chapter

2 Image Rollovers

In This Chapter

- Project I: Image Rollover Script
- Project II: Multiple Image Rollover Script
- Project III: Random Banner Ad Rotator Script
- Recap
- Advanced Projects

You've gotten the ball rolling on the site and have shown that you've got what it takes to get the job done—now your boss wants you to wow him. He wants to make the site more dynamic and give it a little bit of "flash." He has checked out the competition's sites, and they all have image rollovers on theirs—JavaScript rollovers are used to swap an image with a different version of that image when the user moves the cursor over it. Conversely, when the user moves the cursor off of the image, the new image is replaced with the original. This technique is widely used on Web pages, and your boss feels this is just what your site needs to spruce it up.

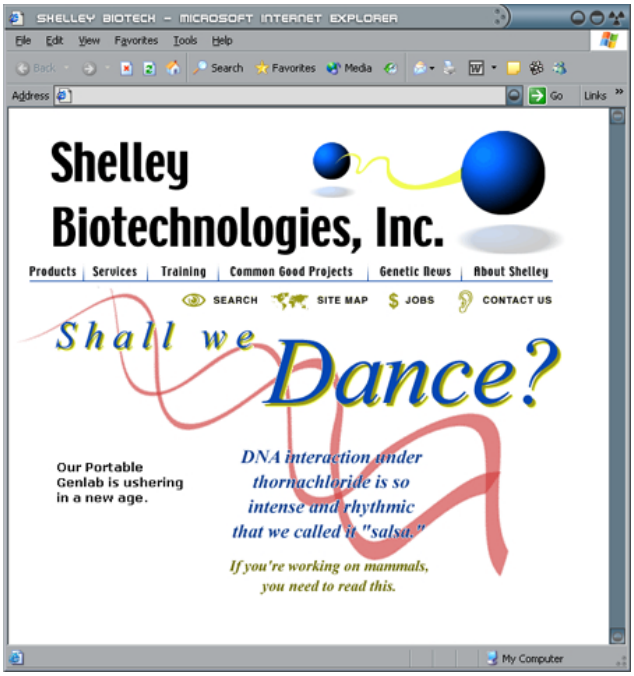

FIGURE 2–1 Shelley Biotech homepage.

The first thing to do is choose which graphics we want to be affected by the rollovers. The best place for image rollovers on most pages is on the main navigation images, and the Shelley Biotech site is no different (see Figure 2–1).

One of the designers has come up with a great graphic treatment for the rollovers (see Figure 2–2), and it's now up to you to implement them. There are three parts to creating an image rollover: Define the `IMAGE` objects, create the functions that will do the work, and insert the necessary JavaScript event handlers into your image and anchor tags. The first step in writing our script is the creation of the `IMAGE` objects.

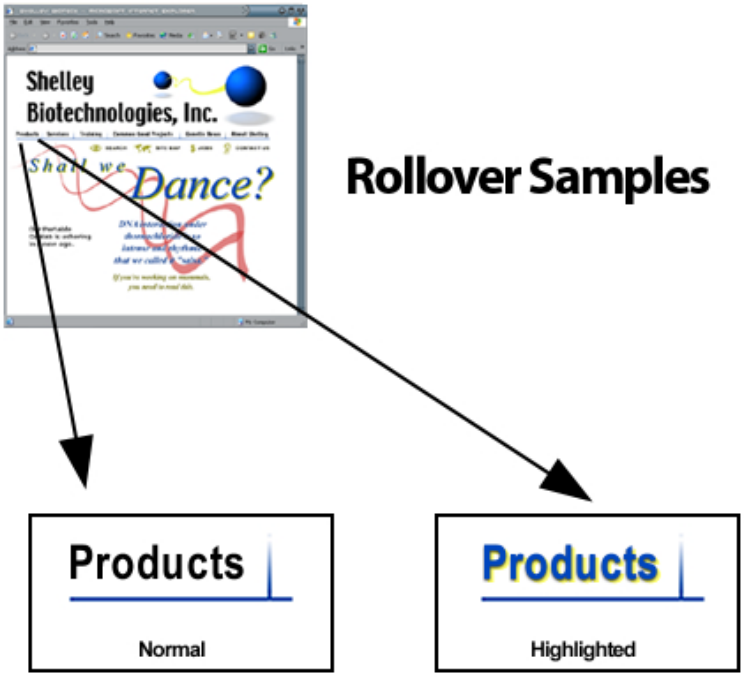

FIGURE 2–2 Image rollover sample.

◆ Project I: Image Rollover Script

Creating and Inserting the Image Objects

Before we get to the creation of the image objects themselves, we must decide where to put our script. Because the rollovers will be on every page of the site, it makes sense to place our script into an external JavaScript file. That way, we only need to place it in a single location instead of on each page. We have already created an external JavaScript file called *jsFunctions.js* for one of our

previous projects, so we can simply add our new script to that file. The home page already has the call to the external file, so we can just copy that `<script>` tag to all of the other pages on the site.

The first step in our script is to cloak for older browsers that don't support the `IMAGE` object, which is necessary for our rollover script. In Chapter 1 we learned how to use JavaScript to gather the browser and platform information. We could use a variation of that method in which we set up a bunch of `if` statements that test for every browser that doesn't support the `IMAGE` object; however, this would be loads of work. Fortunately for us, there is a simpler way. We can test if the user's browser supports rollovers using only a single `if` statement, as shown in the following lines of code:

```
   // Creation of the image objects
if (document.images) {
. . .
}
```

In the preceding lines of code we used

```
document.images
```

as the condition in our `if` statement. The condition returns a value of `true` if the browser supports the `IMAGE` object and a value of `false` if it does not. By inserting our code within this `if` statement, we are in effect cloaking our code from browsers that cannot handle the script.

When an HTML page is loaded and the browser creates the objects that make up the JavaScript hierarchy, every image laid out on the page is made into an object and put into an array called `images`. This is the first time we have come across arrays, so let's take a quick look at what an array is and, in particular, the `images` array and how we are going to use it.

You can think of an array as a filing cabinet. Let's say you have a page with four images on it. When the browser reads the HTML file, it goes down the page, and when it reaches the first image, it creates an `IMAGE` object for it and stores it in the first drawer of our filing cabinet. When it reaches the second image, it again creates a new `IMAGE` object, and then puts it into the second drawer of the filing cabinet, and so on until each of the images on the page has its own `IMAGE` object, which is stored in its own drawer.

When the browser needs to reference any of those images, it knows in which drawer to look for each of them. You, too, can reference the array and the IMAGE objects held therein—there are two ways to do this. The first is to reference the location in the array at which the IMAGE object resides. The problem with this method is that if you add an image somewhere on the page, the position of all of the images below it will change. This will cause you to go through your whole script and make sure an image you were calling isn't now in a different position. The second way, which we use in this script, takes care of this problem.

If you specify a NAME attribute in your <img> tags, you can then reference that graphic in the array by the name that you have assigned to it. With this method, even if you add other graphics to the page, you will still be able to reference the IMAGE object the same way, and your script won't be adversely affected.

There are six navigation images that will be affected by our rollover script, so for each of those images, we need to assign a NAME attribute in the <img> tags.

```
     <a href="products.html"><img
src="images/products_off.gif" width=71 height=33
border=0 name="Products" alt="Products"></a>

     <a href="services.html"><img
src="images/services_off.gif" width=67 height=33
border=0 name="Services" alt="Services"></a>

     <a href="training.html"><img
src="images/training_off.gif" width=75 height=33
border=0 name="Training" alt="Training"></a>

     <a href="common.html"><img
src="images/common_off.gif" width=157 height=33
border=0 name="Common" alt="Common Good"></a>

     <a href="genetic.html"><img
src="images/news_off.gif" width=98 height=33
border=0 name="News" alt="Genetic News"></a>

     <a href="about.html"><img
src="images/about_off.gif" width=106 height=33
border=0 name="About" alt="About Shelley"></a>
```

In the preceding lines of HTML, we added a NAME attribute to each of the six <img> tags. The name for each is the first word of the category that the graphic represents. JavaScript is case-sensitive, so

take notice that the first letter of each name is capitalized. Now that we have the names of these images squared away, we can move on to the creation of some new IMAGE objects that we will need for our script.

