CHAPTER 6

Preparing for the Browser Wars

Now that we have set up the scripts to identify the user's browser, we need to set up scripts to manage the specific requirements of each browser. You can deal with multiple browsers in many different ways. The first solution, which almost all developers come up with on their own, is to create different documents for different browsers. This solution is obviously far from the best, since it requires a great deal of hard work for those who maintain the site. By creating a copy of the site for Netscape Navigator users, one for Microsoft Internet Explorer users, and copies for all the users of other browsers out there, you end up with many copies of the document to maintain. This doubles or triples your maintenance time and expenses.

Another alternative is to take the "lowest common denominator" approach. This limits you only to those JavaScript statements, HTML elements and attributes, and style sheet properties that are supported by both mainstream browsers and the other primary browsers, such as Opera. This can dramatically limit your ability to be creative. And this is the approach taken in many of the documents currently on the Internet. It creates less work for the developers and maintenance people, but the developers do not have the opportunity to stretch their imagination or their skill. If you take this approach, you do not have to maintain multiple copies of a document. You do not have to worry about elaborate scripts. You do not have worry about complicated design schemes. You simply have to worry about making sure you and any clients you may have are happy with the completed site.

It seems clear that neither of these approaches is optimal. A third one has to be found. In fact, I use an approach that allows you to branch to alternative pages when you have to, to create pages on the fly when necessary, and to create a simple API to handle most of the events and processes encountered in the development of an interactive site. The key to using all of these mechanisms is to define a variable at the top of each document that identifies the browser and version being used. Once this variable is defined, you can create external page branches, create parts of documents on the fly, and identify variables that can be used in `eval()` statements to create the code needed to make your site interactive. Let's take a look at the hardest part of this process, creating the API.

Using the DOM to Create an API

As a developer, you have many ways to create an Application Programming Interface (API) for use with all browsers. These can be simple or complicated. For most documents, you need only a simple API. To make it accessible to all the documents in a site, you need to place the JavaScript files containing this information in a location that is accessible to all visitors.

NOTE *One way to do this is to place all script files in a Scripts directory; you can also simply leave them in the same directory that contains your documents. If I have more than a couple of scripts, I generally create a directory to hold them separately from my documents and images.*

The following script segment creates variables that will be used as the foundation of your own API.

```
If (NS4 || NS6) {
   docs =  "document";
   styles = " ";
   html = ".document";
   xposition = "e.pageX";
   yposition = "e.pageY";
} else if (IE4 || IE6) {
   docs =  "document.all";
   styles = ".style";
   html = " ";
   xposition = "event.x";
   yposition = "event.y";
}
```

These variables will be implemented throughout the site's scripts to create a cross-platform API for use with the scripted elements of the document. This script needs to be loaded in each document of your site within `<script></script>` element tags in the `<head>` of the HTML document.

```
<head>
<script type = "text/JavaScript">
</script>
</head>
```

Once all the scripts have been compiled into a single document, or into a series of external scripts files referenced from the primary HTML page, you can use them to create an interactive site with cached scripts. If you save all scripts as external files, and then load them using separate `<script>` commands, they will be cached by the user's computer, reducing the total loading time of the site. You can reference an external script file using the following configuration of the `<script>` element.

```
<script lang = "JavaScript" src = "scriptfile.js">
</script>
```

Using linked and *cached pages,* the site will be able to load quickly after the first page. Cached pages are those that have been stored in the memory of the computer viewing the site, and can therefore be loaded directly from the visitor's hard drive without having to be downloaded one more time. The first page will load slowly, since it is the document that will preload the images and scripts used in the site. One way to start the preloading of images is to have a "catch page" that will provide an introduction to the site, provide visitors with information about what they can expect, and tell them what software works best with the site. This page serves many functions. Its primary job is to preload images and scripts in order to decrease the amount of time it takes to load the pages on the rest of the site. It also serves as an informational page for users of pre-4.0 Web browsers; and it provides a means to quickly catch the attention of your visitors and encourage them to visit the rest of your site. You can use this "splash page" to automatically load scripts and images to be used on the first page of the site. (Chapter 8 provides more information about preloading images.)

