CHAPTER 49

Frames, Layers, and the Shell Game

In This Chapter

Showing elements

Hiding elements

Collapsible text

Moving elements

Useful object detection

The DOM and JavaScript make it possible for the contents of a browser page to resemble closely the interactivity and responsiveness of any other application on your desktop. When you see a file listing on your desktop, you can show or hide the files and folders within a given directory. Text can move about on the screen without the use of animated GIF files or specialized plug-ins and applets. With the DOM and JavaScript, you can simulate these types of dynamic responses to user interaction or create spontaneous behavior.

Defining Hidden Text

JavaScript is a powerful language. JavaScript enables you to define HTML elements right from the script, even if you don't define those elements in your HTML body. This probably isn't the way you want to do things, however. Why not?

- Writing the page takes longer
- Maintaining the page is more difficult
- Testing the page is more difficult

An easier way to create hidden text is to include anything you want hidden right in your HTML and then to define it with a style of `display: none`.

```
<DIV CLASS="leveltwo" STYLE="display:none">
<H1>Defining Hidden Text</H1>
<H2>Bringing hidden text into view</H2>
<H2>Designing an interactive table of contents
</H2>
<H2>Dynamically modifying styles</H2>
</DIV>
```

In this example, these four lines of H1 text will not appear until the display property is set to something other than *none*.

Bringing hidden text into view

To make your hidden text display, all you do is turn the display property of the `DIV` element that contains your hidden text to `"visible"` or `""` (empty quotes). The following JavaScript turns the previous hidden text into visible text or visible text into hidden text. In the first line, an `if` statement tests to see whether the element is visible. If the text is visible (`style.display==""`), then JavaScript makes the text invisible. If the text is invisible (`style.display=="none"`), then JavaScript sets the text to be visible.

```
if (thisChild.style.display == "" ) {
thisChild.style.display = "none"; }
else {
thisChild.style.display = ""; }
```

Note that *thisChild* is the variable name assigned to the H2 entries listed in the preceding example.

Designing an interactive table of contents

In the following example, the table of contents dynamically shows and hides section headings from Chapter 48 and this chapter. Figures 49-1 and 49-2 show examples of the table of contents fully collapsed and fully revealed.

