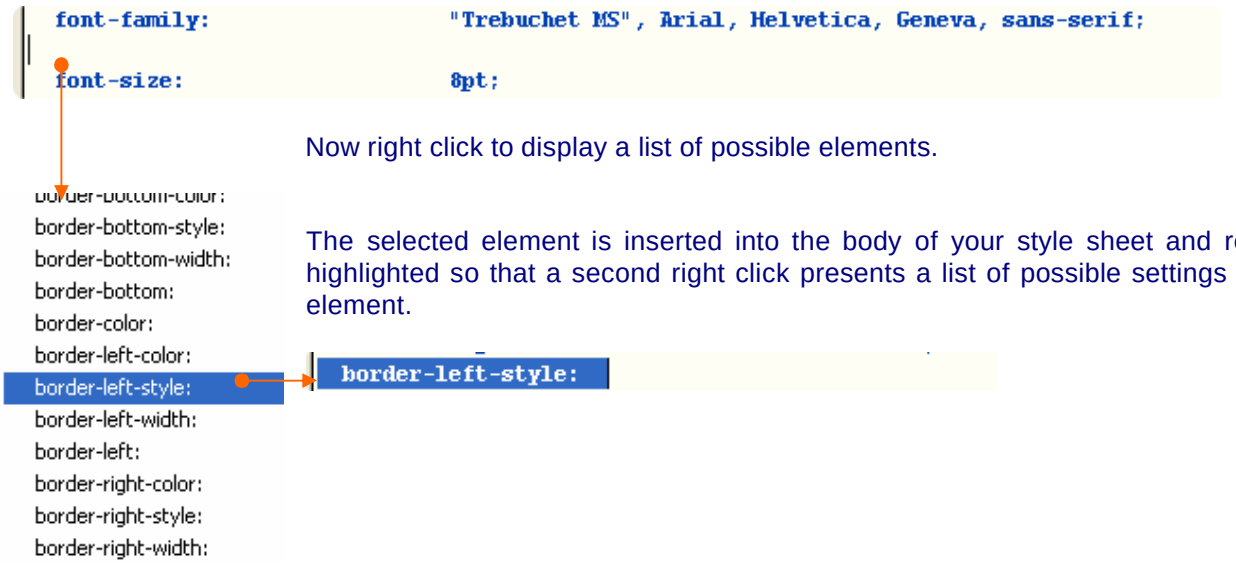


Illustration 3.2

EXERCISE 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.

To add server variables, open "*Config -> Server Variables*" from the HNDMTSNG.EXE application. Click the "+ Insert" button and select "Link" from the drop down list of possible types presented. Insert a location value that describes what this server variable does. This becomes the variable name in your page scripts (with the spaces removed). For our example, on the next page, we chose the name "Home Page Link Test One" as the location and "http://www.cwhandy.com" as the value.

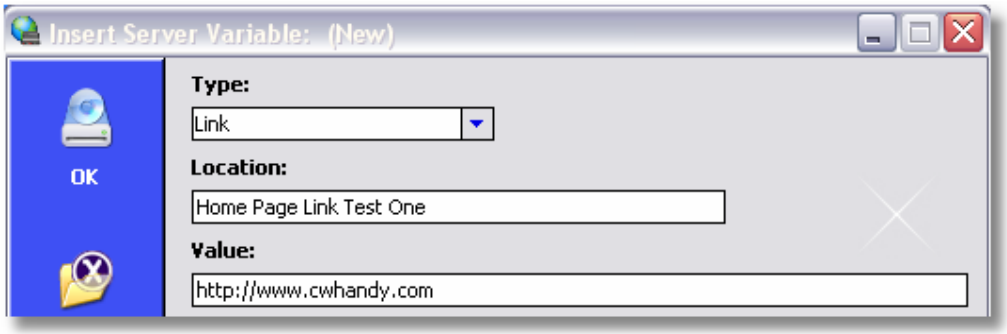


Illustration 3.3

To make this variable appear in the list of pink "Jump Start" variables when we open the "Server Variables" browse and select the "Jump Start" menu, press "Ctrl1" on your keyboard from the "Insert Server Variable:"

window pictured above. A radio control will appear, from which you should select "Jump Start". **(See Illustration 3.4)**

