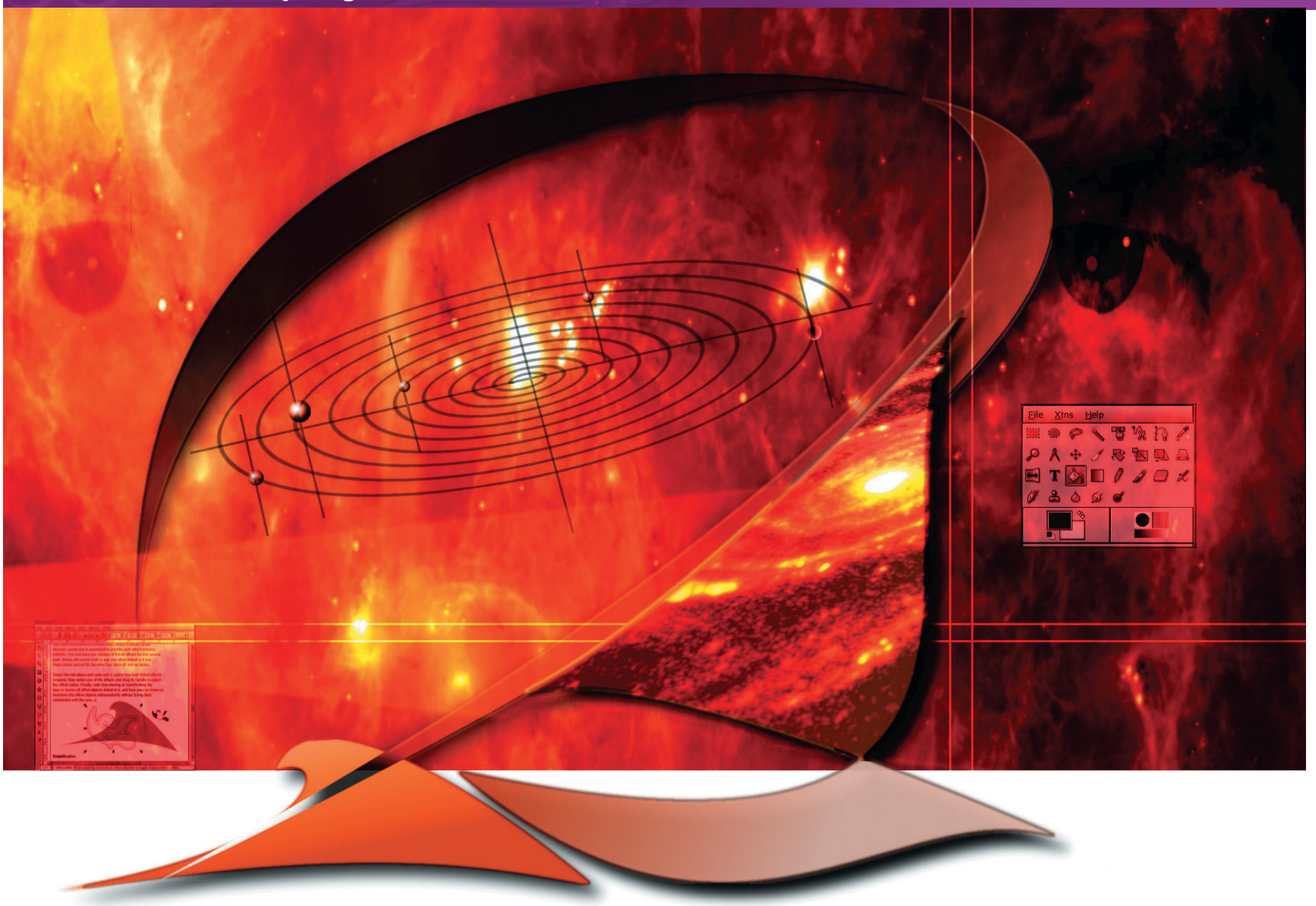


DESIGN AN IMAGE GALLERY

Gallery design using Gimp, PHP and CSS

PART 2 **In Part I, Michael J Hammel took us part way through a *Gimp* online gallery design. But online systems make use of a variety of tools...**

In last month's tutorial we started in on a project to build an online, dynamic gallery. We began by laying out a set of seven design requirements for the project, which were as follows:

1. We must be able to handle any number of images in a single directory.
2. The images must be sorted alphabetically by filename when displayed.
3. We must handle landscape (wide), portrait (tall) or square images, and no thumbnail should be taller or wider than 100 pixels.
4. Each thumbnail must reference a page with a full size original of the image.
5. We must provide a 3 x 3 Layout.