Because the images that we want to come up when the user rolls over one of the six graphics aren't explicitly placed on the page with HTML, we need to create two new IMAGE objects for each image we want to roll over. This will add these images to the images array and allow us to access their properties. Let's start by creating the IMAGE objects for the Products image.

```
   // Creation of the image objects
if (document.images) {
ProductsOn=new Image(71, 33);
. . .
}
```

In this first line, we initialize a new IMAGE object by using an image constructor, the syntax for which is

```
New Image(width, height);
```

By putting the image constructor on the right-hand side of the assignment operator, and the name that we wish the IMAGE object to have on the left, we have just created a new IMAGE object called ProductsOn.

The naming of these new objects is very important to the operation of the script. The first part of the name should be the same as the name of the image to which it corresponds that is already on the page. For example, in this first line, we created an object to hold the location of the rolled over version of the Products graphic, so the name starts with Products. Because this object holds the rolled over version of the graphic, the second part of the name is On. When combined, we have a new object with the name ProductsOn. For each of the six navigation images, we need not only an object to hold the rolled over version of the graphic but one to hold a regular version of the graphic as well. This second object's name for our Products image will also start with Products but will end with Off. Let's add this new IMAGE object after our first one.

```
   // Creation of the image objects
if (document.images) {
```

```
     ProductsOn=new Image(71, 33);
     . . .
     ProductsOff=new Image(71, 33);
     . . .
}
```

If you deviate from this naming convention, the functions that we will be writing won't know which `IMAGE` object to use for replacement, and you will get errors. We now have the two new `IMAGE` objects that we will need for the `Products` rollover, but at the moment they are empty. We need to assign actual images to them. We do this by assigning the location (URL) of the image to the source property of the `IMAGE` object, as follows:

```
   // Creation of the image objects
if (document.images) {
     ProductsOn=new Image(71, 33);
     ProductsOn.src="images/products_on.gif";

     ProductsOff=new Image(71, 33);
     ProductsOff.src="images/products_off.gif";
     . . .
}
```

We have completed `On` and `Off` objects for the `Products` rollover, but we now have to create objects for each of the other five image rollovers. We do this using the same naming conventions that we used for our first objects.

```
   // Creation of the image objects
if (document.images) {
     ProductsOn=new Image(71, 33);
     ProductsOn.src="images/products_on.gif";

     ProductsOff=new Image(71, 33);
     ProductsOff.src="images/products_off.gif";


     ServicesOn=new Image(67, 33);
     ServicesOn.src="images/services_on.gif";

     ServicesOff=new Image(67, 33);
     ServicesOff.src="images/services_off.gif";


     TrainingOn=new Image(75, 33);
```

```
        TrainingOn.src="images/training_on.gif";

        TrainingOff=new Image(75, 33);
        TrainingOff.src="images/training_off.gif";


        CommonOn=new Image(157, 33);
        CommonOn.src="images/common_on.gif";

        CommonOff=new Image(157, 33);
        CommonOff.src="images/common_off.gif";


        NewsOn=new Image(98, 33);
        NewsOn.src="images/news_on.gif";

        NewsOff=new Image(98, 33);
        NewsOff.src="images/news_off.gif";


        AboutOn=new Image(106, 33);
        AboutOn.src="images/about_on.gif";

        AboutOff=new Image(106, 33);
        AboutOff.src="images/about_off.gif";
}
 . . .
```

Creating the IMAGE objects in this manner also has another desired effect. When the browser creates the new IMAGE objects, it loads the images into its cache, thereby preloading it into memory. This preloading allows the image to display immediately upon rollover. It is important to keep this preloading in mind when creating your HTML pages—if you do not put `Height`, `Width`, and `alt` attributes into your `<img>` tags, some browsers will wait to display any of the page until all of the content and images are loaded into memory. You can run into some serious load times if you have a lot of IMAGE objects loading for rollovers in the background, so make sure you put all of the necessary tags into your HTML.

We have now created IMAGE objects for all of the graphics that we need to accomplish our rollovers. Our next step is the creation of the functions that will run our rollovers.

Image Rollover Functions

When the user rolls over one of our graphics, JavaScript event handlers call the functions we are going to write to actually perform the image replacements. Functions are an essential part of JavaScript; a function is a set of JavaScript statements that perform specific tasks. We cover both defining and calling functions in this example, but for now, let's concentrate on defining the functions we need for our rollovers. To define a function, you need four basic elements: the *function* keyword, a name for the function, a set of arguments that are separated with commas, and the statements that you want the function to execute.

We need to create two functions for our rollovers to work: one that switches to the `On` graphic when we roll onto an image and another to switch back to the `Off` graphic when we roll off the image. Generally, you should define the functions for a page in the `<head>` of your HTML, so let's start our functions right after our `IMAGE` objects.

```
// function to turn on rolled over graphic
function on(pic) {
. . .
}
```

The first step in defining a function is to call the function keyword followed by the name of our function; in this case, we'll call it `On`. Following the name, within its set of parentheses, we need to put a list of variables separated by commas, one for each argument we want to pass into the function. These variables will let us pass values into our function from the event handlers that call the function. We need to pass only a single argument into our function; we'll use the variable `pic` to hold that argument.

Next, we need to insert the set of instructions that we want the function to execute—these statements are enclosed within curly brackets.

```
// function to turn on rolled over graphic
function on(pic) {
     if (document.images) {
       document.images[pic].src=eval(pic + "On.src");
     }
}
```

The first of the statements that we want the function to execute is an `if` statement that will check for older browsers. If the

browser supports the IMAGE object, the next line will be executed. In this line, we tell the browser to replace the source of the image that is being rolled over with the source of the On version of the graphic. A lot is happening in this line, so let's break it down into smaller parts. First, let's look at the left-hand side of the assignment operator.

```
document.images[pic].src
```

We reference the source property of the image located at the position in the images array that has the value of the variable pic, so let's say the value of pic is Products. We would be referencing the source property of the Products IMAGE object and assigning it the value passed to it from the right-hand side of the assignment operator.

If we explicitly reference the images, like so:

```
document.product.src
```

we need separate functions for every image we wanted a rollover for—not the most efficient way to code. When scripting, you always should think about how you can most efficiently write your code. Any time you can use variables in this way, it will save you a lot of time and trouble. There are, however, occasions when you will want to call a specific image every time the function is called, and the preceding line of code is an example of how to do that.

On the right side of the operator, we use the eval() statement to combine the value of pic with a string to create a new value to assign as the source of the image being rolled over.

```
eval(pic + "On.src");
```

If the value of pic is Products, the eval() statement will return ProductsOn.src, and the value of the source of the ProductsOn IMAGE object will be assigned to the source of the Products IMAGE object. When this happens, the image on the page will change to the On image until the user rolls off.

When the user rolls off the image, another function will be called. This is the second function that we have to create—let's call this function Off.

```
// function to turn off rolled over graphic

function off(pic) {
```

```
    if(document.images) {
    document.images[pic].src= eval(pic + "Off.src");
   }
}
```

This function should look similar to your `On` function; in fact, the only difference is that the second half of what is being evaluated on the right-hand side of the assignment operator is `Off.src` instead of `On.src`. Therefore, when this line is evaluated, it will assign the source of the `ProductsOff IMAGE` object to be the source of the `Products IMAGE` object, thereby returning the image to its regular look.

Our functions are now complete and we are almost finished with the script. All that's left for us to do is to put the proper JavaScript event handlers into our HTML code.