Illustration 3.4

You may now click "OK" to store your new server variable. This new addition is visible in the "Jump Start" list as illustrated below:

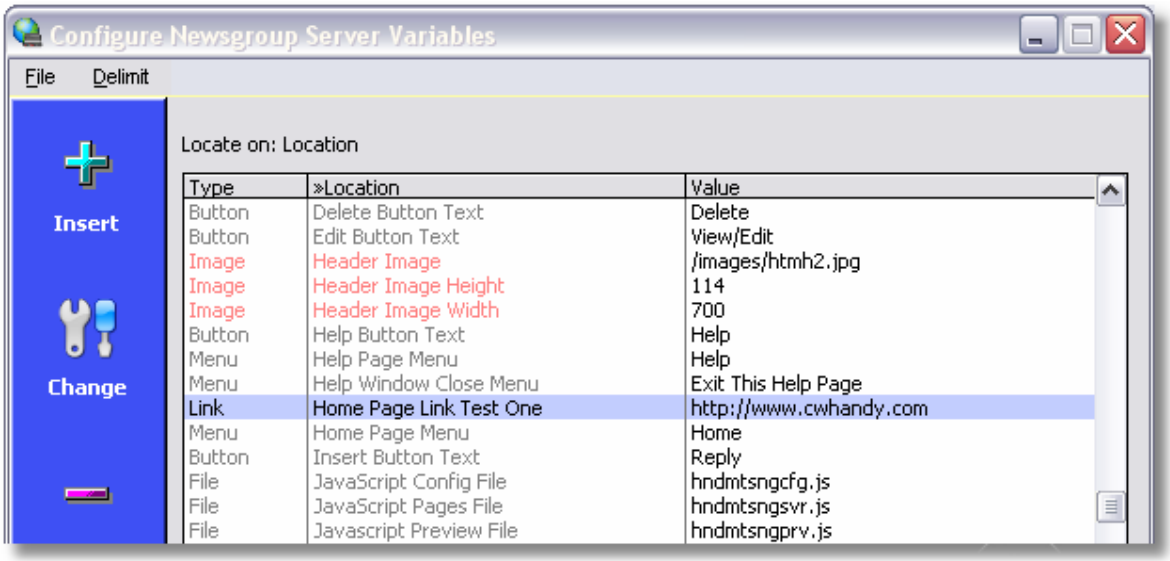


Illustration 3.5

In a script, then, we could write the following code to display this link on our web page.

```
<script>
Javascript:document.write('<a href="' + link.homepagelinktestone + '">My Test Link</a>') ;
</script>
```

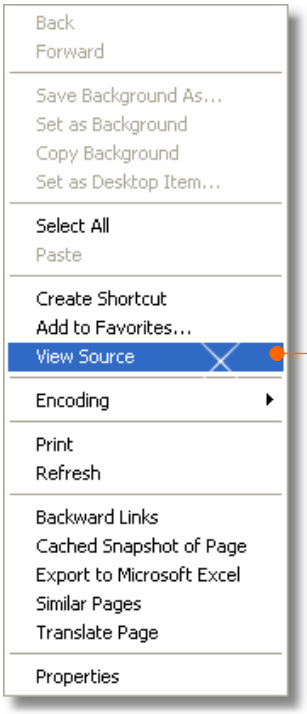
With this script component in place on several of our web pages we could change the link to our home page by simply editing the server variable called **link.homepagelinktestone** instead of having to modify our script code in numerous places. The server variables script file **hndmtsncfg.js** is generated automatically any time you exit the "Server Variables" browse. It is generated into the server's "Site Run Directory" so that the next page hit will cause it to begin downloading to the client browser to replace the one in that browser's cache.

EXERCISE 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. Figure out where that file is located on your machine, open the file and isolate the sig data object. Copy it 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 at that sig data package.

