8 GOING BEYOND THE BASICS WITH SPRY AND AJAX

The Spry effects and widgets described in the previous chapter owe their existence to a fundamental shift that has taken place in the past few years in the way that JavaScript is used to generate dynamic effects in the browser. Traditionally, JavaScript has been used to tackle specific tasks. For example, if you wanted to change an image on rollover, you would write a script designed simply for that purpose or use an existing one (Dreamweaver automates the process for you with Insert ➤ Image Objects ➤ Rollover Image). This has the advantage of producing lightweight dedicated scripts. For example, Dreamweaver's image rollover script is fewer than 20 lines of code. However, improvements in browser capabilities and better support for the DOM (see Chapter 7) spurred developers to see how far they could push JavaScript. The Spry effects might look quite simple, but they all involve changing the state of the target element (its position, transparency, or color) over a specified period. The amount of scripting required for each effect is considerable. Yet each effect shares common tasks: the need to identify the target element, a timer to control the transition, ways of dynamically manipulating the element's style rules, and so on. Rather than reinvent the wheel for each new script, it became more efficient to develop a framework or library of common functions.

The sudden explosion of JavaScript frameworks in recent years is a mixed blessing for web developers. In one respect, using a framework reduces the amount of code the developer needs to write because most complex tasks are handled by the framework. On the other hand, it involves a considerable learning curve. Books about the most popular frameworks, Prototype, script.aculo.us, jQuery, and Mootools, run to hundreds of pages. Dreamweaver has tried to reduce the Spry learning curve by automating the insertion and configuration of a large number of widgets and effects. All the JavaScript coding is handled for you seamlessly behind the scenes (it might come as a surprise that `SpryEffects.js` contains nearly 2,500 lines of code).

If you don't want to get your hands dirty with JavaScript, you can skip this chapter. On the other hand, if you do, you might find yourself frustrated at not being able to use Spry to its full extent. Because Spry is a fully fledged JavaScript framework, it's capable of doing much more than you can achieve through the Property inspector or dialog boxes. Doing things such as opening a panel from a link or making the height of accordion panels expand and contract depending on the amount of content in them involves diving under the hood and hand-coding JavaScript. Spry code hints make this a relatively painless process.

In this chapter, you'll learn about the following:

- Passing additional arguments to Spry effects and widgets
- Creating an accordion with flexible-height panels
- Opening Spry panels from links
- Combining Spry effects
- Using the Spry selector to manipulate styles on the fly
- Saving bandwidth with minified Spry files
- Creating unobtrusive JavaScript with the JavaScript Extractor
- Using other JavaScript libraries with Dreamweaver CS4
- Installing Dreamweaver extensions
- Experimenting with jQuery and YUI Library web widgets

If you have worked previously with JavaScript, you should have little difficulty customizing Spry effects and widgets. However, for the benefit of readers taking their first steps with programming, the following section explains some of the basic concepts.

In spite of the similarity of names, JavaScript is wholly unrelated to Java. They are different programming languages, and "Java" should never *be used as an abbreviation for JavaScript.*

Programming terminology 101

Programming languages like JavaScript and PHP (which you'll use in the second half of this book) change the output displayed in a web page in response to events or user input. Since the developer has no way of knowing in advance how users will interact with a page, programming languages use a variety of mechanisms to produce the required output. The following are some of the most important.

A **variable** acts as a placeholder for an unknown or changeable value, which may come from user input, a database, the result of a calculation, and so on. Although this sounds abstract, we use variables all the time in everyday life. My name is David, and my editor's name is Ben. In this case, "name" plays the same role as a variable—the word "name" always remains the same, but the *value assigned* to it can change.

Functions can be regarded as the verbs of programming languages; they do things. Many functions are built into the language, but you can also build your own functions by combining a series of commands. In both JavaScript and PHP, function names are always followed by a pair of parentheses. Often, the parentheses contain variables, known as **parameters** or **arguments**. Passing a value as an argument tells the function to do something with it, such as perform a calculation or format text.

JavaScript is triggered by **events**, such as when the page has finished loading or the user clicks a link. You tell the browser to run a function by assigning it (plus any arguments, if necessary) to an **event handler** such as `onclick`, `onmouseover`, or `onmouseout`. To give a trivial example, the following code pops up an annoying message when the link is clicked:

```
<a href="#" onclick="alert('You clicked!')">Click me quick</a>
```

A **string** is the name that programming languages give to text. A string is always enclosed in quotes (single or double). By contrast, numbers should not normally be enclosed in quotes, unless they're part of a string.

An **array** is a variable that can hold multiple values, rather like a shopping list.

An **object** is like a super variable, which can have variables (called **properties**) and functions (called **methods**) of its own. New instances of an object are created using the `new` keyword followed by a constructor function, which looks and works very much like any other function.

Both JavaScript and PHP are case-sensitive. You must use the right combination of uppercase and lowercase when typing JavaScript and PHP code. Dreamweaver code hints are invaluable in helping get the spelling right.

Understanding Spry objects

In common with other JavaScript frameworks, Spry uses JavaScript objects. The idea of using objects is that all the complex coding remains locked away in the object definition, so you need concern yourself only with parts exposed through the object's methods and properties. Methods are functions that can be used to get the object to perform particular actions. Properties define the state of an object. All Spry effects and widgets are objects. So, for example, the properties of an accordion determine whether a panel is open or whether the panels have a fixed height; and to open the panel of a tabbed panels object, you use its `showPanel()` method.