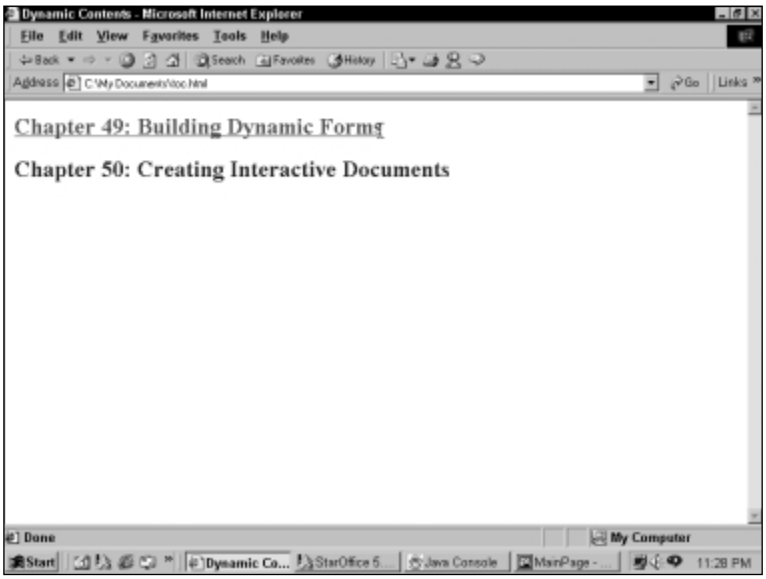

Figure 49-1: The table of contents collapsed

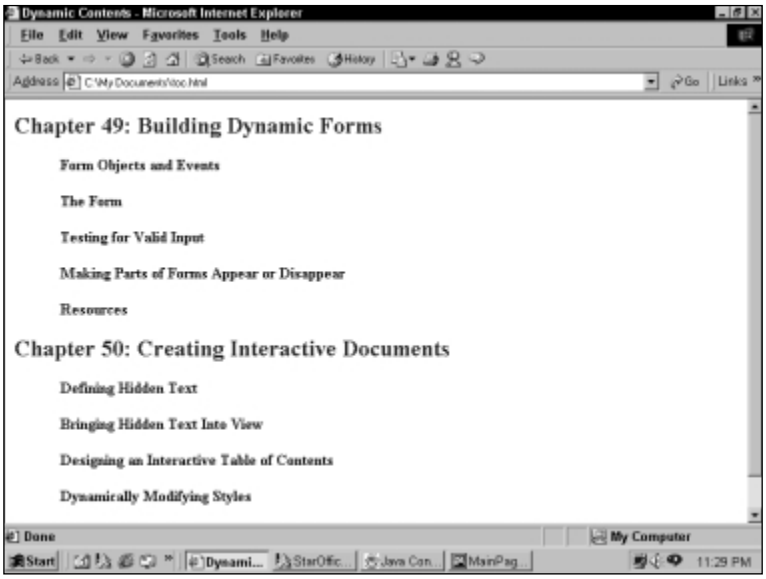

Figure 49-2: The table of contents with all text visible

The HTML used to accomplish the results shown in the preceding figures is as follows:

```
<HTML>
<HEAD>
<TITLE>Dynamic Contents</TITLE>
<STYLE TYPE="text/css">
  H1 {
color: blue;
font-size:18pt;
}
  H2 {
color: black;
font-size: 12pt;
}
  DIV.leveltwo {
margin-left: 0.5in;
}
</STYLE>
<SCRIPT Language="JavaScript">
<!--
... script goes here ...
-->
</SCRIPT>
</HEAD>
<BODY>
<DIV class="levelone" STYLE="position:absolute; width:550">
```

```
<DIV> <!-- although this DIV has no apparent purpose, it needs
to be here -->
<H1 onMouseOver="onLevelOne( this );"
  onMouseOut="notOnLevelOne( this );"
  onClick="hideContents( this );">Chapter 49: Building Dynamic
Forms</H1>
  <DIV CLASS="leveltwo" STYLE="display:none">
       <H2>Form Objects and Events</H2>
       <H2>The Form</H2>
       <H2>Testing for Valid Input</H2>
       <H2>Making Parts of Forms Appear or Disappear</H2>
       <H2>Resources</H2>
  </DIV>
</DIV> <!-- close for DIV with no apparent purpose -->
<DIV> <!-- DIV with no apparent purpose -->
<H1 onMouseOver="onLevelOne( this );"
  onMouseOut="notOnLevelOne( this );"
  onClick="hideContents( this );">Chapter 50: Creating
Interactive Documents</H1>
  <DIV CLASS="leveltwo" STYLE="display:none">
       <H2>Defining Hidden Text</H2>
       <H2>Bringing Hidden Text Into View</H2>
       <H2>Designing an Interactive Table of Contents</H2>
       <H2>Dynamically Modifying Styles</H2>
       <H2>The Script</H2>
  </DIV>
</DIV> <!-- close for DIV with no apparent purpose -->
</DIV> <!-- close for levelone DIV -->
</BODY>
</HTML>
```

Note these few things about the preceding HTML:

- The styles are defined right in the `HEAD`, rather than in a separate style sheet.
- The script is missing. Because this requires its own explanation, it will be outlined later in this chapter.
- Everything is contained in `DIV` elements. You can pretty much divide the page into a series of nested `DIV` elements. This is important to the way formatting works, as well as in the way text is made to appear and disappear. In fact, if you strip out all the rest of the content, you have the following sets of `DIV` elements:

```
<DIV class="levelone">
<DIV>
        <DIV class="leveltwo">
        </DIV>
   </DIV>
```

```
    <DIV>
        <DIV class="leveltwo">
        </DIV>
    </DIV>
</DIV>
```

- Some of the DIVs have no apparent purpose. We want to give you a good explanation for why they are needed, but we can't. This script simply won't work properly without them.
- The H1 elements have three event handlers on them: onMouseOver(), onMouseOut(), and onClick(). The first event handler, onMouseOver(), enables your script to do something when the mouse is over the contents of the H1 element — in this case, change the text color. The second event handler, onMouseOut(), enables your script to do something when the mouse is no longer over the contents of the H1 element — in this case, return to the original text color. The final event handler, onClick(), is the one you are most used to seeing. It actually calls the function that shows or hides the contents.
- This example includes the actual text that will show and hide. It would be more common actually to pull the text to be used in this table of contents from an external file or database. Exactly how you would do this depends on the technology you are using, which would be specific to your Web server.