Creating and Inserting the Event Handlers

You will find that most things in JavaScript are event-driven—events generally are the result of some action on the part of the user. If a user rolls over a link or changes the value of a text field, these would constitute events. To make use of these events with your JavaScript, you must define an event handler to react to these events. There are many predefined event handlers that you can use for this purpose; the two that we use for our rollovers are `onMouseOver` and `onMouseOut`. Let's first see what our existing HTML looks like without the event handlers.

```
<a href="products/index.html"><img
src="images/products_off.gif" width=71 height=33
border=0 name="Products" alt="Products"></a>
```

For our purposes, we must add the event handlers to the `ANCHOR` tag. First, we add the `onMouseOver` handler.

```
<a href="products/index.html"
onMouseOver="on('Products'); return true;">
```

When an event handler is called, it executes any JavaScript statements that follow the handler. We enclose these statements in quotes and separate them with semicolons. The first thing we want our event handler to do is call our `On` function. To call a function, you simply call its name and enclose any arguments you wish to pass along to the function in parentheses separated by commas.

```
on('Products');
```

The preceding statement calls the `On` function and passes it the value `Products`; again, we are passing it `Products` because that is the name of the image that we want our function to affect.

Another often seen addition to rollovers is the display of a relevant phrase in the browser's status bar. To do this, we add another statement to our event handler.

```
<a href="products/index.html" onMouseOver="on('Products');
window.status='Products'; return true;">
```

By calling on the `WINDOW` object's `status` property, we can display text in the browser's status bar by assigning it a new value. The last addition that we need to make to our event handler is the following statement:

```
return true;
```

This tells the JavaScript engine that this is the end of what needs to be executed in the event handler and it can go on with the rest of its business.

Well, one event handler down, and one more to go. Let's add our `onMouseOut` handler to take care of things when the user rolls off an image.

```
<a href="products/index.html" onMouseOver="on('Products');
window.status='Products'; return true; "
onMouseOut="off('Products'); window.status=' '; return
true;">
```

For our `onMouseOut` handler, we call the `Off` function, again passing it the name of the graphic that we want to affect. To turn off the phrase that we put into the status bar, we now reset the `window.status` to a blank value and then leave the handler with the `return` statement.

Now that we have inserted the event handlers for the `Products` image, we need to put some in for the rest of the images we want to have rollovers for. The syntax will be the same, the only difference being that you have to change the value that you are passing to the functions to the name of the graphic that you want to be changed.

```
<a href="products/index.html" onMouseOver="on('Products');
window.status='Products'; return true; "
```

```
onMouseOut="off('Products'); window.status=' '; return
true;"><img src="images/products_off.gif" width=71
height=33 border=0 name="Products" alt="Products"></a>

<a href="services/index.html" onMouseOver="on('Services');
window.status='Services'; return true; "
onMouseOut="off('Services'); window.status=' '; return
true;"><img src="images/services_off.gif" width=67
height=33 border=0 name="Services" alt="Services"></a>

<a href="training/index.html" onMouseOver="on('Training');
window.status='Training'; return true; "
onMouseOut="off('Training'); window.status=' '; return
true;"><img src="images/training_off.gif" width=75
height=33 border=0 name="Training" alt="Training"></a>

<a href="common/index.html" onMouseOver="on('Common');
window.status='Common Good Projects'; return true; "
onMouseOut="off('Common'); window.status=' '; return
true;"><img src="images/common_off.gif" width=157 height=33
border=0 name="Common" alt="Common Good"></a>

<a href="genetic/index.html" onMouseOver="on('News');
window.status='Genetic News'; return true; "
onMouseOut="off('News'); window.status=' '; return
true;"><img src="images/news_off.gif" width=98 height=33
border=0 name="News" alt="Genetic News"></a>

<a href="about/index.html" onMouseOver="on('About');
window.status='About Shelly Biotechnologies'; return true;
" onMouseOut="off('About'); window.status=' '; return
true;"><img src="images/about_off.gif" width=106 height=33
border=0 name="About" alt="About Shelley"></a>
```

Reviewing the Script

We should now have all of the elements that we need to get our Web site up and running with some great rollovers. Let's take a look at the completed script and go over what we did in each section.

First, we created the IMAGE objects that we need in our script.

```
   // Creation of the image objects
if (document.images) {
     ProductsOn=new Image(71, 33);
     ProductsOn.src="images/products_on.gif";

     ProductsOff=new Image(71, 33);
```

```
    ProductsOff.src="images/products_off.gif";


    ServicesOn=new Image(67, 33);
    ServicesOn.src="images/services_on.gif";

    ServicesOff=new Image(67, 33);
    ServicesOff.src="images/services_off.gif";


    TrainingOn=new Image(75, 33);
    TrainingOn.src="images/training_on.gif";

    TrainingOff=new Image(75, 33);
    TrainingOff.src="images/training_off.gif";


    CommonOn=new Image(157, 33);
    CommonOn.src="images/common_on.gif";

    CommonOff=new Image(157, 33);
    CommonOff.src="images/common_off.gif";


    NewsOn=new Image(98, 33);
    NewsOn.src="images/news_on.gif";

    NewsOff=new Image(98, 33);
    NewsOff.src="images/news_off.gif";


    AboutOn=new Image(106, 33);
    AboutOn.src="images/about_on.gif";

    AboutOff=new Image(106, 33);
    AboutOff.src="images/about_off.gif";
}
```

Here are the steps we took in creating the needed IMAGE objects:

1. We set up cloaking for browsers that do not support IMAGE objects.
2. We added the NAME attribute to the six images that we want to put rollovers on.
3. We created a set of two new IMAGE objects for each of the six images that will be affected by our rollover script.

4. We assigned each of the IMAGE object's source property the location of the proper image file.

Next, we created the functions that run our rollovers.

```
// function to turn on rolled over graphic
function on(pic) {
    if (document.images) {
     document.images[pic].src=eval(pic + "On.src");
    }
 }

// function to turn off rolled over graphic
function off(pic) {
     if(document.images) {
     document.images[pic].src= eval(pic + "Off.src");
    }
  }
```

Here are the steps we took in creating the rollover functions:

1. We first created a new function called `On` that changes the image the user is currently over to the rolled over version of that graphic.
2. We wrote a function called `Off` that changed the graphic back to its default state when the user rolled off of it.

Finally, we made the necessary changes to the `<a href>` tags in the existing HTML.

```
<a href="products/index.html" onMouseOver="on('Products');
window.status='Products'; return true; "
onMouseOut="off('Products'); window.status=' '; return
true;"><img src="images/products_off.gif" width=71
height=33 border=0 name="Products" alt="Products"></a>

<a href="services/index.html" onMouseOver="on('Services');
window.status='Services'; return true; "
onMouseOut="off('Services'); window.status=' '; return
true;"><img src="images/services_off.gif" width=67
height=33 border=0 name="Services" alt="Services"></a>

<a href="training/index.html" onMouseOver="on('Training');
window.status='Training'; return true; "
onMouseOut="off('Training'); window.status=' '; return
true;"><img src="images/training_off.gif" width=75
height=33 border=0 name="Training" alt="Training"></a>
```

```
<a href="common/index.html" onMouseOver="on('Common');
window.status='Common Good Projects'; return true; "
onMouseOut="off('Common'); window.status=' '; return
true;"><img src="images/common_off.gif" width=157 height=33
border=0 name="Common" alt="Common Good"></a>

<a href="genetic/index.html" onMouseOver="on('News');
window.status='Genetic News'; return true; "
onMouseOut="off('News'); window.status=' '; return
true;"><img src="images/news_off.gif" width=98 height=33
border=0 name="News" alt="Genetic News"></a>

<a href="about/index.html" onMouseOver="on('About');
window.status='About Shelly Biotechnologies'; return true;"
onMouseOut="off('About'); window.status=' '; return
true;"><img src="images/about_off.gif" width=106 height=33
border=0 name="About" alt="About Shelley"></a>
```

Here are the steps we took to insert the event handlers into the HTML:

1. We added the `onMouseOver` event handlers to each of the six images' `<a href>` tags. Within that handler we called the `On` function, passing it the name of the graphic that we were rolling over, changed the status bar to display a new message, and added the `return` command to exit the handler.
2. We added the `onMouseOut` event handlers to each of the six images' `<a href>` tags. Within that handler we called the `Off` function, passing it the name of the graphic that we were rolling over, reset the status bar to nothing, and added the `return` command to exit the handler.

We introduced several new aspects of JavaScript in this first script, including

- A new method of cloaking that looks for browsers that support specific objects that are needed in your script.
- An introduction to arrays, specifically the `images` array.
- How to create new `IMAGE` objects and how to change their source property.
- How to create a function.
- The concept of event handlers and how to use two of them: `onMouseOver` and `onMouseOut`.