The object definitions aren't literally locked away. You can study them by opening the external JavaScript files that Dreamweaver copies to the Spry assets folder. However, you should never edit the JavaScript in those files unless you really know what you're doing. And if you do know what you're doing, you would probably create your own methods and properties without touching the original files.

Initializing a Spry object

When you insert a Spry tabbed panels widget, Dreamweaver initializes the JavaScript object at the bottom of the page just before the closing `</body>` tag like this:

```
<script type="text/javascript">
<!--
var TabbedPanels1 = new Spry.Widget.TabbedPanels("TabbedPanels1");
//-->
</script>
</body>
```

The line of JavaScript highlighted in bold creates a new tabbed panels object and stores a reference to it in a JavaScript variable with the same name as the ID of the `<div>` that contains the panels. The ID and the JavaScript variable don't need to be the same, but Dreamweaver adopts this convention to make it easy to use Spry properties and methods.

Dreamweaver normally handles all the coding for you, but if you want to get more adventurous with Spry widgets, you need to understand what the code means. So, let's analyze it piece by piece:

- var: This is a JavaScript keyword used to declare a new variable. Variable names in JavaScript cannot begin with a number and should not contain any spaces or punctuation, except for the underscore (_).
- TabbedPanels1: This is the name of the new variable, which can be used elsewhere in the script to represent whatever value is assigned to it.
- The assignment operator (=): This assigns the value on the right to the variable on the left. Try not to think of it as an equal sign, because both JavaScript and PHP use *two* equal signs to indicate equality.
- new: This is a JavaScript keyword that creates an instance of an object.
- Spry.Widget: This is the object of which a new instance is being created.
- TabbedPanels("TabbedPanels1"): This is the constructor method of the object. In this case, it creates a tabbed panels widget. The value in quotes between the parentheses is an argument being passed to the new object. Arguments set the values of specific properties. In the case of Spry objects, the first argument is always the ID of the target element.

If you change the value of the Default panel in the Property inspector to Tab2, Dreamweaver changes the initialization code like this:

```
var TabbedPanels1 = new Spry.Widget.TabbedPanels("TabbedPanels1", ➥
{defaultTab:1});
```

The format of the second argument is important. Unlike the first argument, it's not enclosed in quotes but in a pair of curly braces. In JavaScript, this is called an **object literal**. An object literal is simply a shorthand way of creating a new object. It consists of name/value pairs surrounded by curly braces. Each name/value pair defines a property, with a colon separating the value from the property name. This object literal contains a single name/value pair: defaultTab, which is a property of a tabbed panels widget, and 1, which is the value assigned to that property. No, it's not a mistake. Like most programming languages, JavaScript counts from zero, so the number of the second tab is 1, not 2.

The second argument in most Spry constructor methods sets various options. Since an object literal can accept multiple name/value pairs as a comma-separated list, using an object literal as the second argument makes it easy to pass multiple options to the Spry effect or widget like this:

```
var TabbedPanels1 = new Spry.Widget.TabbedPanels("TabbedPanels1", ➥
{property1:value1, property2:value2, property3:value3});
```

You can put whitespace around the colons and insert new lines after the commas for ease of reading. Don't worry if all this terminology sounds intimidating. As you'll see in the following exercises, hand-coding Spry is relatively painless.

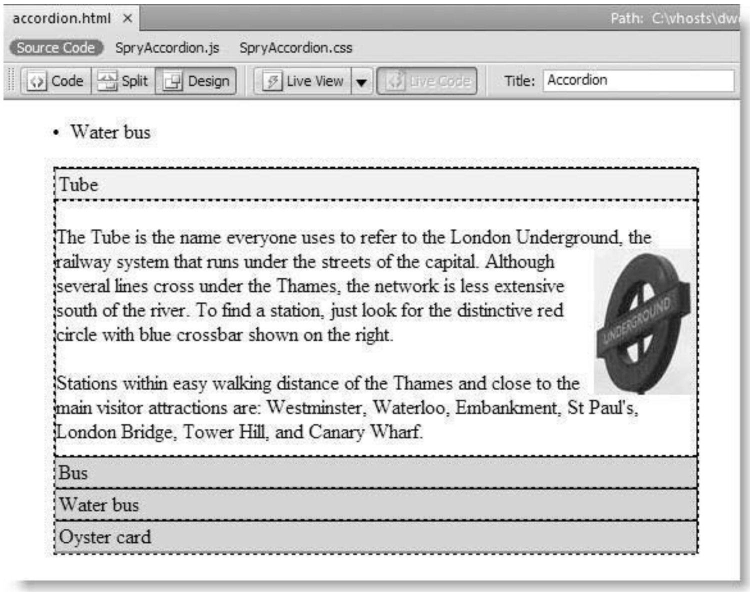
Changing accordion defaults

As explained in Chapter 7, the Property inspector for an accordion lets you change only the ID and the number and order of panels. Unlike Spry tabbed panels, there's no option to select a panel to be displayed by default when the page first loads. What's more, changing the default behavior of using fixed-height panels isn't just a question of tweaking the style sheet. To make both changes, you need to pass options to the accordion object constructor.

Changing an accordion's default open panel

By default, an accordion is always displayed with the first panel open. This exercise shows how to display a different panel when the page first loads. This technique always displays the same panel. It cannot be used to open a specific panel from a link in a different page.