TUNING FOR BROWSERS

The gallery created here will work fine under the Firefox browser, but may not work as expected in other browsers. Fine tuning the CSS for cross-browser support is a big topic. There are many good texts on the subject such as those by Eric Meyer, whose *More on Eric Meyer on CSS* offered the basic premise for this project.

The complete code and a working version of this two-part tutorial is available online at the author's web site www.ximba.org/articles/lf/63

6 The interface must provide a 'film slide' presentation, with image names listed on each slide frame.

7 For interactivity, we want hover images – when the mouse pointer is over the image – in colour, while default images are displayed in greyscale.

We built a *Gimp* Perl script that processed a directory of images and produced a series of greyscale and colour thumbnail images. This script enabled us to meet requirements 1 and 7 of our design. Now it's time to add online interaction to our project.

This month we'll address the other five requirements by developing a PHP script, an HTML template and a CSS style sheet. Along the way we'll make one more trip into *Gimp* to add a little style to our layout.

The Design: HTML Layout

The first thing we need to do is create a page layout in HTML. Our gallery will be a 3 x 3 sheet of images. Because we'll be using Cascading Style Sheets (CSS) to define the page layout, our HTML is rather terse. It consists of a header section, a body with one external and three internal blocks, and some placeholders for our gallery images. Each block is a row surrounded by **<DIV>** tags. Each row will have three cells. Figure 1 shows the basic layout.

The HTML we've created is a template. When visitors come to our site they'll cause our PHP script to be run. The script will read in the HTML template and replace the place holders (which we call template tags) with the HTML needed to insert one of our nine images.

The HTML on its own doesn't do much. Figure 2 shows how it would look with the template tags replaced with coloured blocks when no CSS is used to control the layout (though a little CSS is used to provide colour to the cells).

The HTML for this is extremely simple and will require few changes as the project evolves. Code listing 1 shows the code inside our sheet of rows and columns. What is important to note here is that the outside block is called 'sheet' and the inside blocks are called 'row' and 'cell'. The id and class notations will be used in our CSS later to describe how the sheet, rows and cells should be displayed. We'll also replace all the cell code in the HTML template with a template tag that looks like this

[--divN--]

There will be nine of these, and the 'N' will be replaced by the appropriate number. Our PHP script will replace these with the code for a cell that includes the images and any in line styles we need.

Listing 1

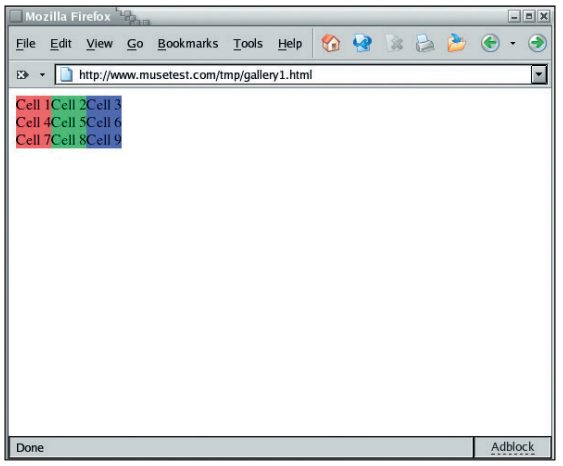
```
<div id=sheet>
  <div class=row>
  <div class=cell> Cell 1 </div>
  <div class=cell> Cell 2 </div>
  <div class=cell> Cell 3 </div>
  </div>
  <div class=row> ... </div>
  <div class=row> ... </div>
</div>
```

The Design: CSS Layout

The HTML template is a black box for our gallery – it holds our content but doesn't show us how it should look. The appearance of the gallery comes from Cascading Style Sheets, also known as CSS, which works by providing style to various elements of a page.