- How to change the message being displayed in the status bar of the browser window.

◆ Project II: Multiple Image Rollover Script

It's that time of the year when the company puts out its annual report. This year the public relations department wants to make it available online. Most of it has already been posted, but there is one section that they could use a little help on. In the section of the report that deals with the sales figures, there is a set of graphs that compares sales over the last three years. The public relations department would like it if users could compare the charts for adjoining years. In other words, they want to be able to compare 1998 to 1999, and 1999 to 2000, and finally 2000 to 2001. After seeing the magic you've worked with the navigation rollovers, they wonder if you can set up this functionality with JavaScript rollovers instead of just laying out all of the graph comparisons in a huge page.

After some research and a little tinkering around with one of the designers, you think you have a way to give them what they are looking for, so you show them what you've worked out. Figure 2–3 shows what the page would look like before the user rolls over one of the year sets. Figure 2–4 shows what would happen if the user rolls over 2000–2001. They love the idea and tell you to get started on it.

First, let's look at the HTML for the section of the page that we are concerned with.

```
<tr>

     <td valign="top" align="left">
          <br><br><img
src="images/sales_comp_left_default.gif" alt=""
width="162" height="87" border="0"></td>

     <td valign="top" align="center">
          <font class="blkBldText">Year to Year Sales
Comparison</font><br>
           <br>
           <a href=" " class="blueLink">2000 - 2001</a><br>
           <br>
           <a href=" " class="blueLink">1999 - 2000</a><br>
```

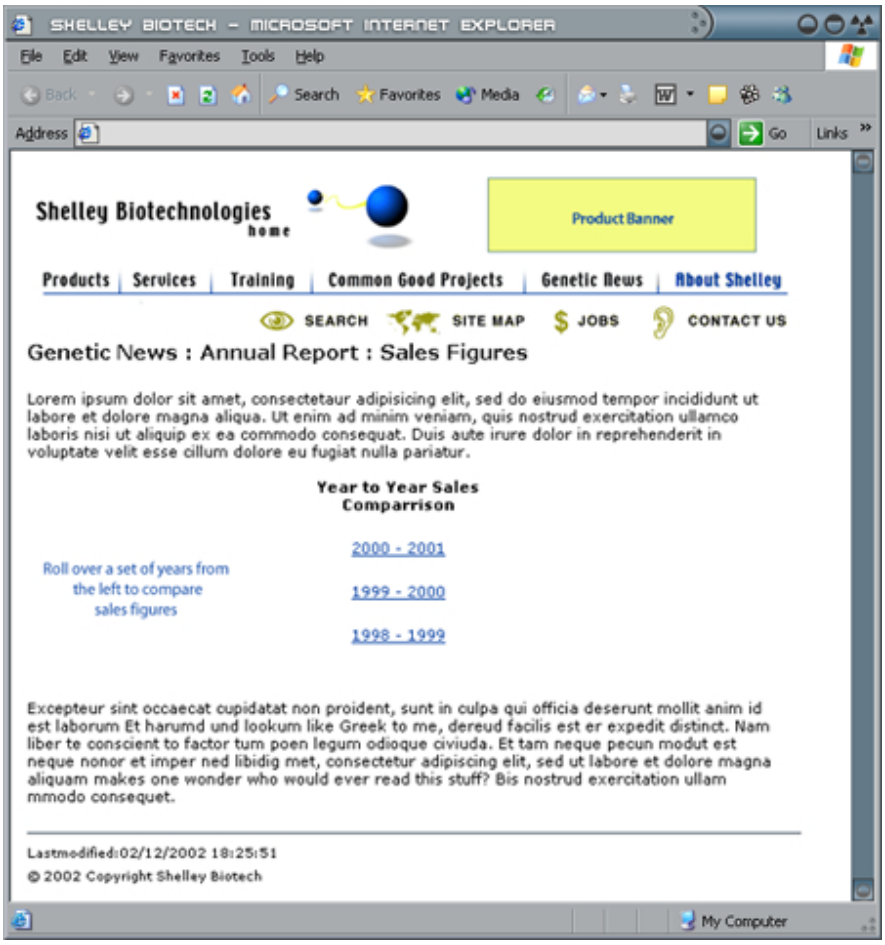

FIGURE 2–3 Shelley Biotech annual report page.

```
            <br>
            <a href=" " class="blueLink">1998 - 1999</a></td>
      <td vlaign="top" align="left">
            <br><br><img src="images/
sales_comp_right_default.gif" alt="" width="162"
height="87" border="0"></td>
</tr>
```

The designer on the project has gone ahead and created some temporary images for us to use until they get the final chart graphics done. In the existing HTML, you will notice that there

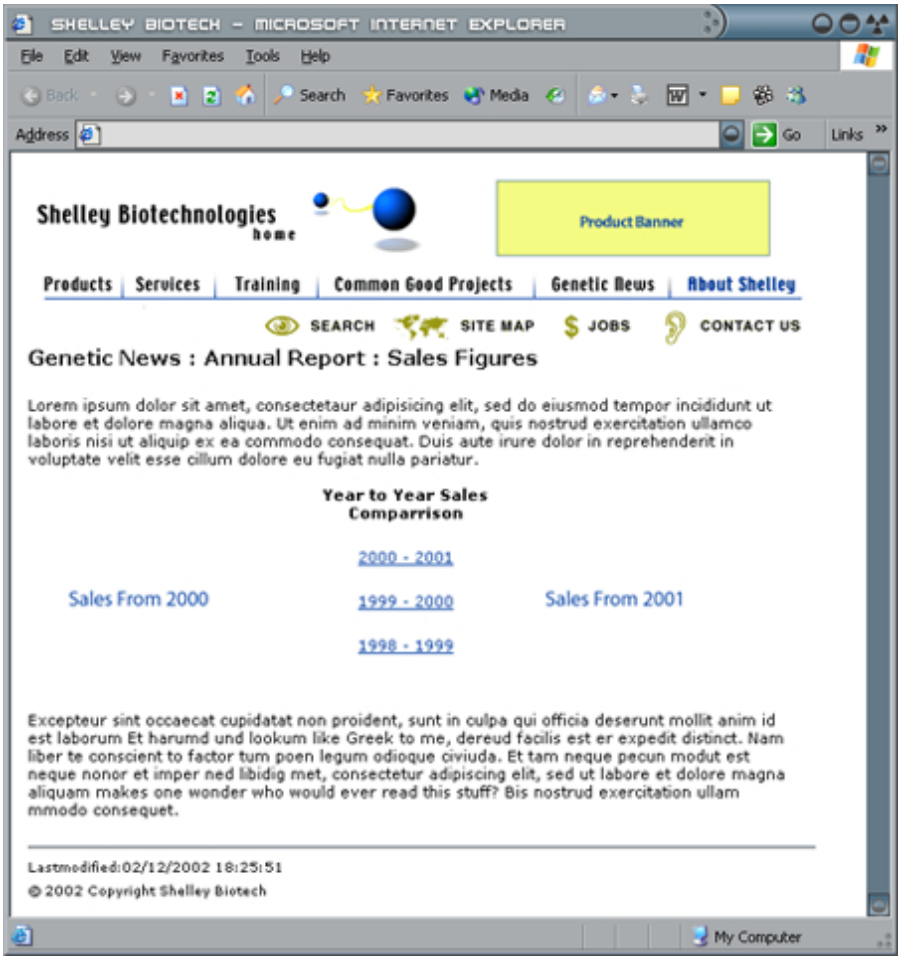

FIGURE 2–4 Annual report page—with a year set highlighted.

are two images that we will have to switch out when the user rolls over a set of years: `sales_comp_left_default.gif` and `sales_comp_right_default.gif`. At the moment, the image that goes on the left has instructions for the user, while the image on the right is a plain white image. When a user rolls over a set of years, the image on the left will change to a chart of the first year in the set, while the image on the right will change to a chart of the second year in the set. Luckily, this is fairly similar to what we did in the previous project, so we'll be able to use the functions we wrote for the navigation rollovers as a starting point for our func-

tions here. But first we need to create the image objects we will need for the project.