1. Create a new folder called `ch08` in your `workfiles` folder, copy `accordion_start.html` from examples/ch08, and save it in the new folder as `accordion.html`. Update the links when prompted. The page should look like this in Design view:



- Water bus

Tube

The Tube is the name everyone uses to refer to the London Underground, the railway system that runs under the streets of the capital. Although several lines cross under the Thames, the network is less extensive south of the river. To find a station, just look for the distinctive red circle with blue crossbar shown on the right.

Stations within easy walking distance of the Thames and close to the main visitor attractions are: Westminster, Waterloo, Embankment, St Paul's, London Bridge, Tower Hill, and Canary Wharf.

Bus

Water bus

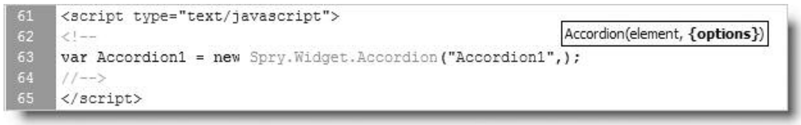
Oyster card

The accordion contains the same material as used in Chapter 7. However, I have left the accordion unstyled apart from constraining its width.

2. To change the default open panel, open the page in Code view, and scroll down to the bottom. Locate the following line of code, which initializes the accordion object:

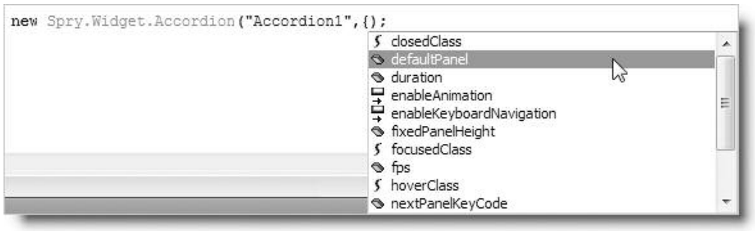
```
var Accordion1 = new Spry.Widget.Accordion("Accordion1");
```

3. Insert your cursor just before the closing parenthesis, and type a comma. Dreamweaver displays the following code hint:


```
<script type="text/javascript">
<!--
var Accordion1 = new Spry.Widget.Accordion("Accordion1",);
//-->
</script>
```

This tells you that Spry expects the constructor method to take two arguments: `element` (the ID of the `<div>` that houses the accordion) and `options`. Because `options` is highlighted in bold, that's what Dreamweaver now expects you to enter. The curly braces remind you that `options` must be a JavaScript object literal.

4. Type an opening curly brace. This pops up a second code hint, as shown here:



```
new Spry.Widget.Accordion("Accordion1",{);
```

This shows you some of the available options. Double-click defaultPanel, or use the down arrow key to select it and press Enter/Return. Dreamweaver inserts the `defaultPanel` property followed by a colon ready for you to insert the value. JavaScript numbers the panels from 0, so to open the third panel, type 2 followed by a closing curly brace. The code should now look like this:

```
var Accordion1 = new Spry.Widget.Accordion("Accordion1", ➥
{defaultPanel:2});
```

5. Save `accordion.html`, and load it in a browser (or activate Live view). The third panel (Water bus) should open instead of the first one.

   Check your code, if necessary, with `accordion_default.html` in `examples/ch08`.

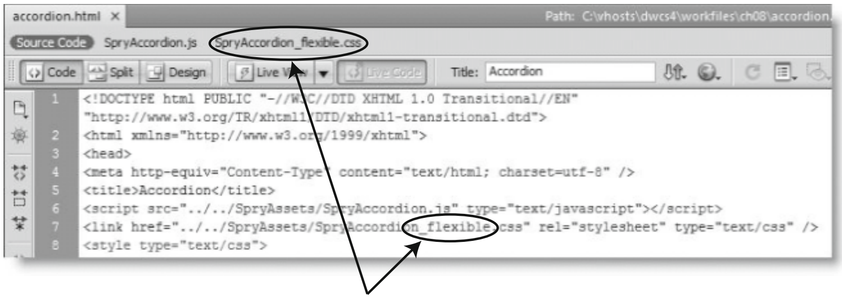
Converting an accordion to flexible height

Using a fixed height for an accordion is very useful when you need to keep different parts of a page in alignment, but the scrollbars tend to look unsightly (only Internet Explorer for Windows supports the nonstandard CSS properties for styling scrollbars).

Converting an accordion to flexible height involves two stages: editing the CSS and passing an option to the accordion object's constructor method. Continue using `accordion.html` from the preceding exercise.

1. With `accordion.html` open in the Document window, select SpryAccordion.css in the Related Files toolbar. Then select File ➤ Save As, and save the style sheet as `SpryAccordion_flexible.css`. This opens the style sheet in a new tab. Close the new tab straightaway, because you'll work with it as a related file.
2. Although you have saved the style sheet with a different name, the original style sheet is still attached to `accordion.html`. The quickest way to attach the new style sheet is to select Source Code in the Related Files toolbar to reveal the HTML code of `accordion.html`. Change SpryAccordion.css to SpryAccordion_flexible.css in the