Dynamically modifying styles

The presence of the onMouseOver() and onMouseOut() event handlers in the previous HTML suggest we can make changes to the contents of those elements or to other elements on the page based on where the mouse is. In fact, what we do in this example is change the color and decoration of the text in the H1 elements. Of course, you could change anything about the styles of the H1 elements.

The style definition of the H1 element is as follows:

```
H1 {
color: blue;
font-size:18pt;
}
```

Changing the value of the color property to red and adding a text decoration of underline is easily accomplished in JavaScript:

```
function onLevelOne( el ){
  el.style.color = "red";
  el.style.textDecoration = "underline";
  return;
}
```

Changing the style back to the original style is just as simple:

```
function notOnLevelOne( el ){
  el.style.color = "blue";
  el.style.textDecoration = "none";
  return;
}
```

The script

Finally, here is the entire JavaScript you need to include in the `SCRIPT` element in the previous HTML. Most of it should look familiar.

```
function onLevelOne( el ){
  el.style.color = "red";
  el.style.textDecoration = "underline";
  return;
}
function notOnLevelOne( el ){
  el.style.color = "blue";
  el.style.textDecoration = "none";
  return;
}
function hideContents( el ){
  var elParent = el.parentElement;
  var childrenCount = elParent.children.length;
  var thisChild = 0;
  for( i = 0; i<childrenCount; i++ ){
       thisChild = elParent.children(i);
       if (thisChild != el ){
            if (thisChild.style.display == "" )
                 thisChild.style.display = "none";
            else
                 thisChild.style.display = "";
       }
  }
  return;
}
```

The only function that should look unfamiliar to you is `hideContents()`. This function loops through all the children of the element that calls it (this means it loops through all the `DIV` elements nested within the `DIV` element of the element that calls it) and changes the display property for each one.

As mentioned in the previous two chapters, JavaScript and the implementation of the DOM vary for Microsoft Internet Explorer and Netscape Navigator. This example only works properly in Internet Explorer 4 and higher.

Moving Layers

We've played with hiding and displaying layers. Now let's see about moving one around on the screen. Microsoft includes a script on the DHTML section of their Web Workshop (under Positioning) called glide.html. This script shows how to move a layer across a page using JavaScript and the Microsoft DOM. The script can be found at http://msdn.microsoft.com/workshop/Author/dhtml/dhtml.asp

```
<HTML>
<HEAD><TITLE>Glide the DIV</TITLE>
<SCRIPT LANGUAGE="javascript">
var action;
function StartGlide() {
    Banner.style.pixelLeft = document.body.offsetWidth;
    Banner.style.visibility = "visible";
    action = window.setInterval("Glide()",50);
}
function Glide() {
    document.all.Banner.style.pixelLeft -= 10;
    if (Banner.style.pixelLeft<=0) {
        Banner.style.pixelLeft=0;
        window.clearInterval(action);
    }
}
</SCRIPT>
</HEAD>
<STYLE type="text/css">
DIV {
visibility:hidden;
position:absolute;
top:0;
left:0
}
</STYLE>
<BODY onload="StartGlide()">
<P>With dynamic positioning, you can move elements and their
content anywhere in the document even after the document has
loaded!
<DIV ID="Banner">Welcome to Dynamic HTML!</DIV>
</BODY>
</HTML>
```

This page works exactly as designed — but only on Internet Explorer (see Figure 49-3). On other JavaScript-enabled browsers that don't share the Microsoft DOM, the script does nothing or generates errors for the user (see Figures 49-4 and 49-5). Simply being inoperable is okay for the majority of users. Causing errors is to be avoided at all costs.

Figure 49-3: Glide.html, as created by Microsoft, works well on Internet Explorer.