NOTE *You may turn visitors away from your site if the splash page takes too long to load. People want information off the Internet immediately. They don't want to have to wait for it.*

Setting Up Internal Page Branches

Internal page branching is the term for creating document content on the fly based on the browser viewing the document. This type of page development merges the document and the scripts to such an extent that it would be an understatement to say that maintenance of the site is difficult. The following script is an example of the way document content can be incorporated differently based on the browser being used. In this script, the `<layer>` element is available only for Netscape browsers, while the `<div>` element is used to create layers inside Microsoft browsers.

```
<SCRIPT type ="text/JavaScript">
<–
var docOutput =" ";
if (IE4 || IE5) {
   docOutput += "<DIV id='bodytext' style='top:75; width:300; left: 110; ";
   docOutput += "border:none; background-color: #999999; font_face: Arial;";
   docOutput += "font_color:blue;'>";
   docOutput += "<P align='center'><H1>Myth's For Sale .. Maybe</H1>";
  docOutput += "… Add more text here…";
} else if  (NS4 || NS6){
   docOutput += "<LAYER id='bodytext' top=75 width=300 left=110";
   docOutput += " src='store.htm'>";
}
document.write (docOutput);
//–>
</SCRIPT>
```

In this script, the `docOutput` variable is used to hold the content that will be written to the page. The `document.write` command writes the contents of the `docOutput` variable to the screen so it will be processed and properly displayed by the browser. Once the content is created through the script, any content that follows the script will be loaded as the document is read into the browser.

Creating Style Sheets

After the scripts have been created to set up your API, you must create a style sheet to set up the look and feel of the entire Web site. This style sheet will be linked to every page of the site, which allows it to be cached and therefore loaded quickly with each page or each page change. The following `<link>` statement needs to be placed at the top of every document used in your site. This statement links the document to the style sheet so that your visitors must load the style sheet content only once.

```
<HEAD>
   <LINK rel="stylesheet" src="acropolis.css">
</HEAD>
```

Once you have included the style sheet in the main HTML document, you have to make sure it uses only properties that are available with both the Netscape Navigator and the Internet Explorer browsers. The properties and their available values that work for both browsers are:

`background`: <background-image url>

`clear`: none

`color`: <color>

`font`: <font-family> | <font-style>

`font-family`: <family-name> | <generic-family> | serif | monospace

`font-size`: <percentage>

`font-weight`: bold

`line-height`: normal

`text-align`: left | right | center

`text-decoration`: line-through

`text-indent`: <length> | <percentage>

The CSS document shown in Listing 6-1 has been created for the Acropolis Web site. It covers the majority of the style elements used in the document.

Listing 6-1. A simple, complete style sheet for the Acropolis Web Site

```
//Cascading style sheet for the Acropolis Web site//
H1        { color: "red";
            font: "arial italic";
            font-size: "largest";
            text-align: "center";}
H2        { color: "red";
            font: "italic";
            font-family: "monospace";
            font-size: "largest";
            text-align: "left";
            text-indent: "30px";}
p:zeustxt { background: "red";
            border: "black";
            color: "black";
            font: "Verdana, normal";
            font-size: "larger";
            text-align: "left";}
```

```
p:hermestxt { background: "yellow";
              border: "blue";
              color: "black";
              font: "Times new Roman, normal";
              font-size: "smaller";
              text-align: "left";}
p:bldgtxt { background: "green";
            border: "blue";
            color: "black";
            font: "Arial, normal";
            font-size: "normal";
            text-align: "left";}
p:menutxt  { background: "black";
             border: "red";
             color: "white";
             font: "Times new Roman, normal";
             font-size: "larger";
             text-align: "left";}
```

Just as the preceding style sheet uses only properties that are acceptable for all browsers, the following list is essentially the "danger" list for CSS developers. If you wish to use the properties on this list with their associated value options, you will need to use them within internal branching statements for the corresponding browsers. Also, be warned that they are, at best, buggy in both Netscape Navigator and Internet Explorer, working to some degree differently in each browser.