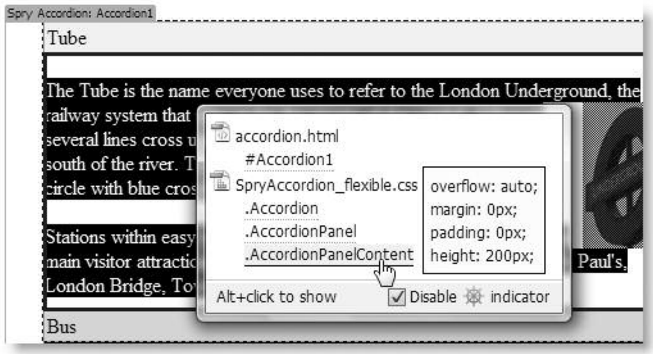
`<link>` tag in the `<head>` of the document, save the page, and press F5 to update the Related Files toolbar, as shown here:



```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Accordion</title>
<script src="../../SpryAssets/SpryAccordion.js" type="text/javascript"></script>
<link href="../../SpryAssets/SpryAccordion_flexible.css" rel="stylesheet" type="text/css" />
<style type="text/css">
```

Change the name of the style sheet in the code.
Then press F5 to update the Related Files toolbar.

3. You need to change the properties in the `.AccordionPanelContent` selector of the style sheet. There are several ways you can do it, but a quick way to find the right section of code to edit is to switch to Design view, hold down the Alt key (or Opt+Cmd on a Mac), and click anywhere inside the accordion. Click the link for the `.AccordionPanelContent` selector in the Code Navigator, as shown in the following screenshot:



The Tube is the name everyone uses to refer to the London Underground, the
railway system that
several lines cross u
south of the river. T
circle with blue cros
Stations within easy
main visitor attractio
London Bridge, To

```
overflow: auto;
margin: 0px;
padding: 0px;
height: 200px;
```

4. Change the value of `overflow` from `auto` to `hidden`. If you leave the `overflow` property set to `auto`, some longer panels still spawn a scrollbar. You need to set it to `hidden` so that only the currently open panel is visible. Also delete the `height` property from the rule, which should now look like this:

```
.AccordionPanelContent {
  overflow: hidden;
  margin: 0px;
  padding: 0px;
}
```

That takes care of the CSS. Now you need to tell the `Accordion` object to use flexible height.

5. Select Source Code in the Related Files toolbar to return to the HTML code for `accordion.html`, and then scroll right to the bottom of the page and locate the code that initializes the `Accordion` object (see step 2 in the preceding exercise).
6. If you changed the default open panel in the preceding exercise, amend the constructor function like this (new code is in bold):

```
var Accordion1 = new Spry.Widget.Accordion("Accordion1", ➥
{defaultPanel:2, useFixedPanelHeights:false});
```

If you just want to remove the fixed panel heights, amend the code like this:

```
var Accordion1 = new Spry.Widget.Accordion("Accordion1", ➥
{useFixedPanelHeights:false});
```

Make sure you don't omit the comma after `"Accordion1"`.

7. Select File ➤ Save All to save the changes to both `accordion.html` and the style sheet, and test the page in your browser. You now have a flexible-height accordion and no ugly scrollbars.

Check your code, if necessary, with `accordion_flexible.html` in `examples/ch08`. The style sheet, `SpryAccordion_flexible.css`, is in the `SpryAssets` folder.

Using an object's methods

Once you have created an object, you can use its methods. You do this by adding a period to the end of the variable that contains the object, followed by the method name and any arguments. So, to open the second panel of a tabbed panels widget stored in `TabbedPanels1`, you use its `showPanel()` method like this:

```
TabbedPanels1.showPanel(1)
```

Opening panels from a link on the same page

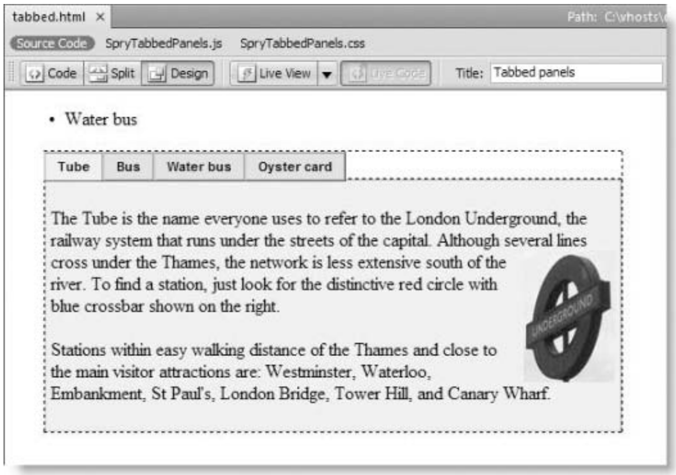
The technique for opening a specific panel differs not only for each type of Spry widget but also depending on whether the link is located in the same page. This section contains instructions for opening a panel from links within the same page as the widget. There are separate instructions for tabbed panels, accordions, and collapsible panels.

Opening a tabbed panel from a link on the same page

This exercise shows you how to open a specific panel in a tabbed panels widget from a link in the same page.

1. Copy `tabbed_start.html` from `examples/ch08` to `workfiles/ch08`, and save the file as `tabbed.html`. Update the links when Dreamweaver prompts you to do so.

The page contains a tabbed panels widget, as shown in the following screenshot:



• Water bus

Tube	Bus	Water bus	Oyster card

The Tube is the name everyone uses to refer to the London Underground, the railway system that runs under the streets of the capital. Although several lines cross under the Thames, the network is less extensive south of the river. To find a station, just look for the distinctive red circle with blue crossbar shown on the right.