Creating and Inserting the Image Objects

As stated above, the designer has created temporary chart graphics for each of the four years we need to use in our comparisons. They are called:

```
sales_1998.gif
sales_1999.gif
sales_2000.gif
sales_2001.gif
```

To use them in our script, we need to create an `IMAGE` object for each of them. Because this script will be used only on this page, there is no need to put our script into an external JavaScript file, so let's insert it into the `<head>` of the HTML page.

```
<head>
<script Language="JavaScript1.2">
<!-- Code after this will be ignored by older browsers

// Creating Image Objects for Rollovers
if (document.images) {
sales2001=new Image(162, 87);
sales2001.src="images/sales_2001.gif";

sales2000=new Image(162, 87);
sales2000.src="images/sales_2000.gif";

sales1999=new Image(162, 87);
sales1999.src="images/sales_1999.gif";

sales1998=new Image(162, 87);
sales1998.src="images/sales_1998.gif";
}

    . . .
// -->
</script>
```

Just as in our previous project, we've inserted the script tag, cloaked the script from older browsers, and created the `IMAGE` objects that we'll need for our script.

Now that we have all of the IMAGE objects that we will need, let's go on to the next step of our script—modifying the rollover functions.

Inserting the Rollover Functions

Let's take a look at the functions we are using for our navigation rollovers. First, the On function will replace the image of the category name that the user rolls over with a highlighted image. Then the Off function changes the image back to the default version once the user rolls off of it. What's different in our current project is that we need the script to swap out not one but two images each time the On function is called; likewise, when the user rolls off of a set of years, the Off function needs to turn both of the changed images back to the default versions that show up when the page is loaded. To get started, let's copy the On and Off functions from the *jsFunctions.js* file and paste them underneath our new IMAGE objects.

```
<head>
<script Language="JavaScript1.2">
<!-- Code after this will be ignored by older browsers

// Creating Image Objects for Rollovers
if (document.images) {
sales2001=new Image(162, 87);
sales2001.src="images/sales_2001.gif";

sales2000=new Image(162, 87);
sales2000.src="images/sales_2000.gif";

sales1999=new Image(162, 87);
sales1999.src="images/sales_1999.gif";

sales1998=new Image(162, 87);
sales1998.src="images/sales_1998.gif";
}


// function to turn on rolled over graphic
function on(pic)  {
     if (document.images)  {
           document.images[pic].src=eval(pic + "On.src");
     }
}

// function to turn off rolled over graphic
```

```
function off(pic)  {
     if(document.images)  {
          document.images[pic].src= eval(pic + "Off.src");
     }
}
```

The first thing we need to do is change JavaScript comments and the names of the functions so that they don't conflict with the functions that control the navigation rollover functions. Let's call these new functions `salesOn` and `salesOff`.

```
// function to turn on sales comparison charts
function salesOn(pic)  {
     if (document.images)  {
           document.images[pic].src=eval(pic + "On.src");
     }
}

// function to turn off sales comparison charts
function salesOff(pic)  {
     if(document.images)  {
          document.images[pic].src= eval(pic + "Off.src");
     }
}
```

Now, let's make the changes to the `salesOn` function. Because we need this function to change two graphics, we need to create two variables to accept values passed in from the event handlers: one for the name of the image to be swapped out for the left-hand image and one for the name of the image that will show up on the right-hand side. Let's call these variables `salesPic1` and `salesPic2`. These will replace the `pic` variable in our initial function declaration.

```
// function to turn on sales comparison charts
function salesOn(salesPic1, salesPic2)  {
     if (document.images)  {
           document.images[pic].src=eval(pic + ".src");
           . . .
     }
}
```

Next, we need to modify the line of code that will change out the image that will show up on the left-hand side. Now, because we are always going to be changing the same image on the left-hand side, we can specify exactly which image we want changed

instead of using a variable as we did for the navigation images. So, where we had specified

```
document.image[pic].src
```

in the line that changes the navigation image, we will specify the image called `salesLeft` in our new line of code.

```
document.images['salesLeft'].src
```

On the right-hand side of the assignment operator, however, the image we want to show up will depend on which set of years the user has rolled over. We will use the `salesPic1` variable that is being passed into the function from the event handler to specify which year `IMAGE` object we want to use as the source for the image.

```
=eval(salesPic1 + ".src");
```

Let's see what our modified line of code looks like in the script.

```
// function to turn on sales comparison charts
function salesOn(salesPic1, salesPic2)  {
     if (document.images)  {

     document.images['salesLeft'].src=eval(salesPic1 +
".src");
            . . .
     }
}
```

This newly modified line will take care of turning on the proper year graphic for the left-hand side; now we need to add another line that will do the same for the right-hand side.

```
// function to turn on sales comparison charts
function salesOn(salesPic1, salesPic2)  {
     if (document.images)  {

     document.images['salesLeft'].src=eval(salesPic1 +
".src");
     document.images['salesRight'].src=eval(salesPic2 +
".src");
      }
}
```

This line is pretty much identical to the first line, except that we are specifying the `salesRight` image to change and for the script to use the year held in `salesPic2` as the new `IMAGE` object.

That will do it for the `On` function. Let's move on to the `Off` function for the last step in our script functions. Here is the current `Off` function.

```
// function to turn off sales comparison charts
function salesOff(pic)  {
     if(document.images)  {
          document.images[pic].src= eval(pic + "Off.src");
   }
}
```

Now let's add the two lines of code that will turn the chart graphics back to their default states once the user has rolled off the set of years.

```
function salesOff(pic) {
     if(document.images) {
        document.images['salesLeft].src= "images/
sales_comp_left_default.gif";
        document.images['salesRight'].src= "images/
sales_comp_right_default.gif";
     }
}
```

As in our `On` function, on the left-hand side of the assignment operator we are explicitly telling the JavaScript engine which images we want to change. On the right-hand side, though, we are doing something new: In the past we have always specified for the engine to look in the `source` property of another image object for the `IMAGE` we want to replace. However, because no matter which set of years the user has rolled over, we want to replace it with the original default images, when the user rolls off, we can explicitly tell the JavaScript engine the path to those images, which is what we have done above.

Well, that's it for the changes to the functions; we now simply have to add the event handlers to the right links and we'll be done.

Creating and Inserting the Event Handlers

We will not be swapping out an image that the user has rolled over, as we did in the previous project. Instead, we will swap out two other images when the user has rolled over a text link, so we will place the event handlers inside `<a href>` tags that are around the text instead of around an image. The event handlers don't really care what is used as the rollover focus, so we can put them

in most any object on a page. In this case we're using the text links of the years that the sales charts compare.

```
<a href=" " class="blueLink">2000 - 2001</a><br>
<br>
<a href=" " class="blueLink">1999 - 2000</a><br>
<br>
<a href=" " class="blueLink">1998 - 1999</a>
```

Before we insert the handlers, we first need to add the `name` attribute to the left- and right-hand graphics that we are specifying in our functions.

```
     <td valign="top" align="left">
          <br><br><img src="images/
sales_comp_left_default.gif" alt="" width="162"
height="87" border="0" name="salesLeft"></td>

     <td valign="top" align="center">
          <font class="blkBldText">Year to Year Sales
Comparrison</font><br>
          <br>
          <a href=" " class="blueLink">2000 - 2001</a><br>
          <br>
          <a href=" " class="blueLink">1999 - 2000</a><br>
          <br>
          <a href=" " class="blueLink">1998 - 1999</a></td>
     <td valign="top" align="left">
          <br><br><img src="images/
sales_comp_right_default.gif" alt="" width="162"
height="87" border="0" name="salesRight"></td>
```

With that done, we can move on to inserting the handlers. The event handlers are going to be very similar to the handlers that we used in our navigation rollovers; there will be both an `onMouseOver` and `onMouseOut`. The difference is that we will be calling our new sales functions instead of the rollover functions. We will also not worry about changing the text that shows up in the status bar of the browser. Let's look at link for the years 2000–2001 with the new handlers inserted.

```
<a href=" " class="blueLink"
onMouseOver="salesOn('sales2000', 'sales2001'); return
true;"  onMouseOut="salesOff(); return true;">2000 - 2001
</a>
```