`#ID` (Id selector)

`border`: <border-width> | <border-style>

`border-bottom`: <border-bottom-width> | <border-style> | <color>

`border-left`: <border-left-width> | <border-style> | <color>

`border-right`: <border-right-width> | <border-style> | <color>

`border-top`: <border-top-width> | <border-style> | <color>

`display`: inline | list-item

`float`: none | left | right

`margin`: <length>

`margin-left`: auto

`margin-right`: auto

`padding`: <length> | <percentage>

`padding-bottom`: <length> | <percentage>

`padding-left`: <length> | <percentage>

`padding-right`: <length> | <percentage>

`padding-top`: <length> | <percentage>

`text-decoration:` overline | underline | blink | line-through | none

`vertical-align`: bottom | text-bottom

`white-space:` nowrap

Creating Graphics

Images can make or break a Web site. Elaborate images are great to provide the visual stimuli you want on your sites, but often their large size can make sites unwieldy for visitors. There are a few things you must do to put usable images on your Web site that will not annoy visitors:

- **Crop images.** By reducing the dimensions of an image, you reduce its size and therefore its load time.

- **Use GIFs for graphic elements.** GIF graphics give you the smallest graphical elements using 256 colors or less.

- **Use JPEGs for photographs.** JPEG files use 16 million colors and create the smallest files that use the many colors needed for photographic quality.

- **Reduce number of colors.** For clarity and good image resolution, reduce the number of colors in each image to the smallest number possible.

- **Display low-resolution images while loading larger images.** By preloading images and then displaying low-resolution images while you load larger ones, you will make your clients happier, since their pages will appear to load faster. In the end, however, they will still have their high-resolution images. The total load time of the page will increase overall, although the visitor will see an image in less time.

- **Interlace GIF images.** If you interlace GIF images, they load faster and give the visitor an idea of what the image will look like before it has completely loaded. Interlaced GIF images will show approximately ten copies of the first line of the image, so that the image is filled in with chunks rather than from top to bottom.

- **Specify height and width attributes for `<img>` elements.** The height and width attributes of the `<img>` element allow the remaining text to load, correctly formatted, while the image is downloaded to the browser. As an aside, this is required in the HTML 4 specification for valid documents, although Web browsers will still display the document.

- **Trim down animations.** Crop all animations so that they take up as little space as possible and don't include any static areas of an image in the animation.

- **Use thumbnail images when possible.** Use small, clear thumbnail images rather than large full-scale images so that visitors have the option (but are not obligated) to look at larger images.

- **Break images into reusable parts.** Whenever possible, break an image into multiple parts that can be reused. By reusing parts of an image, you save visitors the time it takes to download the full image again, especially if only a few changes have been made in it. Just be judicious in your choices. There is no reason to break a single black line into multiple pieces.

- **Provide text alternatives.** Many users don't like to download images and would rather just view the text in the document. You can provide text descriptions of your images in the `alt` attributes of the `IMG` element. This attribute is required for HTML 4 validity, although Web browsers currently don't care if you have this attribute specified. You can also provide a completely text-based version of your document with no images at all. Text versions of your document would have to be loaded through a script at the selection of the site visitor.

NOTE *Vector images can be used with Internet Explorer 5 and Netscape Navigator 6. These images use the XML-based Scalable Vector Graphics (SGV) to create images with text-based instructions. These images can take up significant document space and can be somewhat difficult to align properly. At present, the best way to create SVG images is to use a graphics program that supports the SVG format to draw the images, then copy the SVG instructions into your current document. Since SVG is not supported in Netscape 4.5 or earlier, or in Internet Explorer 4 or earlier, you will either have to provide an inline branching system to draw the SVG or load a standard image to provide the same interface for all users. You can read more about SVG at the World Wide Web Consortium site located at* `http://www.w3.org/Graphics/SVG/Overview.html`.

Identifying Layers

The last concern when you prepare the main foundation for your Web site is the way browsers deal with layered content. If you are unfamiliar with layers, try thinking of them as a stack of glass plates with images or text on each, so that their contents can be viewed all at once. The size and content of each individual layer can differ. Some can contain images, others text, and still others a mix of both. Before the development of this three-dimensional Web site creation model, all Web sites were flat, and each had only one “sheet of glass” to write or draw on. Now that a stack of layers is used, each layer contains specific information that can be changed while the site is being viewed. Figure 6-1 shows how these layers are stacked together.