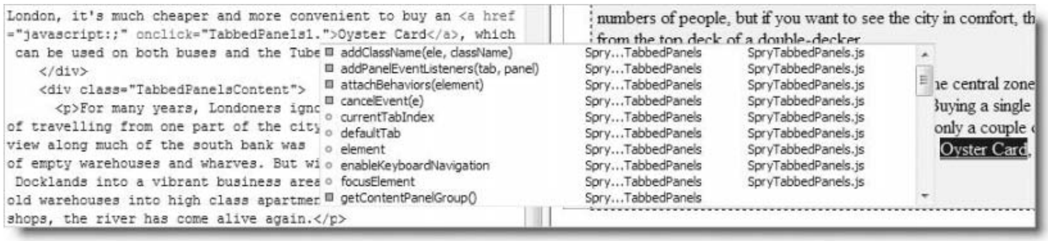
Stations within easy walking distance of the Thames and close to the main visitor attractions are: Westminster, Waterloo, Embankment, St Paul's, London Bridge, Tower Hill, and Canary Wharf.

As with the accordion in the previous exercises, the panels are unstyled apart from a rule that constrains their width.

2. In Design view, select the tab named Bus in the Property inspector, or click its eye icon to reveal the panel content.
3. Select the words Oyster Card in the final sentence, and type javascript:; in the Link field of the HTML view of the Property inspector to create a dummy link.
4. With the words Oyster Card still highlighted, open Split view to reveal the underlying code, and position your cursor just before the closing angle bracket of the `<a>` tag.
5. Press the spacebar. Code hints should pop up. Type onc, and press Enter/Return when onclick is highlighted. The link surrounding Oyster Card should now look like this (with the cursor between the quotes following `onclick`):

   ```
   <a href="javascript:;" onclick="">Oyster Card</a>
   ```

6. To call one of the Spry methods (functions) on a widget, type the ID of the widget followed by a period and the name of the method. The ID of this widget is `TabbedPanels1`. As soon as you type the period after the ID, Dreamweaver pops up code hints for the selected widget, showing the available methods (see Figure 8-1).



```
London, it's much cheaper and more convenient to buy an <a href
="javascript:;" onclick="TabbedPanels1.">Oyster Card</a>, which
 can be used on both buses and the Tube
    </div>
    <div class="TabbedPanelsContent">
      <p>For many years, Londoners ignc
of travelling from one part of the city
view along much of the south bank was
of empty warehouses and wharves. But wi
 Docklands into a vibrant business area
old warehouses into high class apartmer
shops, the river has come alive again.</p>
```

```
addClassName(ele, className)           Spry...TabbedPanels    SpryTabbedPanels.js
addPanelEventListeners(tab, panel)     Spry...TabbedPanels    SpryTabbedPanels.js
attachBehaviors(element)               Spry...TabbedPanels    SpryTabbedPanels.js
cancelEvent(e)                         Spry...TabbedPanels    SpryTabbedPanels.js
currentTabIndex                        Spry...TabbedPanels    SpryTabbedPanels.js
defaultTab                             Spry...TabbedPanels    SpryTabbedPanels.js
element                                Spry...TabbedPanels    SpryTabbedPanels.js
enableKeyboardNavigation               Spry...TabbedPanels    SpryTabbedPanels.js
focusElement                           Spry...TabbedPanels    SpryTabbedPanels.js
getContentPanelGroup()                 Spry...TabbedPanels
```

numbers of people, but if you want to see the city in comfort, th
from the top deck of a double-decker
ne central zone
Buying a single
only a couple
Oyster Card,

Figure 8-1. Code hints recognize Spry widgets and display available methods.

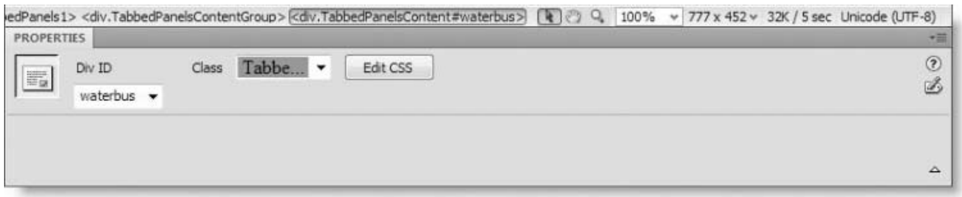
Use your mouse or keyboard arrow keys to select showPanel(elementOrIndex), and double-click or press Enter/Return. This inserts showPanel followed by an opening parenthesis. Type 3 followed by a closing parenthesis.

Following JavaScript convention, Spry counts the panels from 0, so 3 represents the fourth panel (Oyster Card). The Oyster Card link code should now look like this:

```
<a href="javascript:;" onclick="TabbedPanels1.showPanel(3)">Oyster ➥
Card</a>
```

7. Activate Live view. Select the Bus tab, and click the Oyster Card link within the displayed panel. The fourth panel should open.
8. The link to open another panel doesn't need to be inside the widget; it can be anywhere in the page. You can also identify the panel you want to open with an ID rather than counting its number from zero. This is particularly useful if the order of the panels is likely to change.