When you use the auto-login link supplied by the server via email after registration, the first page provided is the "Messages Browse". The server takes you there on the assumption you want to see the latest messages in reverse chronological order. The "Messages Query Page" is available from here via the "Messages" menu so that visitors can describe, in the form of a query, other messages they might want to see besides the latest ones.

A right click on any page yields the following chain of events:

You're presented with an opportunity to examine the "Source" code for the page that you're viewing.



The source code for either the messages browse or the messages query page don't differ all that much. That's because the differences are coded into the scripts. The server data file is the third download file in the list of JavaScript file requests. It's the one with the name that's different every time. In the example below it's:

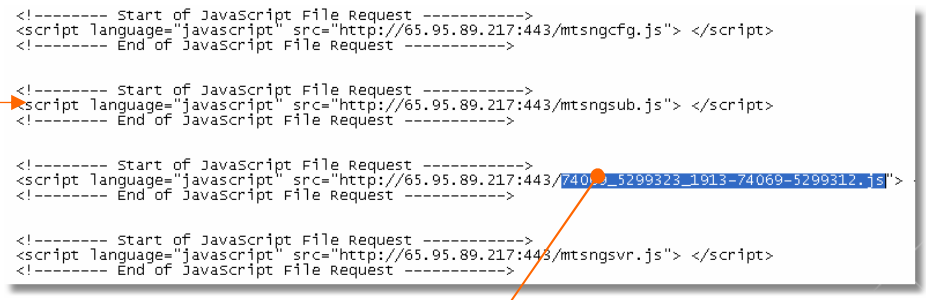


Illustration 3.6

All of these JavaScript file requests can be found in the web cache of the machine from which you're viewing your website. There are several ways to get to see these files. Here is one way to get to them:

- After right clicking your web page and selecting "View Source", highlight the file name and copy the name to the clipboard.
- Start Windows Explorer and navigate to your web cache directory. (You already know how to find the name of this).
- Once in the web cache directory, click "*Search->All Files and Folders*" and paste the file name into the top search field.
- When the correct file name appears, right click it and choose "Copy".

- Start another copy of Windows Explorer, navigate to a work directory of some sort, say c:\temp, and paste the file into that directory.
- Use your clarion editor or notepad, to open the file from c:\temp.

If you're going to repeat these steps several times for study purposes, it helps to leave the two copies of Windows Explorer running so that you don't have to restart them each time you repeat these steps with another file name.

Some of you may have tried to take the Clarion editor or notepad to open the file directly from the web cache directory. That would make a lot more sense than going through all of the steps outlined above. Unfortunately (or fortunately, depending on how you look at it) MS Windows protects this directory from prying eyes since it may contain information - in the form of cached pages - which is not meant for public consumption. We asked you to find the **sig** data package in this file, copy it to the clipboard and paste it into your answer. The package looks like this:

```
function obj_sig() {
   this.address = "gcreces@sympatico.ca" ;
   this.emailaddress = "gcreces@sympatico.ca" ;
   this.first = "Gus";
   this.last = "Creces" ;
   this.name = "Creces, Gus" ;
   this.company = "The Clarion Handy Tools Page" ;
   this.timeslogged = "1024" ;
   this.allowsendmail = "Allow" ;
   this.website = "http://www.cwhandy.com" ;
   this.loggeddate = "10/14/2003" ;
   this.lastvisit = "10/14/2003" ;
   this.maildate = "8/28/2003" ;
   this.readonly = "0" ;
}
var sig = new obj_sig();
```

This package contains signature information about the person logged in. In this case it shows my information. In your case it will display your information. We're not giving away much private information about the person logged in, since it's your own information coming back to you, and it's only being sent to your computer, since you're the one logged in. However, the fact that we were able to recover this information from the web cache is something you need to keep in mind when you design your own web servers. Since web caches will always retain a certain amount of information that you send up until the web cache is cleared. So be careful when you plan a design to send up only what you need and are willing to share.