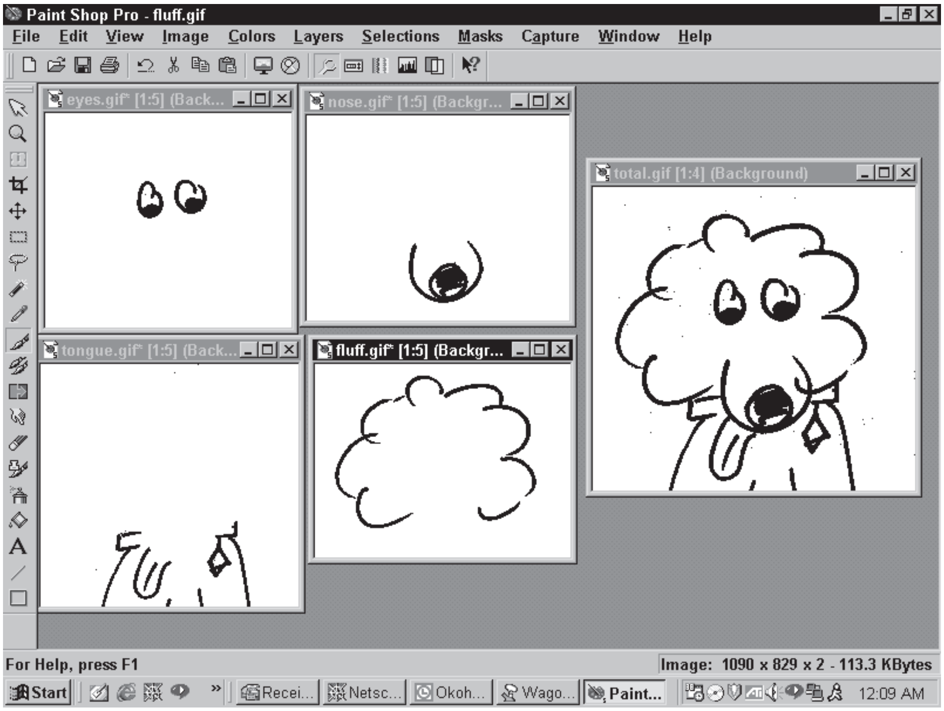
Paint Shop Pro - fluff.gif
File Edit View Image Colors Layers Selections Masks Capture Window Help
eyes.gif* [1:5] (Back...
nose.gif* [1:5] (Backgr...
total.gif [1:4] (Background)
tongue.gif* [1:5] (Back...
fluff.gif* [1:5] (Backgr...
For Help, press F1
Image: 1090 x 829 x 2 - 113.3 KBytes
Start
Recei...
Netsc...
Okoh...
Wago...
Paint...
12:09 AM

Figure 6-1. Each layer in this image contains one piece of the dog illustration shown. This technique allows you to "put the dog together," with its pieces flying toward each other from opposite sides of the visitor's monitor. Or, you can use the layers to create a game in which the dog becomes a puzzle for visitors to put together on their own.

Quite a few layers will have to be created for the Acropolis Web site. Let's take a closer look at these layers.

- **Zeus in his chair.** This layer has a rotating image of Zeus. It will also slide into position from the side of the screen.

- **Zeus** word balloon. This layer will be invisible most of the time. It simply contains a word-balloon image with variable text when the mouse pointer is placed over Zeus.

- **Zeus's lightning bolt.** This layer will slide across the screen when the user selects a menu option to collect information about other gods or goddesses. This layer will be invisible when the document is loaded. It will be made visible only when the menu option is selected. When this layer leaves its "hiding place" behind Zeus and becomes visible, an audio file containing a short burst of thunder will be played.

- **Hermes symbol.** An image of Hermes. The wings on his feet will be animated; the rest of the image will be static.
- **Hermes word balloon.** When the mouse is placed over Hermes, a word balloon will appear that contains the text of the message Hermes is delivering to Zeus.
- **The menu.** The menu layer is set aside so it can be correctly positioned and sized when the visitor's screen size and resolution change.
- **The Greek building.** This layer uses colonnades as a backdrop for text that will change with menu requests.
- **Greek building text.** This layer contains text that will be placed between columns of the Greek building.
- **Copyright information.** This text will not change. It is placed in a layer strictly for ease of positioning on screens of various sizes and at various resolutions.