In the `onMouseOver` handler, we are calling the `salesOn()` function and passing it the two years that we want to compare, first the year we want to show up on the left-hand side, then the year that shows up on the right-hand side. To pass multiple values into a function, you simply need to separate them with a comma. In the `onMouseOut` handler, we are calling the `salesOff()` function without passing any values into it. This is because we already know which images we need to change and what we need to change them back to. Let's add the handlers to the two other text links on the page.

```
<a href=" " class="blueLink"
onMouseOver="salesOn('sales2000', 'sales2001'); return
true;"  onMouseOut="salesOff(); return true;">2000 - 2001
</a><br>
<br>
<a href=" " class="blueLink"
onMouseOver="salesOn('sales1999', 'sales2000'); return
true;"  onMouseOut="salesOff(); return true;">1999 - 2000
</a><br>
<br>
<a href=" " class="blueLink"
onMouseOver="salesOn('sales1998', 'sales1999'); return
true;"  onMouseOut="salesOff(); return true;">1998 - 1999
</a>
```

We now have handlers inserted for the other two sets of years that we want to compare, which wraps it up for this project. The public relations department will be thrilled with the results and no doubt will soon be looking for other ways we can enhance the annual report. Before they find anything else for us to do, let's go back over how we completed this script.

Reviewing the Script

While some of the functionality for our script was repurposed from our existing rollovers on the homepage, we made some changes and added some new functionality to our rollovers, which deserve to be gone over again. Let's review exactly what we did for this project.

First, we worked with the designer to come up with a new layout for the section and recoded the HTML to match the new layout.

```
<tr>

     <td valign="top" align="left">
```

```
          <br><br><img src="images/
sales_comp_left_default.gif" alt=""td>

     <td valign="top" align="center">
          <font class="blkBldText">Year to Year Sales
Comparison</font><br>
          <br>
          <a href=" " class="blueLink">2000 - 2001</a><br>
          <br>
          <a href=" " class="blueLink">1999 - 2000</a><br>
          <br>
          <a href=" " class="blueLink">1998 - 1999</a></td>
     <td valign="top" align="left">
          <br><br><img src="images/
sales_comp_right_default.gif" alt="" width="162"
height="87" border="0"></td>
</tr>
```

After redoing the HTML, we needed to create four new IMAGE objects to hold the years' sales chart graphics.

```
<script Language="JavaScript1.2">
<!-- Code after this will be ignored by older browsers

// Creating Image Objects for Rollovers
if (document.images) {
sales2001=new Image(162, 87);
sales2001.src="images/sales_2001.gif";

sales2000=new Image(162, 87);
sales2000.src="images/sales_2000.gif";

sales1999=new Image(162, 87);
sales1999.src="images/sales_1999.gif";

sales1998=new Image(162, 87);
sales1998.src="images/sales_1998.gif";
}
```

Here are the steps we took in creating the IMAGE objects:

1. We added a new image for each of the years that we needed a sales chart graphic.
2. We modified the functions that power the script.

```
// function to turn on rolled over graphic
function salesOn(salesPic1, salesPic2)  {
     if (document.images)  {
```

```
     document.images['salesLeft'].src=eval(salesPic1 +
".src");

     document.images['salesRight'].src=eval(salesPic2 +
".src");
     }
}

// function to turn off rolled over graphic
function salesOff()  {
     if(document.images)  {
          document.images['salesLeft'].src= "images/
sales_comp_left_default.gif";
          document.images['salesRight'].src= "images/
sales_comp_right_default.gif";
    }
}
// End cloaking from non JavaScript browsers -->
</script>
```

We had to do several things to get our functions to where they needed to be for this project:

1. We copied our `On` and `Off` functions from our navigation rollovers and renamed them `salesOn` and `salesOff`.
2. We modified the function to accept two values instead of one to be passed into the function held in the variables `salesPic1` and `salesPic2`.
3. We modified the statements within our `salesOn` function. In the first line, we changed the code to specify `salesLeft` as the image we would be swapping out and for it to use the `IMAGE` object specified in the `salesPic1` variable. We then copied that line of code and changed it to change the `salesRight` image and use the value in the variable `salesPic2` for the new `IMAGE` object.
4. We modified the `salesOff` function. First, we removed the `pic` variable from the function declaration, as we will not need to pass in any values for this function to work. Next, we modified the first line of code to specify the `salesLeft` as the image we would be swapping out, and for it to return to the image `sales_comp_left_default.gif` when the user triggered this function. We then copied that line of code and changed it to change the `salesRight` image and for it to return to the image `sales_comp_right_default.gif` again when the user triggered this function.

Finally, we inserted our event handlers into the HTML.

```
<td valign="top" align="left">
      <br><br>
      <img src="images/sales_comp_left_default.gif" alt=""
width="162" height="87" border="0" name="salesLeft"></td>

<td valign="top" align="center">
      <font class="blkBldText">Year to Year Sales
Comparison</font><br>
      <br>
      <a href=" " class="blueLink"
onMouseOver="salesOn('sales2000', 'sales2001'); return
true;"  onMouseOut="salesOff(); return true;">2000 - 2001
</a><br>
      <br>
      <a href=" " class="blueLink"
onMouseOver="salesOn('sales1999', 'sales2000'); return
true;"  onMouseOut="salesOff(); return true;">1999 - 2000
</a><br>
      <br>
      <a href=" " class="blueLink"
onMouseOver="salesOn('sales1998', 'sales1999'); return
true;"  onMouseOut="salesOff(); return true;">1998 - 1999
</a></td>

<td valign="top" align="left">
      <br><br>
      <img src="images/sales_comp_right_default.gif" alt=""
name="salesRight" width="162" height="87" border="0"></td>
```

1. We added the name attributes `salesLeft` and `salesRight` to the left- and right-hand graphics, respectively.
2. We added the `onMouseOver` and `onMouseOut` event handlers to each of the three text links. In the `onMouseOver` handler, we inserted a call to the `salesOn()` function and passed it the two years of sales charts that we wanted to compare. In the `onMouseOut` handler, we inserted a call to the `salesOff()` function and passed in no values, as none are needed to return the graphics to their default state.

We have introduced several new aspects of JavaScript in this script, including

- Repurposing existing script elements for use with new scripts.
- Passing multiple values into a function.

- Explicitly specifying the name of the image we wish to change.
- Explicitly specifying the location of the image we want to use as the source for an image.

◆ Project III: Random Banner Ad Rotator Script

Word of your skills with JavaScript and images has spread to the marketing department, and they have tracked you down for a project that is near and dear to their hearts. Currently on the site, at the top of every secondary page, there is a little banner advertisement for one of the company's products that, if clicked, will take the user to the store section of the site (see Figure 2–5). The way it is set up now is that if they want to have different products showing up, they need to hardcode which product banner goes on a specific page. Because it's quite a hassle to keep track of multiple images and which page they show up on, up until now they have settled for having just a single product banner throughout the site. After seeing what you've done on the site, they're hoping that you can give them the ability to have multiple banners up on the site, so they can showcase a broader range of products. You've given it some thought and looked around on the Internet for similar functionality on other sites, and it looks as if you've come up with a good idea.

The plan is to set up a script that will not only randomly insert a different product banner on the page each time the page is loaded by a user, but rotate which banner is showing every 20 seconds while the user is on the page. So, if the marketing people want to showcase five different products, all they'll have to do is create the five graphics, drop them in a specified directory, and our script will randomly choose one of those five images to place on a given page. The first step is to find out the number of different banners the marketing department wants to have up on the site. After a quick call to marketing, you find out they indeed need five different product banners. Armed with that knowledge, it's time to get going on our script.