The style includes position and padding on the page, colours, borders, and even simple visual effects.

1/ The layout is a simple block format – a 3x3 sheet with titles printed with each slide.



2/ Our first attempt displays coloured cells.

DIG AROUND

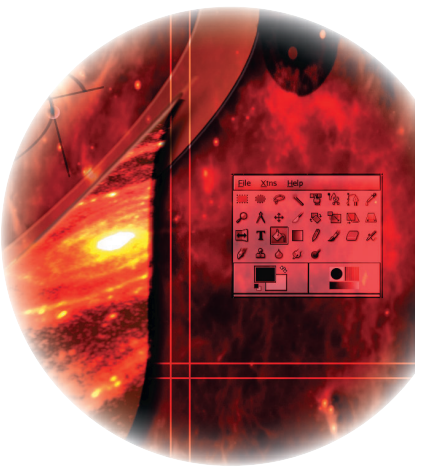
For a good taste of what CSS can do, check out the CSS Zen Garden, www.csszengarden.com/. This is a collection of designs that all use exactly the same content but display it differently using nothing but modified CSS style sheets.

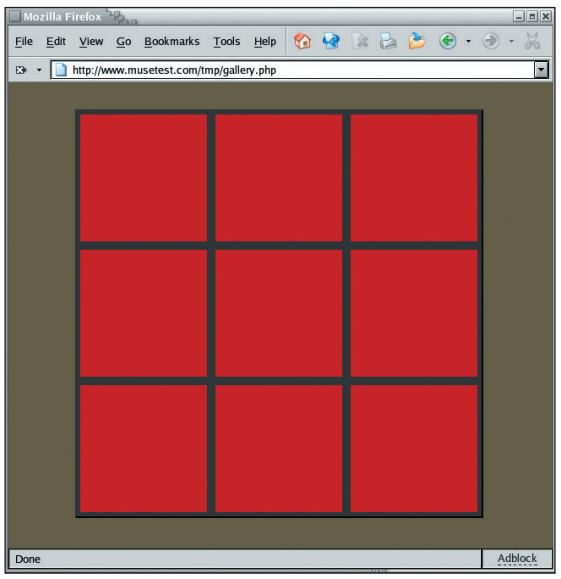
A web page uses CSS in one of three ways. It can references a Style Sheet as an external link within the **<HEAD>** section of a page; or by placing Style Sheet information directly inside the **<HEAD>** section; or by specifying styles directly aniline with HTML elements using the **STYLE** tag. Listing 2 shows how each of these might be used.

Listing 2

```
<head>
<link href="my.css" type="text/css" rel="stylesheet"
 title="Preferred"/>
<STYLE>
 body {
   background: red;
   margin: 2px;
 }
</STYLE>
</head>
<html><body>
...
<div STYLE="margin: 2px;">...</div>
...
</body></html>
```

We're going to use all three of these methods. First we'll create an external style sheet. This will contain style settings that are universal and never change no matter what images are displayed in the gallery. Then we'll use our PHP script to generate **<STYLE>** elements and in line styles for each cell on the sheet.





We centre the sheet and set the row and cell dimensions. The cells are coloured red to show how they are aligned in the rows.

The external style sheet is just another file. Listing 2 showed how we would reference it. In this file we'll specify that the page should go all the way to the edge of the browser, specify a default font for the page and provide a default background colour.

