Dynamic Generation of Menu Structures and JavaScript Rollovers in PHP

By Chris Mospaw

Table of Contents

Starting Out

Did you ever have one of those sites that started out small, but kept growing and growing? Adding new pages to a site or removing old ones can often take more time than developing the pages themselves.

Even with advanced search and replace, it can quickly become a real chore to manually change all the links on every page, and a nightmare to keep up with things as the site grows to 20 or even 30 pages. If you throw JavaScript rollovers into the mix, things can quickly get out of hand.

Fortunately, PHP is perfectly suited for creating menu structures and generating JavaScript for rollovers automatically. I hate to do anything twice, especially coding, so I used these techniques when developing my company website (http://insight.ourinternetsite.com). You might want to take a look at the pages to get an idea of the effect we're after.

All I need do to add a page to the site, and more importantly, a link to that page on every other page as well as JavaScript rollovers, is to generate a handful of graphics files from templates in PhotoShop, and plug the text into the template file. PHP scripts take care of the rest, and maintenance is a snap because changes are required on only one or two scripts rather than 30 or 40 pages.

This article is a step−by−step process, so you might want to pull up your favorite HTML editor. It also assumes that you have a basic familiarity with JavaScript and HTML table layout. The servers on all of my sites are all set up to parse HTML files as PHP, so these examples are called FOO.HTML, instead of FOO.PHP. You may have to rename the files based on your server configuration.

There is a zip file available of an example of how this works, complete with the necessary graphics files and all source code. Download it here.

Starting Out

Whenever building a site of any size, it's best to start out with a test page or two to make sure the basic underlying structure is sound. Most pages seem to take the form of a large table, with a menu structure about 100 pixels wide on the left or right, and a "main body" area to take up the rest of the room.

Call me old−fashioned, but I still develop for 640 pixel wide screens, which makes a 596−pixel−wide table about the maximum, with cell spacing and cell padding set to 0. Typically, the link buttons within the menu structure are about 100 pixels wide, justified to either side of the cell. Diagram1 shows this basic layout that this code (named TEMPLATE.HTML) produces:

```
<HTML> <HEAD> <TITLE>Template</TITLE> </HEAD> <BODY> <TABLE
WIDTH=596 BORDER="1" CELLPADDING="0" CELLSPACING="0"> <TR> <TD
width=596 VALIGN="baseline" COLSPAN="2"> <P ALIGN="center"> <IMG
SRC="blank.gif" WIDTH=596 HEIGHT=1 BORDER=0 ALT=""><BR> 1 − Header
Area</P> </TD> </TR> <TR> <TD WIDTH="100" VALIGN="top"> <P ALIGN="center">
```

```
<IMG SRC="blank.gif" WIDTH=100 HEIGHT=1 BORDER=0 ALT=""><BR> 2 − Menu
Area</P> </TD> <TD WIDTH="496" VALIGN="top"> <P ALIGN="center"> <IMG
SRC="blank.gif" WIDTH=496 HEIGHT=1 BORDER=0 ALT=""><BR> 3 − Main
Body</P> </TD> </TR> </TABLE> </BODY> </HTML>
```

The file BLANK.GIF is a 1x1 pixel clear cell, perfect for fleshing out tables, and is necessary to make the page render the same in Netscape and Explorer. BORDER is set to 1 for this example, but is usually left at 0 on the final pages.

The principal areas of concentration are: 1, the Header area; and 2, the Menu area; since 3, the Main Body, does not lend itself to dynamic generation (unless you have database−generated sites, but that's another topic).

The key to making all of this work is keeping the graphic file names within a strict convention. It is easiest to base this naming convention on the page name, using a suffix for graphics files to indicate which component they are. For example, on a page named TEST.HTML the graphics file for the header graphic would be named TEST−H.GIF (or TEST−H.JPG). We'll get into more details in just a bit.

The only other limitations are that each type of element (button, header, etc.) must be the same format (JPG, GIF, or PNG) and they should all be the same pixel dimensions. In other words, if your header is a 400 x 60 pixel JPG file, all of your headers must also be 400 x 60 pixel JPG files; while if your buttons are 92 x 15 pixel GIF files, they all must be 92 x 15 pixel GIF files. They can be any pixel depth, any byte size, etc. This can all be changed at any point. The key is consistency.

(NOTE: If you don't care about generating proper HTML with HEIGHT and WIDTH tags in the <IMG> tag, you don't have to keep all items of a type the same pixel dimensions, but your pages risk looking really weird, and they will certainly take longer to render on the visitors' screens. I've tried it, and it just doesn't work well.)