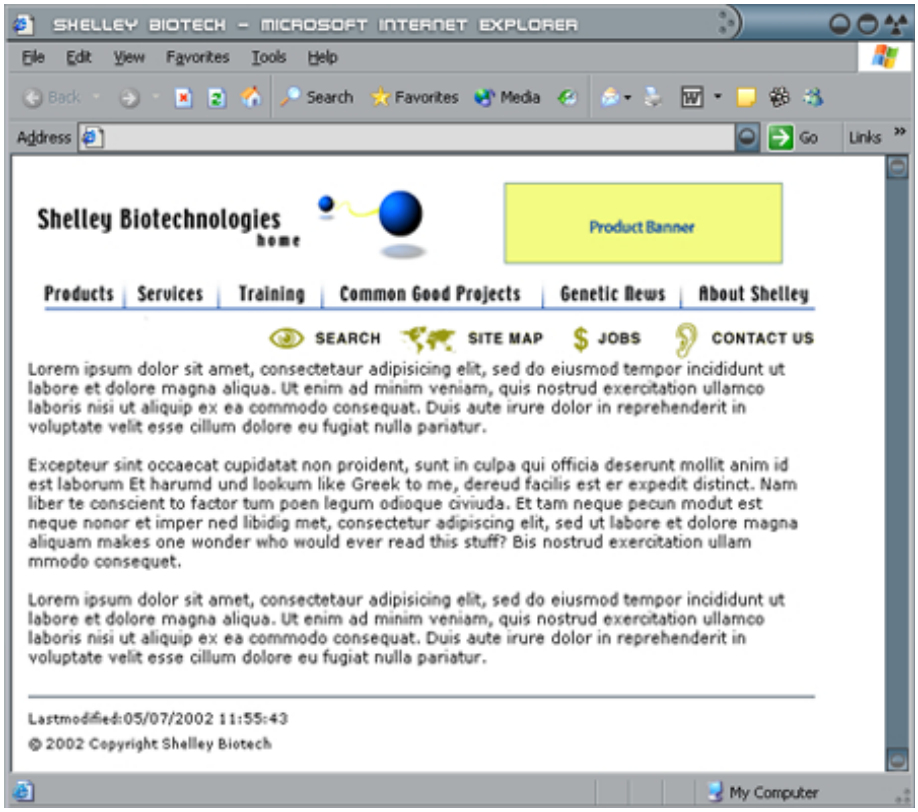

FIGURE 2–5 Secondary page with product banner.

Creating and Inserting the Image Objects

Because this is a script that is going to be used on all pages of the site except the home page, it makes sense to put it into our external JavaScript file, so that's where we will be working for the most part in this project. Now, as in the other scripts that we've done that have worked with image replacement, we need to create some IMAGE objects, one for each product banner that we want available to our script. Let's start by creating the five IMAGE objects that we'll need. Unlike in our previous scripts, we don't have the

FIGURE 2–6 Temporary product banner images.

graphics that we'll be actually using up on the site yet—the marketing folks are still trying to figure out which products to highlight. We've had one of the designers create five temporary graphics that we can use for now (see Figure 2–6).

Because we don't know the names of the products that are going to be used and because marketing may want to occasionally change which products are showcased, we've named the graphics as follows:

```
product_banner_1.gif
product_banner_2.gif
product_banner_3.gif
product_banner_4.gif
product_banner_5.gif
```

This not only will make it easier to write our script, but it will make for less work later on when marketing wants to switch the products that they are showcasing. The reason for this will become clear when we get down to the function that will be running most of our script. But first, now that we have graphics to use in building our IMAGE objects, let's get that done. As we will be placing this script into our *jsFunctions.js* file, we'll place it right below our navigation rollover functions.

```
// Creating Image Objects for Product Banners
if (document.images) {
product1Banner=new Image(200, 56);
product1Banner.src="images/product_banners/
product_banner_1.gif";

product2Banner=new Image(200, 56);
product2Banner.src="images/product_banners/
product_banner_2.gif";

product3Banner=new Image(200, 56);
product3Banner.src="images/product_banners/
product_banner_3.gif";

product4Banner=new Image(200, 56);
product4Banner.src="images/product_banners/
product_banner_4.gif";

product5Banner=new Image(200, 56);
product5Banner.src="images/product_banners/
product_banner_5.gif";
}
. . .
```

We have created five `IMAGE` objects named in numeric order `product1Banner` through `product5Banner`, and we told the marketing department that the images they should use are in a subdirectory of the images directory called `product_banners`. This step should look pretty familiar, as we used it in the last two projects. Now that we have our `IMAGE` objects, let's move on the creation of the function that will run the script.

Random Image Generator Function

The first step we need to take in our script is to figure out which graphic will show up when the page first loads, because there are five images to choose from. We need to generate a random number between 1 and 5 that we can use to choose a graphic. Luckily for us, there is built-in functionality within JavaScript to do this. Let's get started on our first line.

THE MATH OBJECT

To achieve the first part of our script, we need to use the built-in `Math` object of JavaScript. This object contains methods and properties that supply mathematical functions and values. For our

purposes, we need the two methods `Math.random()` and `Math.round()`. First, we use the random method.

```
// First Generate a Random number that will be used to
// choose the initial product banner displayed on the page

var prodStart = Math.random()*4+1;
. . .
```

Let's take a close look at what is going on here. On the left-hand side of the assignment operator, we have the by now familiar creation of a variable, which will hold the value of whatever is on the right-hand side of the operator. In this case, it's a variable name `prodStart`, which will end up holding a random number between 1 and 5. How we get the random number takes a little more explaining. The random method of the `Math` object returns a random number between 0 and 1, so what you end up with is a decimal between 0 and 1, like so:

```
0.5489732579249
```

You're probably wondering how we get from there to a number between 1 and 5, huh? Have no fear: Enter some of that math that you were sure you would never use once you got out of school. First, we multiply the decimal returned by the random method by 4; once this is done, we will have a decimal number somewhere between 0.0 and 4.0. Finally, we add a value of 1 to the current number so that we end up with a decimal number between 1.0 and 5.0, such as one of these:

```
1.2489732579249
3.4589378128
4.8628879248931138
```

For our purposes, however, we need a nice whole number, so we need to use another of the `Math` object's methods in the next line of our script.

```
// First Generate a Random number that will be used to
// choose the initial product banner displayed on the page

var prodStart = Math.random()*4+1;
prodStart = Math.round(prodStart);
. . .
```

This line uses the round method of the `Math` object; it will round off any number that is placed inside of its parentheses. In

our case, we have inserted the `prodStart` variable, which carries the number we got from the random method on the line above. The round method then returns a whole number between 1 and 5, which will then be restored into the `prodStart` variable. We will use this variable later on in another part of our script, which actually prints out the initial product banner. First, however, we need to create the function that will be called to change the banner randomly every 20 seconds once the page is loaded. We call the function `bannerChanger()`. Let's add the framework for our function directly after the lines above.

```
// First Generate a Random number that will be used to
// choose the intitial product banner displayed on the page

var prodStart = Math.random()*4+1;
prodStart = Math.round(prodStart);

// Next Create the function which will change the images
// once the page is loaded

function bannerChanger ()  {
   . . .
}
```

The first thing we need to do when our function is called is to again generate a random number between 1 and 5, so this first portion of our function will be almost identical to the first two lines of code in our script, with only one small change.

```
// First Generate a Random number that will be used to
// choose the initial product banner displayed on the page

var prodStart = Math.random()*4+1;
prodStart = Math.round(prodStart);

// Next Create the function which will change the images
// once the page is loaded

function bannerChanger ()  {
     prodStart = Math.random()*4+1;
     prodStart = Math.round(prodStart);
     . . .
}
```

The only difference in these first two lines is that we took out the new variable declaration (`var`) from the first line. Because we have already created this variable, there is no reason to do so

again. Once these first two lines execute, when the function is called, we now have a new random number to use to display a new graphic. All we need to do now is tell the JavaScript engine which graphic to display.

```
// First Generate a Random number that will be used to
// choose the initial product banner displayed on the page

var prodStart = Math.random()*4+1;
prodStart = Math.round(prodStart);

// Next Create the function which will change the images
// once the page is loaded

function bannerChanger ()  {
  prodStart = Math.random()*4+1;
  prodStart = Math.round(prodStart);
  document.images['addBanner'].src = eval("product" +
prodStart + "Banner.src");
  . . .
}
```