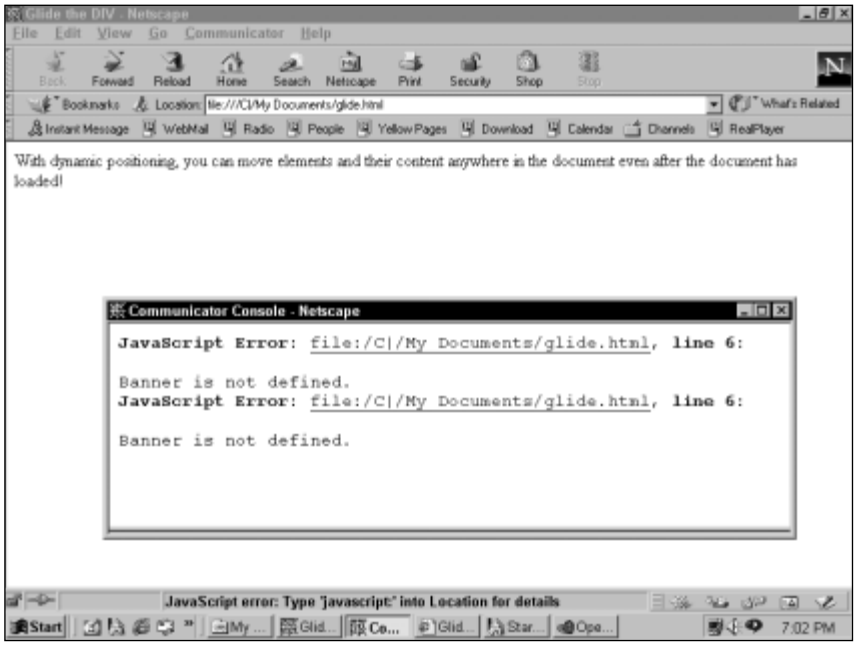

Figure 49-4: The same script creates errors in other browsers such as Netscape Navigator 4.5.

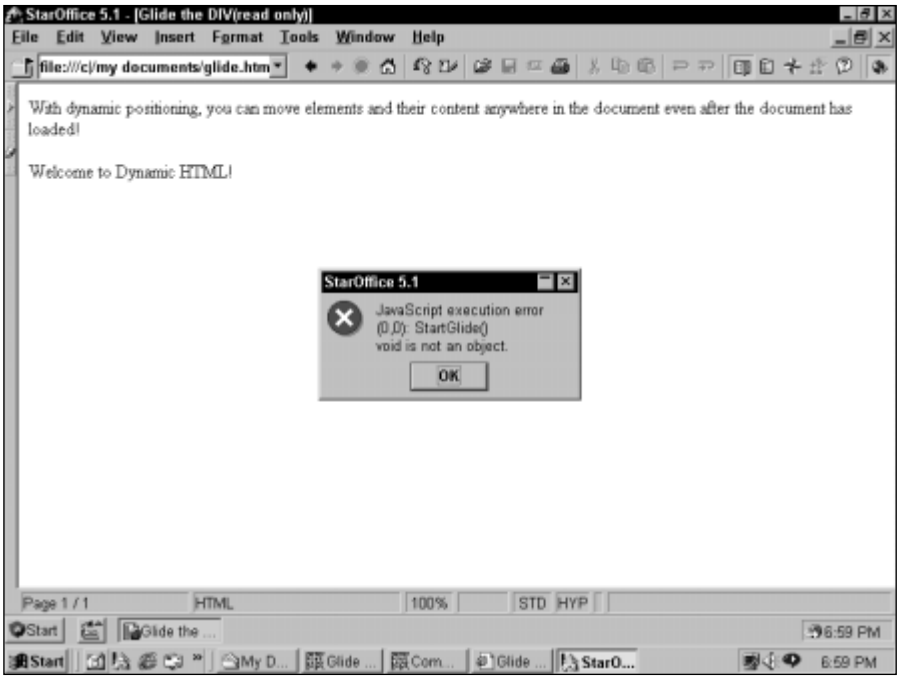

Figure 49-5: Other JavaScript-aware applications also break.

Creating cross-browser HTML

We need to work on making this script cross-browser compatible. We'll start with the HTML, which seems to put the page in reverse order. To make the elements stand out, we'll make the last line into an `H1` heading. All of the style information is also a little unnecessary, so we'll remove it and place an inline style attribute for absolute positioning.

Absolute positioning is required for dynamic placement of objects on a page.

```
<BODY onload="StartGlide()">
<DIV id=Banner style="position:absolute">
<H1>Welcome to Dynamic HTML!</H1>
</DIV>
<P style="position:absolute;top:50">With dynamic positioning,
you can move elements and their content anywhere in the
document even after the document has loaded!
</BODY>
```

Now, we need to go to work on the scripts. We'll add a variable declaration called `divBanner` to the top of the script declaration. This is the object that will be used to represent the layer during execution. It needs to be global to the script, so we place it outside both function declarations.