Making a Template

Now that we have the basic building blocks of the page constructed, it's time to put in some of the parts to automate. At first, however, things should be done manually to make sure they work. We should start out by inserting some JavaScript into the <HEAD> tag, preferably after the <TITLE> and <META> tags. Here is the basic code that we need:

```
<script language="JavaScript"> <!−− This script controls the rollovers in the menu area if (document.images) { image1off = new Image(); image1off.src = "foo−off.gif"; image1on = new Image(); image1on.src = "foo−on.gif"; otherImage1 = new Image(); otherImage1.src = "foo−b.gif"; otherImageDefault = new Image(); otherImageDefault.src = "default.gif"; } function changeImages() { if (document.images) { for (var i=0; i<changeImages.arguments.length; i+=2) { document[changeImages.arguments[i]].src = eval(changeImages.arguments[i+1] + ".src"); } } } // −−> </script>
```

This introduces the naming convention for the buttons. Images in their default state, before the rollover, are FOO−OFF.GIF, and during the onMouseOver are FOO−ON.GIF. The file FOO−B.GIF is for the graphic panel, which changes as the onMouseOver events occur. DEFAULT.GIF, logically enough, is the default for this panel graphic, and is what displays when no rollover occurs.

The only other element to making this script work is in the anchor tag referring to the other pages in the site:

```
<a href="foo.html" onMouseOver="changeImages ('image1', 'image1on', 'otherImage', 'otherImage1')" onMouseOut="changeImages ('image1', 'image1off', 'otherImage', 'otherImageDefault')"> <img name="image1" src="foo−off.gif" border=0></a><BR>
```

For the sake of simplicity, I have listed only the essential parts of the <IMG> tag.

The lines of code above define the images that load through the "changeImages" JavaScript function when the onMouseOver and onMouseOut events happen. The name="image1" in the <IMG> tag ties it into the script defined above.

As you'll soon see, you don't even really need to know what this does, since PHP will automate it. All you'll need to keep track of the different graphics files, and that's easy. This page will probably generate JavaScript errors if displayed in a browser because it is referencing named items which are not yet defined.

Starting to Automate

The first step in automating the generation of these pages is to determine which page is being displayed. PHP comes with great environmental variable support, so determining this information is fairly straightforward.

I've found that certain environmental variables are not uniformly supported on all implementations of PHP and all servers. Since my local development machine is a Windows 95 box and the Internet server a Unix box, the code that works on both is a bit convoluted. But it works on all installations I've tested it on.

This code, which should be put at the top of the script, simply puts the name of the current page (with no file type suffix) into the $page variable:

```
<? $page=getenv(SCRIPT_NAME) ; $page = split( "/", $page, 4); $page = "." . $page[3];
$page = split( "\.", $page, 3); $page = $page[1]; ?>
```

You will probably have to adjust the "4" in line 3, and the "3" in line 4 since they refer to the depth of the directories on your machine, and that's likely to be different than mine. Inserting an echo "$page n"; between each line in the above snippet helps to track where things are going while adjusting lines 3 and 4 to work properly in your configuration.

Now that we have the current page name without any file extensions, we're ready to start having PHP automate some items, such as the header graphic. Replace the "1 − Header Area" text with this line:

```
<IMG SRC="<? echo "$page−h.gif"; ?>" WIDTH=596 HEIGHT=50 BORDER=0
ALT=""><BR>
```

Notice the "inline" PHP in the SRC attribute, which just echoes the page name followed by the "−h" suffix. The height and width attributes are "hard wired" in, but can be changed at any time if the dimensions of the graphics change.

Since this site will have more than one page, the script will have to know what the other pages in the site are named. For ease of maintenance, do this in a separate script called PAGES.HTML that is INCLUDEd in the main script. All PAGES.HTML does it to define an array containing the names of all the pages to be dynamically generated.

```
<? $site_pages = array("index", "info", "clients", "contact" ); ?>
```

There would be 4 pages in the dynamically generated menus on this particular site. Don't forget to insert:

```
<? include "pages.html"; ?>
```

near the top of the TEMPLATE.HTML script.

The power of repetition

Computers sure do tolerate repetitive counting tasks a lot better than I do, and PHP takes maximum advantage of that by allowing stepping through each part of the $SITE_PAGES array, generating the necessary pointers for the JavaScript, as well as the appropriate buttons and links to other pages.

Once the basic page layout and JavaScript are set up, setting up the automation is pretty much a matter of echoing either a numeric value (in the case of JavaScript pointers) or part of an array (in the case of links to other pages in the menu).