   Switch off Live view, and select the Water bus tab. With your cursor anywhere inside the content of the third panel, select <div.TabbedPanelsContent> in the Tag selector at the bottom of the Document window. This selects the `<div>` that contains the third panel.
9. Enter waterbus in the Div ID field of the Property inspector, and press Tab or Enter/Return. The ID of the `<div>` should be added to the selected tag in the Tag selector, as shown here:


edPanels1> <div.TabbedPanelsContentGroup> <div.TabbedPanelsContent#waterbus>
100%
777 x 452
32K / 5 sec Unicode (UTF-8)
PROPERTIES
Div ID
waterbus
Class
Tabbe...
Edit CSS

10. You can now use this to open the panel from a link. Select the text in the bullet point at the top of the page, and create a dummy link by entering javascript:; in the Link field of the HTML view of the Property inspector.
11. Open Split view, and insert an onclick event handler inside the link as you did in steps 5 and 6. However, this time, use the ID of the panel. The link should look like this:

```
<a href="javascript:;" onclick="TabbedPanels1.showPanel('waterbus')"> ➥
Water bus</a>
```

Note that the ID of the panel must be in single quotes. Do *not* use double quotes. In programming languages, quotes must always be in matching pairs. The onclick attribute uses double quotes, so any quotes used inside must be single. Otherwise, the code won't work.

Understanding the use of quotes is vital when working with languages like JavaScript and PHP. In many circumstances, it doesn't matter whether you use single or double quotes, as long as they're a matching pair. For example, `onclick` *could use single quotes, but in that case, the ID nested inside would need to use double quotes. When a programming language sees an opening quote, it grabs the next matching one as the closing quote. So, you always need to make sure you don't accidentally end a command prematurely by using the wrong type of quotation mark.*

12. Activate Live view, and click the Water bus link. The Water bus panel should open.
13. Turn off Live view, select the tabbed panels widget by clicking the turquoise tab at the top left, and use the up and down arrows in the Property inspector to move the Water bus and Oyster card panels to different positions.
14. Test the page in Live view again. The Water bus panel should still open correctly. However, the link that you created in the Bus panel will no longer open the Oyster card panel. Instead, it opens whatever has been moved to the fourth position.

    Check your code, if necessary, against `tabbed_link.html` in `examples/ch08`.

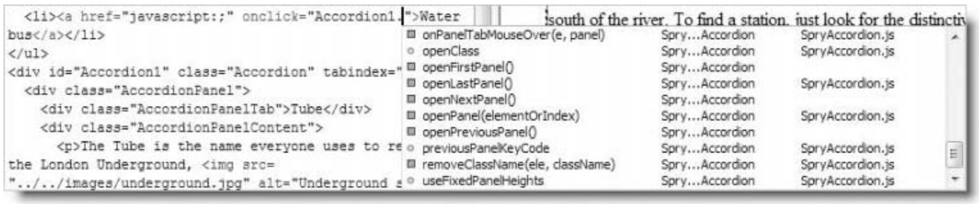
Opening an accordion panel from a link on the same page

Unlike a tabbed panels widget, an accordion doesn't have a `showPanel()` method. However, the process is very similar. Continue working with `accordion.html` from the exercises earlier in the chapter. Alternatively, copy `accordion_flexible.html` from `examples/ch08`, and save it as `accordion.html` in `workfiles/ch08`. Update the links when Dreamweaver prompts you to do so.

1. If you did the exercises with the accordion earlier in this chapter, remove the `defaultPanel` argument from the options used to initialize the accordion constructor. Open Code view, and make sure the code at the bottom of the page looks like this:

    ```
    var Accordion1 = new Spry.Widget.Accordion("Accordion1", ➥
     {useFixedPanelHeights:false});
    ```

2. Back in Design view, highlight the text in the bullet point at the top of `accordion.html`, and type javascript:; into the Link field of the HTML view of the Property inspector to create a dummy link. Open Split view, and add an `onclick` attribute to the `<a>` tag in the same way as in step 5 in the preceding exercise.
3. With your cursor between the quotes of the `onclick` attribute, type Accordion1 followed by a period. As soon as you type the period, Dreamweaver pops up code hints of the available methods. Scroll down to the bottom of the list, as shown in the following screenshot:



```
  <li><a href="javascript:;" onclick="Accordion1.">Water
bus</a></li>
</ul>
<div id="Accordion1" class="Accordion" tabindex="
  <div class="AccordionPanel">
    <div class="AccordionPanelTab">Tube</div>
    <div class="AccordionPanelContent">
      <p>The Tube is the name everyone uses to re
the London Underground, <img src=
"../../images/underground.jpg" alt="Underground s
```

south of the river. To find a station, just look for the distinctiv

```
onPanelTabMouseOver(e, panel)     Spry...Accordion     SpryAccordion.js
openClass                         Spry...Accordion     SpryAccordion.js
openFirstPanel()                  Spry...Accordion
openLastPanel()                   Spry...Accordion     SpryAccordion.js
openNextPanel()                   Spry...Accordion
openPanel(elementOrIndex)         Spry...Accordion     SpryAccordion.js
openPreviousPanel()               Spry...Accordion
previousPanelKeyCode              Spry...Accordion     SpryAccordion.js
removeClassName(ele, className)   Spry...Accordion     SpryAccordion.js
useFixedPanelHeights              Spry...Accordion     SpryAccordion.js
```

Note that showPanel() is not listed, but there are four methods that target specific panels: openFirstPanel(), openLastPanel(), openNextPanel(), and openPreviousPanel(). Because they target specific panels, you don't need to add anything between the parentheses. However, Water bus is the third panel, so none of these will work. Select openPanel(elementOrIndex).