Listing 3

```
body, html {
  margin: 0; padding: 0;
  font: 10pt/12pt verdana, times;
  color: black;
  background: #7D775C;
}
```

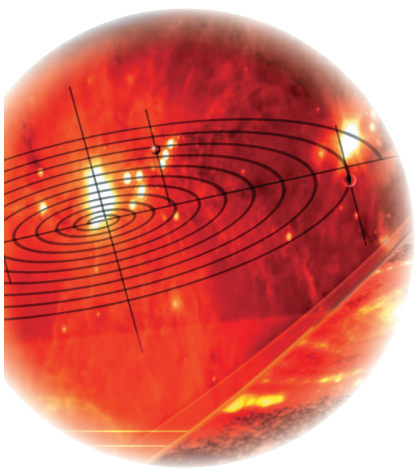
Next we need to centre the big sheet within the browser window (Listing 4). We do this by first defining how big the sheet is: 480 pixels wide (the rows will also cause it to be exactly 480 pixels tall, but we'll get to that in a moment). To centre this we first tell CSS to put the top left corner of the sheet in the middle of the page by telling it the left and top offsets are 50%. Then we move back from there half the width and height to cause the sheet to be centred. Finally we add borders to make the sheet appear to be raised from the background.

Listing 4

```
#sheet {
width: 480px;
position: absolute;
left: 50%;
top: 50%;
margin-left: -243px;
margin-top: -243px;
border-bottom: 2px solid black;
border-right: 2px solid black;
border-top: 1px solid #444;
border-left: 1px solid #444;
background: #4A4A4A;
}
```

The rows and cells come next (Listing 5). Each row is exactly three times the width of a cell (150 px plus 10 px of margin, total) and the same height as all cells (which are square). Cells are forced to line up left to right by using the "float:left" tag. Additionally, in order to force a rows to start on the next line we add a "clear" tag. Without this the rows would line up side by side. The last thing to note is that the cells each use the (yet to be created) slide image as a background.

Note that the slide is given a relative path (actually, no path is specified) which means it must be in the same directory as the CSS style sheet that references it.



Listing 5

```
.cell {
width: 150px;
height: 150px;
margin: 5px 5px 5px 5px;
float: left;
background: transparent url(slide.png);
}
.row {
width: 480px;
height: 160px;
clear: both;
}
```

All of the images we'll use will get a border that makes the image appear to be set back within the slide image. CSS lets us add this border easily – it's a cute trick. Below the images will be an image title, which we will centre, and for which we provide a small font.

Listing 6

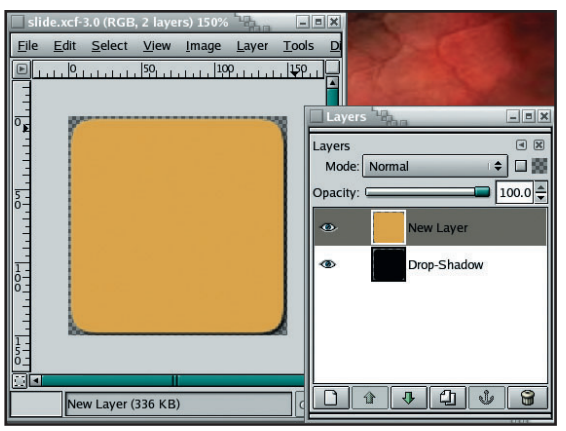
```
img {
border: 2px solid;
border-color: #444 #AAA #AAA #444;
}
#title {
font: bold italic 8pt/9pt Arial, Verdana, Times;
text-align: center;
margin: 0px;
padding: 0px;
}
```

In all this CSS you notice that the HTML elements IDs and Classes are named and then enclose their style inside curly braces. Those elements that are IDs have a **#** as a prefix (such as the **#title** element) and those that are classes are prefixed with a period. The difference between an ID and a class is that any page can use an ID just once, but classes can be used multiple times. So there is just one sheet in our page, but multiple rows and cells.

More images: The Slide Frame

Now we're ready for our slide image. This one is very simple. We create a 150x150 square canvas with a transparent background layer. Then we create a new transparent layer that is 140x140 and position it five pixels from the top and left of the canvas.

Make a rectangular selection of this layer. If you're using *Gimp* 1.2 you can use the *Gimp* Perl 'Select->Round...' option to round the corners of your selection. Feather the selection two pixels just to soften the image a little. Set the background colour to **#efefda** and fill the selection.



Our frame has a slight drop shadow.