DOM object detection

The new `divBanner` variable becomes very important as we take a look at the opening lines of code for script declaration.

```
if (document.all) {
divBanner = document.all.Banner.style; //Line 10
} else if (document.layers) {
divBanner = document.Banner;
} else {
return;
}
```

Here's where we start detecting the environment to see what kind of browser we're working with. The Microsoft DOM includes objects from the body of the document under `document.all`. The first line checks for the existence of this object, and if it exists, assigns `divBanner` to the value of the object. In essence, this creates another handle we can use to manipulate the banner. The trailing `style` object is used by the Microsoft DOM as a prefix to the various style attributes of the object. Because all we'll be manipulating with this object is its style, we add it, also.

If the `document.all` object isn't valid, chances are we're working in a Netscape-compatible environment. Netscape doesn't use the `all` or `style` objects. It just appends the name of the object to `document`.

These are the only two current valid document objects to represent `Banner`. If neither of these shoes fit, then we should exit the function before doing anything else. All script execution will end at this point if neither of the DOM objects were detected.

Setting the initial position

Next, we need to determine where the right side of the screen is so we can position the left side of the banner at its starting point. Again, the Netscape and Microsoft DOMs differ on how to do this, so we use JavaScript's conditional assignment feature, checking to see which value is valid.

```
//                if the document.body object exists...
divBanner.left = (document.body) ?
document.body.offsetWidth : window.innerWidth;
// use the MS DOM              otherwise, use the NS DOM
```

Once this line executes, the left side of the banner is placed at the far left of the screen (out of sight). The next line sets the wheels in motion.

```
action = window.setInterval("Glide()",50);
```

The `action` object becomes a metronome that executes the `Glide` function every 50 milliseconds until it's stopped.

Moving the object

There's one last bit of detail to take care of. In the two DOMs, there are two methods for moving objects. The Microsoft way is to decrement the *pixelLeft* property, while Netscape uses a method called *moveBy*. We check for the existence of the *body* object to determine if we're in a Microsoft browser, and if not, we use the Netscape method.

```
if (document.body) {
  divBanner.pixelLeft -= 10;
} else {
  divBanner.moveBy(-10,0);
}
```

As the last action, we see if we've reached the left side of the window yet. If so, then the action object is disengaged and script execution stops. This new version of the script is only eight lines longer than the original. It still accomplishes the same purpose on Internet Explorer (see Figure 49-6), but now it also works on Netscape Navigator (see Figure 49-7), and it doesn't break StarOffice (see Figure 49-8).

Figure 49-6: Glide-crossbrowse.html still works as intended on Internet Explorer.

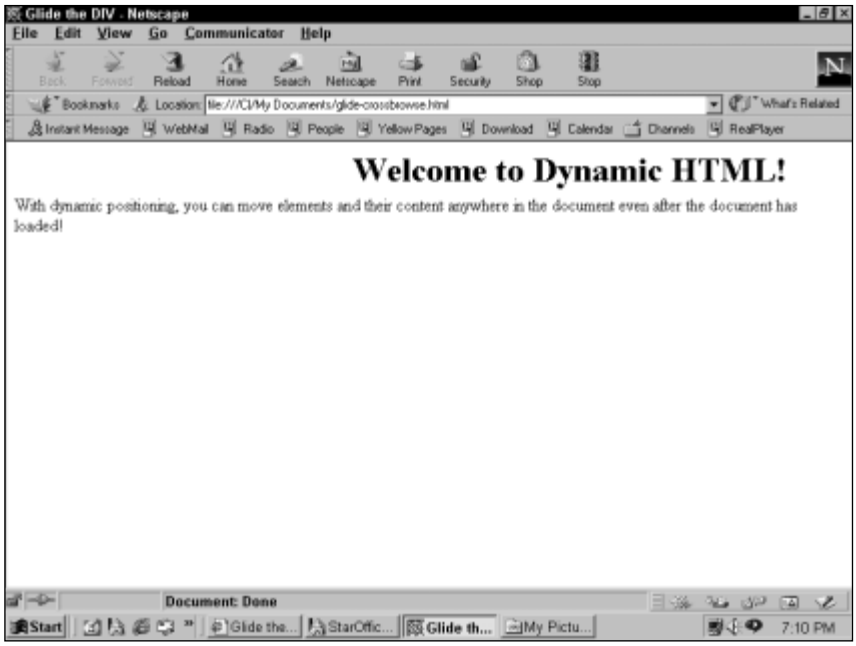

Figure 49-7: The same script, with object detection, also works on Navigator.