4. Since Water bus is the third panel, JavaScript counts its position (or index) as 2. So, add 2, and close the parentheses. The link should look like this:

```
<a href="javascript:;" onclick="Accordion1.openPanel(2)">Water bus</a>
```

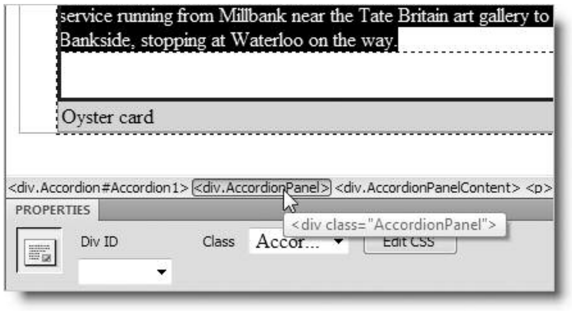
5. Save accordion.html, and load the page into a browser. Click the link at the top of the page. The Water bus panel slides open. This is a considerable improvement over the version of Spry in Dreamweaver CS3, which forced you to go through two steps to open a panel using its index.

6. As you saw in the previous exercise, using a number to identify a panel is risky because you need to recode everything if the panel's position changes. Giving the panel an ID and passing it as an argument to the openPanel() method is more reliable. However, you need to make sure you apply the ID to the correct element.

   Open the Water bus panel in Design view. Select all or part of it to make it easy to identify in Split view. Notice that the tab and the panel content are each in a separate <div> nested inside another <div> that holds tab and content together like this:

```
<div class="AccordionPanel">
  <div class="AccordionPanelTab">Water bus</div>
  <div class="AccordionPanelContent">
   <p>For many years, Londoners . . .</p>
  </div>
  </div>
</div>
```

7. The ID *must* be applied to the outer <div>. Applying it to the <div> that contains the tab or panel content won't work. To make sure you get the correct <div>, either work in Code view or click inside the content of the panel in Design view and select <div.AccordionPanel> from the Tag selector, as shown in the following screenshot:


service running from Millbank near the Tate Britain art gallery to Bankside, stopping at Waterloo on the way.
Oyster card
<div.Accordion#Accordion1> <div.AccordionPanel> <div.AccordionPanelContent> <p>
PROPERTIES
<div class="AccordionPanel">
Div ID
Class
Accor...
Edit CSS

   After selecting the correct tag, enter the ID in the Div ID field of the Property inspector. For this exercise, enter waterbus.

8. Amend the argument passed to the openPanel() method in step 3 like this (using single quotes around the ID):

```
<a href="javascript:;" onclick="Accordion1.openPanel('waterbus')"> ➥
Water bus</a>
```

9. Select the turquoise tab at the top left of the accordion widget to open its details in the Property inspector, and use the up or down arrow to move the Water bus panel to a different position.
10. Save the page, and test it in a browser. When you click the Water bus link, the correct panel should still open.

    You can check your code, if necessary, against accordion_link.html in examples/ch08.

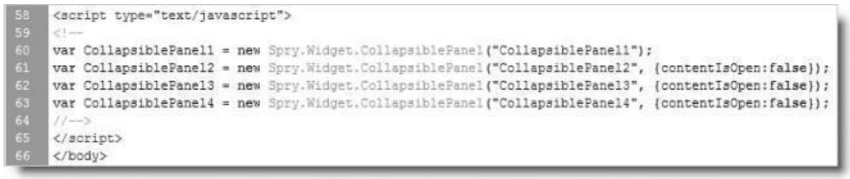
Opening a collapsible panel from a link in the same page

Since collapsible panels work independently, opening one from a link is simply a matter of applying the open() method to the JavaScript variable that identifies the target panel. By default, Dreamweaver names the first panel on a page CollapsiblePanel1 and increments the number by one for each subsequent panel.

This exercise shows how to open collapsible panels from a link on the same page.

1. Copy collapsible_start.html from examples/ch08, and save it in workfiles/ch08 as collapsible.html. Update the links when Dreamweaver prompts you to do so.

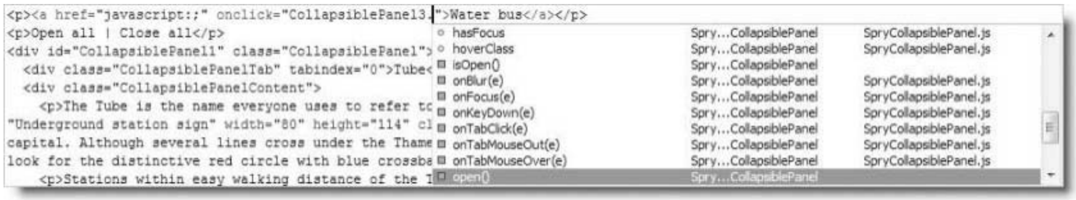
   The page contains four collapsible panels with the same content as before. The first panel is set to display open, while the others remain closed. Again, the only styling on the page limits the width of the panels.
2. Select Water bus at the top of the page, and create a dummy link as you have done in all previous exercises.
3. Switch to Code view, and scroll to the bottom of the page. The code that initializes the collapsible panel objects looks like this:



```
<script type="text/javascript">
<!--
var CollapsiblePanel1 = new Spry.Widget.CollapsiblePanel("CollapsiblePanel1");
var CollapsiblePanel2 = new Spry.Widget.CollapsiblePanel("CollapsiblePanel2", {contentIsOpen:false});
var CollapsiblePanel3 = new Spry.Widget.CollapsiblePanel("CollapsiblePanel3", {contentIsOpen:false});
var CollapsiblePanel4 = new Spry.Widget.CollapsiblePanel("CollapsiblePanel4", {contentIsOpen:false});
//-->
</script>
</body>
```