Finally, add a drop shadow that is offset by three pixels and has a blur radius of three pixels. Save the image as a PNG file (and save it as an XCF if you want to edit the layers again later!) in the same directory as your CSS file.

One other image is necessary for this project: a transparent GIF. Keep it small – perhaps 10 x 10. Save it in the same directory as the CSS file. Call the file "clear.gif". We'll use it in the next section when we create our PHP file.

Putting it all together with PHP

The HTML template, CSS style sheet and slide image are all ready to go. Now we need to dig into the PHP. Fortunately, this isn't all that difficult either.

Our PHP starts by reading in the HTML template and saving it as one long string. This makes it easier to replace our template tags later. We also start building our **<STYLE>** section that will add the HTML template as well. Other setup includes specifying the directory for our gallery images, and creating an empty array for our images.

Listing 7

```
$html = implode('', file('gallery.template'));
$css = "<STYLE TYPE=\"text/css\">\n";
$gallerydir = "./gallery";
$images = array();
```

Next we read in our images. We want to skip the "dot" directories as well as the thumbnails – we only want the names of the full size images, since we'll use that to build the cells for each image. When we're done reading all the filenames we sort them.

Listing 8

```
if ( ($DIR = @opendir($gallerydir)) == FALSE ) return;
while (($file = readdir($DIR)) !== false)
{
if ( preg_match("/^\./", $file) ) continue;
if ( preg_match("/-tn\./", $file) ) continue;
if ( preg_match("/-gs\./", $file) ) continue;
array_push($images, $file);
}
closedir($DIR);
sort($images);
```

With the image names in hand, we need to process them one at a time to build a **<DIV>** that defines the cell in our gallery. Remember the **divN** template tag in our HTML? Here is where we replace it with the **<DIV>** that defines a specific image.

To process the image we first pull just the image name from the file name (removing the .jpg at the end of the name). We then calculate the size of a thumbnail for this image (remember they can be either portrait or landscape according to our design specification) and compute the margins to centre this image in a cell.

Next a cell entry is built as one long string. We play a trick here by using a transparent GIF image as the image in our HTML and use CSS to specify background images for normal and hover states (more on that in a moment). The image links to the larger version and an online **STYLE** is set to specify the margins (which we just computed) to centre the images. The trick here is making the transparent GIF the same size as the thumbnails – this makes a window that the background images (the thumbnails) will show through!

After building the cell entry we update the HTML string by replacing the appropriate **divN** template tag. We add two CSS styles to our **STYLE** string: one specifying the greyscale thumbnail for the background when the mouse pointer is not over the image, and one specifying the coloured thumbnail for when the mouse is positioned (re: hovering) over the image.

The greyscale thumbnail shows by default, until you place the mouse pointer over an image.

The code in Listing 9 shows how we build the cell entry (all the code for this project is available online).

Listing 9

```
$entry =
  "<a href=\"gallery/$file.jpg\">" .
  "<img id=\"pic$id\" " .
  "STYLE=\"margin-left:$margin_left; " .
  "margin-top:$margin_top;\" ".
  "width=\"$width\" height=\"$height\"".
  "src=\"clear.gif\" />".
  "</a>";
```

We add some code to clean out any unused template tags and add the **<STYLE>** entries to your HTML by replacing its template tag as well. Last of all, we print out the HTML string. Voila! Your gallery is ready for viewing!

Where to go from here

This article hasn't spelt out all the code you need, but it's all there in the online code, which also contains some added features such as a "More" link in case you have more than nine images – this was a requirement from our design specification, but we're leaving it to you to read the code and understand how it was accomplished.

In the end, we've learned that a good design specification up front helps keep us pointed in the right direction, and that elegant web layouts are primarily based on good use of Cascading Style Sheets. All of this is pulled together by the great flexibility of PHP, a programming language built specifically for the web.

Try the code and play with the CSS. Try adding background images to the main sheet and to the body of the page. Try adding some new **<DIV>** entries that place images around the page but always behind the sheet, such as a logo in the upper left corner. None of this is particularly hard to do once you learn a little CSS. So dig in! Open Source makes it easy! LXF