ANATOMY OF A CONDITIONAL BROWSER FILE REQUEST

In the answer to question 1 above, we mentioned having a closer look at a conditional browser file request. Here, for example is a browser request for the **hndmtsngcfg.js**, the configuration file.

```
GET /hndmtsngcfg.js HTTP/1.1
Accept: */*
Referrer: http://65.95.86.40/GHP$
Accept-Language: en-ca
Accept-Encoding: gzip, deflate If-Modified-Since: Tue, 14 Oct 2003 13:31:40 GMT
```

Notice that the browser has included a condition: If-Modified-Since: Tue, 14 Oct 2003 13:31:40 GMT.

It only wants to see the file if it's been changed since Tuesday October 14, 2003 13:31:40 GMT. All file time comparisons are done using Greenwich Mean Time, which means that your server has to know what IT'S

time zone is in order to calculate the GMT equivalent of a file's date/time stamp found in it's RUN directory. Your HNDMTSNG.EXE server reads time zone information from your server hardware's operating system so it's important that you configure this correctly.

If the server finds that the file being requested has not been modified since the time specified by the browser, it sends back a no-content header. This is an HTTP header which contains information only, no file is attached. This information, specifically, the number **304**, indicates to the browser that the file was not modified since the requested date. The browser, on receiving a **304** message, then loads the requested file from it's cache.

```
HTTP/1.1 304 Not Modified
Last-Modified: Tue, 14 OCT 2003 17:31:40 GMT
Filename:HNDMTSNGCFG.JS
```

Now suppose the file had been modified. We can cause this to happen by opening the server variables browse and closing it again. Here is the request again, exactly the same as before:

```
GET /hndmtsngcfg.js HTTP/1.1
Accept: */*
Referrer: http://65.95.86.40/
Accept-Language: en-ca
Accept-Encoding: gzip, deflate
If-Modified-Since: Tue, 14 Oct 2003 13:31:40 GMT
```

This time the response is entirely different. The returned code is 200 OK which means the file was sent. Also included is the file length, what type of contents (text/JavaScript), the new last-modified date-time stamp in GMT time, the file name and the present date and time.

```
HTTP/1.1 200 OK
Content-Length: 10134
Content-Type: text/javascript
Last-Modified: Tue, 14 OCT 2003 14:24:59 GMT
Content-File: HNDMTSNGCFG.JS
```

On receiving a **200** code, the browser displays the newly uploaded file and drops it into the web cache to replace the one that was there. A subsequent request from the browser indicates that it is now giving the new file's date-time stamp in all requests for hndmtsngcfg.js. This same request will be repeated for hours, days or weeks until such time as the server manager once again makes a change in the configuration settings.

```
GET /hndmtsngcfg.js HTTP/1.1
Accept: */*
Referrer: http://65.95.86.40/
Accept-Language: en-ca
Accept-Encoding: gzip, deflate
If-Modified-Since: Tue, 14 OCT 2003 14:24:59 GMT
```

The separation of data content from design content makes your dynamic web pages extremely efficient since the design content, unless changed, is never downloaded again. A typical web page, with 4 or 5 fields of back-end data might contain 4-5K of design information when the HTML and data are all jumbled together. When they are separate the data component for a new page is small, and compact. Your pages display faster and your server has less upload work to do.

ANATOMY OF A TYPICAL CHT BROWSER SERVER WEB PAGE

We've already discussed in some detail the various parts of a typical web page produced by the HNDMTSNG.EXE server. Here we'll review them one by one with a brief explanation of each. The page discussed is the messages browse.

PAGE COMPONENTS FOUND BETWEEN <HEAD> AND </HEAD>

The <HEAD> area of an HTML page contains typically, all requests for support files, subroutine files and the like. It also contains the page title, optional meta-data and page-wide style sheet definitions.

```
<head>
```

THE PAGE ID COMPONENT (A <HEAD> COMPONENT)

```
<!---Page ID=(MESSAGESBROWSE)--->
```

This is the page identifier. When you right-click to view the Source Code of any page, it's the first place you should look. Each page has a unique identifier which is available in the form of a server variable called **sys.pageid**. This information can be used in a script to cause different behaviors on different pages. For instance, you might want to suppress the marquee from data pages and display it only on the home, login and register pages.