What we are doing in this new line of code should look pretty familiar. It's the same technique we've used in the first two projects in this chapter. We reset the source property of the image named `prodBanner` to one of the other five product banners, using the random number we have generated. The final step in creating our function is to make sure that this function will be called again in another 20 seconds to change the product banner once more. To do this, we use a built-in timer method called `setTimeout()`. The syntax of the `setTimeout()` method is as follows.

```
setTimeout(expression, msec)
```

Once called, `setTimeout()` waits the specified amount of time and then executes the expression that is given. In our case, we want it to wait 20 seconds and then call our `bannerChanger()` function again, so let's insert the `setTimeout()` method into our script.

```
// First Generate a Random number that will be used to
// choose the initial product banner displayed on the page

var prodStart = Math.random()*4+1;
prodStart = Math.round(prodStart);

// Next Create the function which will change the images
// once the page is loaded
```

```
function bannerChanger ()  {
     prodStart = Math.random()*4+1;
     prodStart = Math.round(prodStart);
     document.images['addBanner'].src = eval("product" +
prodStart + "Banner.src");
     setTimeout("bannerChanger()", 20000);
}
. . .
```

We are now finished with our `bannerChanger()` function. There is just one last piece we have to add to this section of our script, then we will move down to the section of HTML where we want to print out our product banner graphic. We set it up so that once the `bannerChanger()` function has been called, it will keep re-calling itself every 20 seconds. However, we need to set it up so that the function gets called in the first place, so we need to add another `setTimeout()` method which will be executed upon the loading of the page. It will be identical to our first `setTimeout()`, located outside of the `bannerChanger()` function, like so.

```
// First Generate a Random number that will be used to
// choose the intitial product banner displayed on the page

var prodStart = Math.random()*4+1;
prodStart = Math.round(prodStart);

// Next Create the function which will change the images
// once the page is loaded

function bannerChanger ()  {
     prodStart = Math.random()*4+1;
     prodStart = Math.round(prodStart);
     document.images['prodBanner'].src = eval("product" +
prodStart + "Banner.src");
     setTimeout("bannerChanger()", 20000);
}

// Set timer to call the bannerChanger() function for the
// first time

setTimeout("bannerChanger()", 20000);
```

There. We're done with this section of our script; let's move to where we want to have the product banner printed out on our page.

Dynamically Printing Out the Image

Here is the section of HTML where the existing product banner is.

```
<table cellspacing="0" cellpadding="0" width="590">
      <tr>
           <td>
                <img src="images/secondary_top.gif"
alt="Shelley Biotech" width="333" height="68" border="0"><a
href="store/index.html"><img src="images/product_banners/
product_banner.gif" alt="" width="204" height="68"
border="0"></a><br>
```

As you can see, it is currently just calling `product_banner.gif`. We need to add a script that will dynamically print out whichever product banner the initial random number generation specifies. We use a technique similar to the second project we did in Chapter 1. First, we take out the existing image and insert our `<script>` tags and our browser cloaking.

```
<table cellspacing="0" cellpadding="0" width="590">
      <tr>
           <td>
                <img src="images/secondary_top.gif"
alt="Shelley Biotech" width="333" height="68"
border="0"><a href="store/index.html">
<script language="JavaScript1.2">
<!-- cloak from non JavaScript capable browsers

. . .

// End cloaking from non JavaScript browsers -->
</script>
</a><br>
```

Next, we use the `document.write()` method to print out our random product banner graphic.

```
<table cellspacing="0" cellpadding="0" width="590">
     <tr>
          <td>
               <img src="images/secondary_top.gif"
alt="Shelley Biotech" width="333" height="68"
border="0"><a href="store/index.html">
<script language="JavaScript1.2">
<!-- cloak from non JavaScript capable browsers
```

```
document.write('<img src="images/product_banners/
product_banner_' + prodStart + '.gif" width="204"
height="68" alt="Buy Now" name="prodBanner">');

// End cloaking from non JavaScript browsers -->
</script>
</a><br>
```

Let's look at the `document.write()` line a little closer. We have it print out the first part of our image tag, specifying which directory to look in for our image as well as the first part of the image's name. Then we add to that the random number held in the `prodStart` variable, and finally finish the name of the graphic and the rest of its attributes, such as `width`, `height`, and `name`. So, if the variable `prodStart` is holding the value of 3, this is what will be printed out by the script:

```
<img src="images/product_banners/product_banner_3.gif"
width="204" height="68" alt="Buy Now" name="prodBanner">
```

That about wraps it up for this project. We now have a script that randomly prints out a product banner graphic and every 20 seconds refreshes it with another random product banner.

Reviewing the Script

We introduced several new concepts during the creation of the script, so let's take one more look at what we did to create this script. First, we created the `IMAGE` objects and the function.

```
// Creating Image Objects for Product Banners
if (document.images) {
product1Banner=new Image(200, 56);
product1Banner.src="images/product_banners/
product_banner_1.gif";

product2Banner=new Image(200, 56);
product2Banner.src="images/product_banners/
product_banner_2.gif";

product3Banner=new Image(200, 56);
product3Banner.src="images/product_banners/
product_banner_3.gif";

product4Banner=new Image(200, 56);
```

```
product4Banner.src="images/product_banners/
product_banner_4.gif";

product5Banner=new Image(200, 56);
product5Banner.src="images/product_banners/
product_banner_5.gif";
}
// Generate a Random number that will be used to choose the
// intitial product banner displayed on the page

var prodStart = Math.random()*4+1;
prodStart = Math.round(prodStart);

// Create the function which will change the images once
// the page is loaded

function bannerChanger ()  {
     prodStart = Math.random()*4+1;
     prodStart = Math.round(prodStart);
     document.images['prodBanner'].src = eval("product" +
prodStart + "Banner.src");
     setTimeout("bannerChanger()", 20000);
}

// Set timer to call the bannerChanger() function for the
// first time

setTimeout("bannerChanger()", 20000);
```

Here are the steps we took to create this portion of the script:

1. We created the five `IMAGE` objects that we needed for our product banners.
2. We created a variable name `prodStart` and assigned it a random number between 1.0 and 5.0.
3. We rounded off that number to a whole number between 1 and 5 and reassigned it to the variable `prodStart`. This variable holds the value of the graphic that will be displayed when the page is first loaded.
4. We created a function `called bannerChanger()`, which updates the product banner graphic every 20 seconds.
5. Inside of the function, we again generated a random number between 1 and 5 and assigned it to the `prodStart` variable.
6. We used that value to change the source property of the `prodBanner` image on the page.

7. We set a timer using the `setTimeout()` method to re-call the `bannerChanger()` function after 20 seconds.
8. Once finished with the `bannerChanger()` function, we added one more timer to call the `bannerChanger()` function when the page first loads.

Next we moved over to the spot in the HTML where we want the product banner to be located.

```
<table cellspacing="0" cellpadding="0" width="590">
     <tr>
          <td>
               <img src="images/secondary_top.gif"
alt="Shelley Biotech" width="333" height="68" border="0"><a
href="store/index.html">
<script language="JavaScript1.2">
<!-- cloak from non JavaScript capable browsers

document.write('<img src="images/product_banners/
product_banner_' + prodStart + '.gif" width="204"
height="68" alt="Buy Now" name="prodBanner">');

// End cloaking from non JavaScript browsers -->
</script>
</a><br>
```

1. We got rid of the existing product image `<img>` tag.
2. We inserted our script framework into the HTML and cloaked for non-JavaScript-capable browsers.
3. We used the `document.write()` method along with the value held in the `prodStart` variable to print out the random product banner graphic.

We have been introduced to several new aspects of JavaScript in this first script, including

- The `Math` object along with its random and round methods.
- The `setTimeout()` method.

RECAP

You will find that image rollovers are by far the most commonly used JavaScript implemented on Web sites today. Almost every site you go to these days makes use of them in one way or another, so a good knowledge of how they work and what their

limitations are can be a great asset to a programmer. Once you have the basics down, use that knowledge to experiment with different rollover techniques. You will be surprised at some of the creative and exciting rollovers you can devise.

ADVANCED PROJECTS

1. Use the `onClick()` event handler to create navigation images that change not only upon rollover but when the user clicks on the image as well.
2. Create a page on which a random image replaces an existing image when the category images are rolled over.