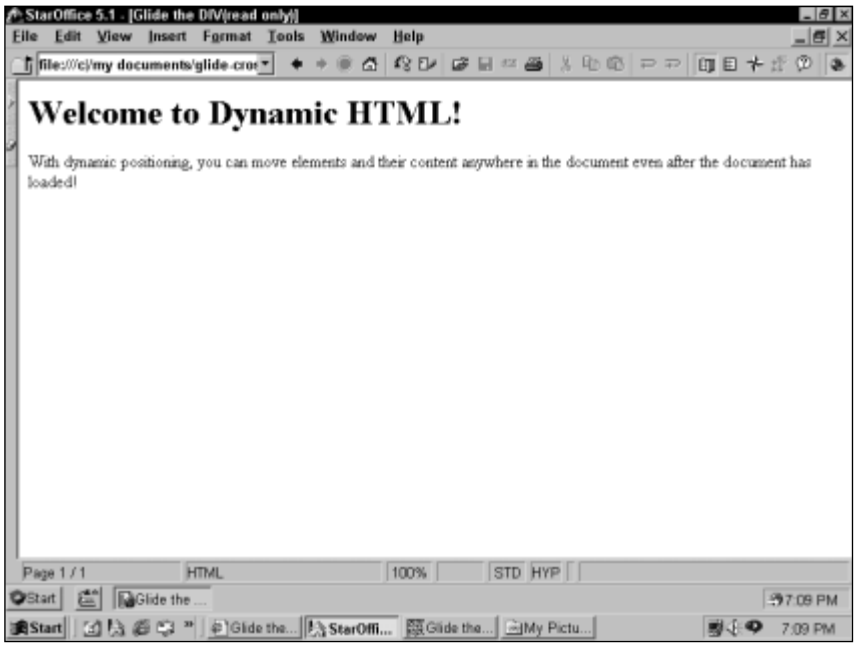

Figure 49-8: Although the dynamic movement doesn't happen, the page still displays in its final intended form without errors on StarOffice.

```
<HTML>
<HEAD>
<TITLE>Glide the DIV</TITLE>
<SCRIPT LANGUAGE="javascript">
var action;
var divBanner;
function StartGlide() {
    if (document.all) {
  divBanner = document.all.Banner.style;
    } else if (document.layers) {
  divBanner = document.Banner;
    } else {
  return;
    }
    divBanner.left = (document.body) ?
document.body.offsetWidth : window.innerWidth;
    divBanner.visibility = "visible";
    action = window.setInterval("Glide()",50);
}
function Glide() {
    if (document.body) {
  divBanner.pixelLeft -= 10;
    } else {
  divBanner.moveBy(-10,0);
    }
    if ((divBanner.left<=0) || (divBanner.pixelLeft<=0)) {
        divBanner.left=0;
        window.clearInterval(action);
    }
}
</SCRIPT>
</HEAD>
<BODY onload="StartGlide()">
<SPAN id=Banner style="position:absolute">
<H1>Welcome to Dynamic HTML!</H1>
</SPAN>
<P style="position:absolute;top:50">With dynamic positioning,
you can move elements and their content anywhere in the
document even after the document has loaded!
</BODY>
```

Macromedia Dreamweaver is a fine software tool for creating DHTML effects, such as moving text. The code Dreamweaver creates for events and moving elements is cross-browser compatible. If you're going to be doing a lot with DHTML, Dreamweaver is the tool to have.

From Here

Want to change styles with DHTML? Move ahead to Chapter 50.

Summary

In this chapter, you saw a simple example of a collapsible table of contents, along with a cross-browser version of moving text. Some of the properties should have looked familiar to you, but you saw them used in new ways. The complete working examples in this chapter can be used as a jumping-off point for your own dynamic Web pages.

With all this exposure to JavaScript, you are well-armed to make your pages dance. If you want to script your pages to the hilt, though, be well-advised to consult the resources on the Web for additional techniques and pitfalls waiting for you. Scripting exclusively for one browser may be easier, but your efforts will go unappreciated by a great many viewers.

✦ ✦ ✦