THE SERVER CONFIGURATION COMPONENT (A <HEAD> COMPONENT)

This request is for the server Config file, the contents of **hndmtsngcfg.tps** that are allowed to be shared on your web page. An example data object from this file is the **image** object that describes the header image used on all pages.

```
<script language="javascript" src="hndmtsngcfg.js"> </script>

function obj_image() {
   this.headerimage = "images/exchangewebgroup.gif" ;
   this.headerimagewidth = "700" ;
   this.headerimageheight = "114" ;
}
var image = new obj_image();
```

THE SERVER SUBROUTINES COMPONENT (A <HEAD> COMPONENT)

The next file being requested is the server subroutines file.

```
<script language="javascript" src="hndmtsngsub.js"> </script>
```

This file contains JavaScript subroutines created in the "Server Scripts" area. These subroutines are typically 100% JavaScript and they perform repetitive duties on some or all of your pages. They're packaged as JavaScript objects based on the name that you assign to them. All JSBUTTON subroutines, for example, are addressed in code as **jsbutton.somethingorother()**. Here is one button method from the jsbutton class. **(Code example next page)**

NOTE: Code has line breaks (╗) to accommodate word-wrap.

```
function obj_jsbutton() {
  this.drawcancelbutton = function(xdisabled,xtabindex) {
    btntext = '<font color="white">' + button.cancelbuttontext.slice(0,1) +
   '</font>' + button.cancelbuttontext.slice(1, button.cancelbuttontext.length);
    if (xdisabled == 0) {
      var rtnvar = '<button type="submit" ╗
       accesskey=button.cancelbuttontext.slice(0,1) id="btnquit"' +
       ' tabindex="' + xtabindex + '" class="bldr_button" ╗
       onClick="return cancelrecord()">' +
       btntext + '</button>' ;
    }else{
      var rtnvar = '<button type="submit" disabled ╗
       accesskey=button.cancelbuttontext.slice(0,1) id="btnquit"' +
       ' tabindex="'+ xtabindex + '" class="bldr_button" ╗
       onClick="return cancelrecord()">' +
       btntext + '</button>' ;
   }
   document.write(rtnvar);
 }
}
var jsbutton = new obj_jsbutton();
```

This example method draws the cancel button in one of two states, disabled or not disabled. Note that nowhere in this code does the word "Cancel" ever appear, since that value comes from server variable **button.cancelbuttontext**. Note also that the **onClick** event handler for this button calls another JavaScript subroutine called **cancelrecord()**.

THE SERVER DATA PACKAGE COMPONENT (A <HEAD> COMPONENT)

The next file being requested has been discussed at length. It's the **data package** for this page. Since this is the messages browse page this package contains array data for the browse.

```
<script language="javascript" src="74070_2212142_1596-74070-2211634.js"> </script>
```

THE SERVER PAGE DESCRIPTIONS COMPONENT (A <HEAD> COMPONENT)

The next file requested is the one that contains the actual HTML page descriptions produced in the "Server Scripts" editor.

```
<script language="javascript" src="hndmtsngsvr.js"> </script>
```

All page descriptions are packaged as JavaScript variables so that to draw any one page component takes one line of JavaScript code. We saw earlier in this lesson the code that draws the web email form. Drawing the home page is done as follows:

```
<script language="javascript">
   document.write(unescape(page.home));
</script>
```

THE PAGE TITLE COMPONENT (A <HEAD> COMPONENT)

This page component is the text that appears in the title bar of the page when it displays in your browser.

```
  <title>
   Messages Query Page
  </title>
```

THE META DATA COMPONENT (A <HEAD> COMPONENT)