Now that each of the layers has been identified, it is necessary to place their proper cross-browser code in the HTML document as shown in Listing 6-2. As discussed in Chapter 1, Netscape uses the `<layer>` element to identify layers. Netscape will also recognize layers created with the `<div>` element. Microsoft only supports the use of the `<div>` or `<span>` elements to create layers as specified in the Web Consortium CSS Positioning Recommendation. For this reason, when creating layers you should always use the `<div>` element as the outlying containment box for the contents of each layer.

Listing 6-2. Identifying layers and their contents

```
<HTML>
<HEAD>
  <SCRIPT type="text/JavaScript">

If (NS4 || NS6) {
   docs =  "document";
   styles = " ";
   html = ".document";
   xposition = "e.pageX";
   yposition = "e.pageY";
} else if (IE4 || IE6) {
   docs =  "document.all";
   styles = ".style";
   html = " ";
   xposition = "event.x";
   yposition = "event.y";
}
  </SCRIPT>
  <STYLE>

//Cascading style sheet for the Acropolis Web site//
H1         { color: "red";
             font: "arial italic";
             font-size: "largest";
             text-align: "center";}
H2         { color: "red";
             font: "italic";
             font-family: "monospace";
             font-size: "largest";
             text-align: "left";
             text-indent: "30px";}
p:zeustxt { background: "red";
             border: "black";
             color: "black";
             font: "Verdana, normal";
             font-size: "larger";
             text-align: "left";}
p:hermestxt { background: "yellow";
               border: "blue";
               color: "black";
               font: "Times new Roman, normal";
               font-size: "smaller";
               text-align: "left";}
p:bldgtxt { background: "green";
             border: "blue";
             color: "black";
             font: "Arial, normal";
             font-size: "normal";
             text-align: "left";}
p:menutxt  { background: "black";
              border: "red";
              color: "white";
              font: "Times new Roman, normal";
              font-size: "larger";
              text-align: "left";}
  </STYLE>
<!— Add links to other scripts here —>
</HEAD>
```

```
<BODY>

      <!–       Zeus in his chair –>

<DIV id="zeuslayer" style = "…">
   <IMG src="images/zeus.gif" alt="Zeus, father of the gods, seated on
   Mt. Olympus" width=100 height=100 lowsrc="zeussm.gif">
</DIV>

      <!–       Zeus word balloon –>

<DIV id="zeusbbllayer"
     style = "background: images/zbubble.gif; …"
     align = "center">
      I am Zeus! Be wary of my wrath.
</DIV>

      <!–Zeus's lightning bolt–>

<DIV id="zeusltbltlayer" style = "…">
   <IMG src="images/lightening.gif" alt="Temper! Temper! Temper!"
   width=30 height=100 lowsrc="lightening.gif">
</DIV>

      <!–Hermes symbol–>

<DIV id="hermeslayer" style = "…">
   <IMG src="images/hermes.gif" alt="Hermes, the Messenger of the Gods."
   width=75 height=75 lowsrc="hermessm.gif">
</DIV>

      <!–Hermes word balloon–>

<DIV id="hermesbbllayer"
     style = "background: images/hbubble.gif; …"
     align = "center">
      Zeus is such a blow hard!
</DIV>

      <!–The menu–>

<DIV id="menulayer" style = "…">
</DIV>
```

```
      <!—The Greek building—>

<DIV id="greeklayer" style = "…">
   <IMG src="images/greek.gif" alt="Imagine a fabulous building sitting on
   Mt. Olympus" width=400 height=350 lowsrc=greeksm.gif">
</DIV>

      <!—Greek building text—>

<DIV id="greektext"
     style = "…"
      Here is the holding text for the informational screen
</DIV>

      <!—Copyright information—>

<DIV id="zeus" style = "…">
   <IMG src="images/zeus.gif" alt="Zeus, father of the gods, seated on
   Mt. Olympus" width=100 height=100 lowsrc="zeussm.gif">
</DIV>
</BODY>
</HEAD>
```