Replacing a few lines of the JavaScript shown earlier with some PHP yields this code:

```
<script language="JavaScript"> <!−− This script controls the rollovers in the menu area if (document.images) { <? // Changed code begins here! $counter = sizeof($site_pages); $i = 0; do { $item = $i + 1; echo " image" . $item . "off = new Image();\n"; echo " image" . $item . "off.src = \"$site_pages[$i]−off.gif\";\n"; echo " image" . $item . "on = new Image();\n"; echo " image" . $item . "on.src = \"$site_pages[$i]−on.gif\";\n"; echo " otherImage" . $item . " = new Image();\n"; echo " otherImage" . $item . ".src = \"$site_pages[$i]−b.gif\";\n"; echo "\n"; $i++; } while ($i < $counter); // Changed code ends here! ?> otherImageDefault = new Image(); otherImageDefault.src = "default.gif"; } function changeImages() { if (document.images) { for (var i=0; i<changeImages.arguments.length; i+=2) { document[changeImages.arguments[i]].src = eval(changeImages.arguments[i+1] + ".src"); } } } // −−> </script>
```

Line 7 gets the number of items in the array $SITE_PAGES (which is defined in PAGES.HTML) and assigns it to a counter used in the DO loop defined in lines 10 through 23. Line 11 assigns the $ITEM variable in order to allow us to start the JavaScript definitions at 1 instead of 0. Line 21 increments the place holder variable $I.

During the development of this project, some limitations of the PHP parser were encountered, but fixed with the "." (string concatenation) command used in the echo lines. Without this workaround, the code would have read:

```
echo "image$itemoff = new Image();\n";
```

which PHP (understandably) chokes on. Whenever you use variables that are embedded in the middle of other text, it's best to use the "." command to tie things together rather than relying on the PHP parser to figure out what you want.

The rest of the JavaScript remains the same because the other items only need to be defined once, and there's

no practical reason to embed it within PHP.

Ordering up a Menu

Now that the JavaScript is properly generated, the page needs a complimentary menu structure for the links to other pages. Let's dive right into the code, replacing the "2 − Menu Area" text with:

```
<? $i = 0; do { $item = $i + 1; if ($site_pages[$i] == $page) { echo "<img name=\"image" .
$item . "\" src=\"$site_pages[$i]−in.gif\" alt=\"" . $site_pages[$i] . ".html\" width=100
height=15 border=0></a><BR>\n"; }else { echo " <a href=\"$site_pages[$i].html\"
onMouseOver=\"changeImages('image" . $item . "', 'image" . $item . "on', 'otherImage',
'otherImage" . $item . "')\" "; echo " onMouseOut=\"changeImages('image" . $item . "',
'image" . $item . "off', 'otherImage', 'otherImageDefault')\">"; echo " <img name=\"image" .
$item . "\" src=\"$site_pages[$i]−off.gif\" alt=\"" . $site_pages[$i] . ".html\" width=100
height=15 border=0></a><BR>\n"; } $i++; }while ($i < $counter); ?>
```

It sure does look complicated, but it's pretty much the same HTML anchor tag we put in the TEMPLATE.HTML file above, but parsed to see if the array element is the current page, and with a whole bunch of escape characters and PHP variables.

This code uses the $COUNTER variable defined earlier in the script in another DO loop, and generates the appropriate HTML code with JavaScript support. Line 7 introduces another graphic file, FOO−IN.HTML, which is the "third" button file, the other two being FOO−ON.HTML and FOO−OFF.HTML referenced already in the JavaScript.

The only part left to do is add the HTML line referencing the graphic panel that changes when hovering over the buttons:

```
<img name="otherImage" src="default.gif" alt=" " width=100 height=100 border=0><BR>
```

This file can actually be put anywhere in the page, but for the sake of simplicity I put it in the menu structure for this example. Since it is referenced by the "NAME" attribute, this is all that is needed to use this graphic in the onMouseOver effects. JavaScript takes care of the rest.

Adding Pages

Now that the TEMPLATE.HTML file is completed (with the possible exception of changing the table border to 0), it can be used as the basis for new pages on the web site. Just do a SAVE AS to the filename of the new page, and put the page content in the "3 − Main Body" area.

Don't forget to go into PAGES.HTML and add the name of the new page to the array definition. By the way, the pages named in the array display in the menu in the exact order in which they occur in the array. By altering this order, you can alter the order in which pages appear in the menus.

You will also need the following 5 graphics files for each page:

- FOO−H.gif − header graphic
- FOO−B.gif − graphic panel that changes when hovering over buttons
- FOO−OFF.gif − default button displayed for non−current page
- FOO−ON.gif − button displayed when onMouseOver
- FOO−IN.gif − button graphic displayed when page is current

(GIF files were used in this example. JPG's and PNG's work just as well. Just remember that every file of each type must be the same format.)

Adding one page or one hundred pages is now just a matter of using the template file and then generating the graphics files.

Using PHP for the dynamic generation of menu structures and JavaScript can dramatically cut the time it takes to develop a site, and certainly makes site maintenance a breeze compared to having to edit each page when other pages are added or removed. Adding or deleting pages from the site's menu structure is now just a matter of making minor changes to a few files, and producing some graphics.

Another good use for PHP is separating the content of a page from the structure of a page, thus making site maintenance even easier and nearly eliminating formatting mistakes, but that's a subject for another article.