Typically meta-data appears only in your home page. The author and description lines are hard coded into the HNDMTSNG.EXE server. When you code your own server, what you put here (if anything) is up to you. The keywords line can be altered even in the HNDMTSNG.EXE server, via the "Web Crawler Keywords" section of the server variables editor. By setting Web Crawler Keywords to "N/A" for not applicable, the server will omit meta-data altogether in the event you don't want to advertise your HNDMTSNG.EXE site to the major search engines. However, if that's what you want, there are other ways to prevent search engines from indexing your page using a **robots.txt** file. But that's a whole other discussion.

```
<meta name="Author" content="GUS M. CRECES">
<meta name="Description" content="CHT Web Groups Server - Author: Gus M. Creces. Author
     Website: http://www.cwhandy.com.">
<meta name="Keywords" content="© 2004 The Clarion Handy Tools Page –
     http://www.cwhandy.com, http://news.cwhandy.ca"
```

THE STYLE SHEET COMPONENT (A <HEAD> COMPONENT)

This file request is for the style sheet descriptions created in the "Server Scripts" editor. Its purpose and the use of style sheets has already been discussed at some length.

```
<LINK REL="STYLESHEET" HREF="hndmtsngsty.css" TYPE="text/css">
```

THE </HEAD> TAG

The </HEAD> closes the head section of the HTML page.

```
</head>
```

PAGE COMPONENTS FOUND BETWEEN <BODY> AND </BODY>

The visible part of any HTML page is found between the opening and closing <BODY></BODY> tags. All pages in the HNDMTSNG.EXE server use the same body style sheet named "bldr_body". This style sheet provides some basic page layout and font stylistic information which may be overridden by style sheets referred to inside the body scripts. Typically, the body style sheet provides page layout parameters such as the left and right margins, basic fonts and a background image with positional information, when there is one.

```
<body class="bldr_body">
```

THE HEAD.IMAGE COMPONENT (A <BODY> COMPONENT)

The HNDMTSNG.EXE server divides the BODY area of each web page into several layers each of which is laid down separately and is contained and described by a separate "Server Script". The server has been programmed to call an opening script called **head.image** which in this case happens to contain code that draws an image (your logo) at the top of each page using information provided in the server variables file.

In the scripts released with Standard Editions 1.10 and 1.11 of the HNDMTSNG.EXE server this script does little more than display the header image on each page. It's important to remember that **head.image** is nothing more or less than a script. And that what that script actually does is entirely up to the server owner. It's 100% under server manager control. Remember too, that a server of your own design could be

made to call a different opening script. In other words, were you designing a server of your own, the script name being called here is under your control as well as what the script does. In the case of the mts newsgroup server, the opening script name is not under your control but what that script does is entirely up to you.

```
<script language="javascript">
    document.write(unescape(head.image));
</script>
```

THE MARQUEE.COMMON COMPONENT (A <BODY> COMPONENT)

The next layer put down is a script called **marquee.common**. Its purpose in the HNDMTSNG.EXE server is to create the marquee that runs across the top of your web pages. Like the previous section, and all body components described here, this is 100% under your control via the "Server Scripts" editor built into your server. If you want to do away with the marquee altogether as we have done, or perhaps display the marquee only on certain pages, the power to do that is there for you to exercise.

```
<script language="javascript">
   document.write(unescape(marquee.common));
</script>
```

THE TITLE.COMMON COMPONENT (A <BODY> COMPONENT)

The **title.common** script is written next. Its purpose in the HNDMTSNG.EXE server is to provide the title bar area of each page, where page identification is provided so that visitors know at all times what page they are viewing. Like the previous sections, and all body components described here, this is 100% under your control via the "Server Scripts" editor built into the server. If you want to do away with this component altogether or perhaps have the title bar provide something other than page-identifying information, that's entirely within your power to change.

The order in which page "layers" or components are laid down does not have to dictate where they appear on your web page, though it will if you're not using style sheets to full advantage. The style sheet **absolute** positioning element, to which *most* browsers now pay attention, allow you to place page components regardless of the order in which they're rendered. This was not always the case in the past and continues to be somewhat of an issue with certain browsers that don't conform, to their own detriment, exactly to style sheet standards.

```
<script language="javascript">
  document.write(unescape(title.common));
</script>
```

THE LINKS.COMMON COMPONENT (A <BODY> COMPONENT)

After the title layer, the **links.common** script is written out. Its purpose in the HNDMTSNG.EXE server is to provide any extra links to other pages that are component pages of this site or another site. Once again, what you do with this entirely up to you. In the HNDMTSNG.EXE server, we're using the **links.common** script to write out the page navigation menus at the top of every page. If you examine the code contained in the script named **links.common**, you'll note that a call is made to a JavaScript called **jslinks.writemenus()** where extensive use is made of the **sys.pageid** variable to vary the menu information conditional on which page is being displayed.

Depending on which page creation template is in use, the StandardizedPageProcessing section shown here between BEGIN: and END: could contain another section - the text portion of the page - which is dropped in before the **links.common** script is written out. You can see this happening on pages like the HOME page and the LOGIN page where there are **TEXT** sections to be displayed rather than **DATA** sections like browse data which we'll discuss shortly in reference to the message browse page, the topic of this discussion.

```
<!-- BEGIN: Query Results Form - StandardizedPageProcessing() -->
<script language="javascript">
document.write(unescape(links.common));
</script>
<!---- END:  Query Results Form - StandardizedPageProcessing() -->
```

THE FORM COMPONENT (A <BODY> COMPONENT)

All pages that have the ability to return information back to the server contain a FORM component between the <FORM> and </FORM> tags. Form tags are sub-components of the page <BODY>. The HTML form structure used on the messages browse page is written out by the server based on template information collected by the **BuildQueryResultsPage_BIC** template. All browses and edit forms are written inside this basic page structure provided by the BuildQueryResultsPage_BIC template.

Forms contain an ACTION command, something they want the server to do. The action attached to the messages browse is **KQY$** or **REQUEST:TakeQuery**. The reason for this is that the messages browse produces a new query when you click on the **View/Edit** button or **Open** button to view or edit a browse record. That new query is one that isolates a single record, the one you want to see or edit, and causes the server to send that record to you.

Also embedded into this page form is a certain amount of "state" information in the form of hidden HTML variables. Embedded in this form are the **sessionid**, the back end view name, the **editaction** flag, the query that got us this page and the query page number that got us to this page. You'll remember the discussion in Lesson 1 about "State" information and HTTP servers. If not, you should go back and review that information before completing this lesson. Suffice to say that HTTP servers should not be required to maintain a lot of state information about any specific "logged-in" browser.

The rational for this is discussed in Lesson 1. State information is therefore stored in form pages as "hidden" variables. This state information is returned to the server when the ACTION on a form page takes place. The key bits of information here are the **sessionid** and the back end **viewid** being processed.

```
<!-- Added by HNDHtml.OpenTableForm() -->
<form action ="http://65.95.89.217:443/KQY$" method="POST" name="ngmessagesvieweditform">
<input type="HIDDEN" id="sessionid" name="sessionid" readOnly  value="1913-74070-3188101">
<input type="HIDDEN" id="viewid" name="viewid" readOnly  value="NGMESSAGESVIEW">
<input type="HIDDEN" id="editaction" name="editaction" readOnly  value="1">
<input type="HIDDEN" id="queryfield" name="queryfield" readOnly  value="DATE RANGE ╗
TODAY()-7,TODAY()ORDER BY -DATE,-TIME">
<input type="HIDDEN" id="querypage" name="querypage" readOnly  value="1">
<!--------- BEGIN: Messages Browse Written From Javascript file. ----------->
<script language="javascript">
   document.write(unescape(page.messagesbrowse)) ;
</script>
<!--------- END:   Messages Browse Written From Javascript file. ----------->
</form>
```

The page script **page.messagesbrowse** follows the hidden variables inside the <FORM></FORM> structure. This script uses the back end data provided in this page's data package **.js** file - the one with the long, numbered file name - to draw out the browse data lines. The number of lines, obviously, varies from browse to browse depending on the query that triggered the browse. Attached to each data line is a set of two buttons. One button, the **View/Edit** or **Open** button, has a query embedded in it that leads to the edit form being requested via the KQY$ action on this form.

The other button, the **Thread** button has a query attached to it that requests a browse in which only the records with the same thread ID as the current record should be displayed.

A note of caution is in order pertaining to "hidden" variables while we're discussing them. As you've just seen, these are really not very well hidden. So they should contain no information of a secure or secret nature. They're only "hidden" in the sense that the information contained in them is not on display in the page interface. Their chief use in dynamic web pages is to carry state information back and forth between the server and the client browser. They remind the server where that specific browser client left off - what state or condition it's in - when it makes its next page request.

THE FOOT.COMMON COMPONENT (A <BODY> COMPONENT)

At the bottom of every HNDMTSNG.EXE server page is a request to write out the **foot.common** script. This can contain the script for the page footer if you like, or it can contain nothing. Again that's up to the script writer.

```
<script language="javascript">
    document.write(unescape(foot.common));
</script>
```

If you examine the **foot.common** script, you'll see that it calls a JavaScript subroutine called **jsfoot.drawpagefooter().** This script is stopped from writing any footer information because the jsfoot.drawpagefoorter() function simply does a return and nothing else.

The **foot.common** script available in (FOOT) 01. Common Page Footer HTML.

```
<table class="bldr_copyright">
 <tr>
  <td>
    <script language="javascript"> javascript:jsfoot.drawpagefooter();  </script>
  </td>
 </tr>
</table>
```

The **jsfoot.drawpagefooter()** JavaScript available in (JSFOOT) Draw Page Footer Script.

```
jsfoot.drawpagefooter() {

  /* DO NOT DRAW FOOTER */
  return  ;
}
```

THE </BODY> TAG

The </BODY> Tag marks the end of the page body section.

```
    </body>
```

LESSON 3 - SUMMATION

In this lesson, we've reviewed in detail the answers to investigations we requested in lesson 2.
These involve:

- Understanding the mechanism by which a browser makes conditional file requests and why.
- Understanding how to create your own style sheets using the server scripts editor tool and how to apply that style sheet on one or more of your pages.
- Understanding how to add server variables, how to prioritize them in the server variables browse and how to make use of your own server variables in the page scripts.
- Understanding how to open and examine an HNDMTSNG.EXE "server data file" from the client side's web cache.

Finally, we covered in considerable detail every component of an HNDMTSNG.EXE server web page, explaining as we went how these web pages incorporate the various pieces - page scripts, subroutines and style sheets - provided by the server via the server scripts editor. The aim of this lesson component was to help you understand that the scripts in HNDMTSNG.EXE are 100% under your control and to encourage you to begin to exercise some control over them. Ultimately this should help you, in upcoming lessons, when you create a back-end server of your own design, to achieve greater control of what happens at the front end, where your scripts are executing.

LESSON 3 - EXERCISES

1) Modify the **form.membersemail** script to disable the email send button depending on the state of server variable **reg.allowsendmail**. In other words, do not allow mail to be sent via the web email form if the member has marked his member record with "disallow" in the email send field.

2) Modify the **head.image** script to handle not only the drawing of the page header image but also to handle some of the work now being handled by the **title.common** script, including drawing the title bar under the logo and the user's name at the top of the page (when known). At the same time, *do not obsolete* the title.common script so that it does nothing, since much of the tool tips text is handled also by title.common.

3) Find the scripts that contain and execute the actions hooked to the messages browse **View/Edit** button and the **Thread** button. Copy the script code to your answer and explain briefly what the code is doing.

Before you start these investigations, please back up any script work that you've done that you want to keep, using the HNDMTSNG.EXE script backup facility available inside the "Server Scripts" editor via "*File -> Back Up Server Data*". If you've already trashed the default scripts with your prior work, and didn't do a backup, it may be helpful to start clean with a re-install of the server.

Cheers...
Gus M. Creces
The Clarion Handy Tools Page