   As you can see, there are four separate objects. The first argument passed to the constructor method of a Spry object is always the ID of the target element, so CollapsiblePanel3 is a unique identifier for the Water bus panel. Even if you move the panels about on the page, each one retains its original ID.
4. Scroll back up to the dummy link, and add an onclick attribute to the <a> tag.
5. With your cursor between the quotes of the onclick attribute, type CollapsiblePanel3 followed by a period. As soon as you type the period, Dreamweaver pops

up code hints of the available methods. Scroll down until you locate open(), as shown here:



```
<p><a href="javascript:;" onclick="CollapsiblePanel3.">Water bus</a></p>
<p>Open all | Close all</p>
<div id="CollapsiblePanel1" class="CollapsiblePanel">
  <div class="CollapsiblePanelTab" tabindex="0">Tube<
  <div class="CollapsiblePanelContent">
    <p>The Tube is the name everyone uses to refer tc
"Underground station sign" width="80" height="114" cl
capital. Although several lines cross under the Thame
look for the distinctive red circle with blue crossba
    <p>Stations within easy walking distance of the T
```

```
hasFocus            Spry...CollapsiblePanel   SpryCollapsiblePanel.js
hoverClass          Spry...CollapsiblePanel   SpryCollapsiblePanel.js
isOpen()            Spry...CollapsiblePanel
onBlur(e)           Spry...CollapsiblePanel   SpryCollapsiblePanel.js
onFocus(e)          Spry...CollapsiblePanel   SpryCollapsiblePanel.js
onKeyDown(e)        Spry...CollapsiblePanel   SpryCollapsiblePanel.js
onTabClick(e)       Spry...CollapsiblePanel   SpryCollapsiblePanel.js
onTabMouseOut(e)    Spry...CollapsiblePanel   SpryCollapsiblePanel.js
onTabMouseOver(e)   Spry...CollapsiblePanel   SpryCollapsiblePanel.js
open()              Spry...CollapsiblePanel
```

6. Double-click open(), or press Enter/Return. That's it.
7. Save the page, and test it in a browser. When you click the Water bus link at the top of the page, the Water bus panel opens. Because collapsible panels are independent of each other, this has no effect on any other panels that are open.

   You can check your code with `collapsible_link.html` in `examples/ch08`.

Although it's useful to open a collapsible panel from a link, wouldn't it be nice to be able to close it as well? As you scrolled down the list of code hints in step 5 of the previous exercise, you probably noticed that there's a `close()` method, too. Although you can use that with a different link, how about toggling a panel open and closed from the same link?

Toggling a collapsible panel open and closed from a remote link

This next exercise shows you how to build a custom function to toggle any collapsible panel open and closed from a link on the same page. Continue working with `collapsible.html` from the preceding exercise.

The instructions in this exercise are deliberately verbose to help readers who are new to JavaScript. If you already have experience writing your own JavaScript, you might prefer to skim over most of the explanations and study the finished (very simple) script in `collapsible_toggle.html`.

1. As you have already learned, a collapsible panel object has both an `open()` method and a `close()` method. To toggle a panel open and closed, you need a way of finding out its current state. Take a closer look at the screenshot in step 5 of the preceding exercise. Among the code hints is another method called `isOpen()` (it's the third one down in the screenshot). There isn't an equivalent method that tells you whether a panel is closed, but that's not important. If a panel's not open, it must be closed.
2. Open Code view, and scroll up to the closing `</head>` tag (it should be around line 24). Create some space before the closing `</head>` tag, and insert a `<script>` block like this (the new code is shown in bold):

```
</style>
<script type="text/javascript">
</script>
</head>
```

3. To create a custom function, you type the keyword `function` followed by the name you want to use for the function. The name is followed by a pair of parentheses. The body of the function goes between a pair of curly braces. So, amend the code like this:

```
<script type="text/javascript">
function togglePanel()
{
}
</script>
```

4. Since we have been working with the Water bus panel (`CollapsiblePanel3`), let's continue doing so. Decisions in programming languages are made by determining whether a condition is true or false. The `isOpen()` method produces a **Boolean value** (`true` or `false`). So, `CollapsiblePanel3.isOpen()` will equate to `true` if it's open. Otherwise, it equates to `false`. In programming terms, a function or method is said to **return** a value. So, what we're interested in is whether it returns `true` or `false`.

Conditional decisions are handled by using the keyword `if` followed by the condition in parentheses. Any code you want to run only if the condition is true goes inside a pair of curly braces.

If the panel is open, you want to close it, but if it's closed, you want to open it. To run different code when a condition is false, you use the `else` keyword and put the code in another pair of curly braces.

Put everything together, and it looks like this:

```
<script type="text/javascript">
function togglePanel()
{
  if (CollapsiblePanel3.isOpen()) {
    CollapsiblePanel3.close();
  } else {
    CollapsiblePanel3.open();
  }
}
</script>
```

5. To use this function, you now need to change the code in the dummy link. It currently looks like this:

```
<a href="javascript:;" onclick="CollapsiblePanel3.open()">Water bus</a>
```

Change it to this:

```
<a href="javascript:;" onclick="togglePanel()">Water bus</a>
```

6. Save `collapsible.html`, and test the page in a browser. When you click the Water bus link, the panel should now open or close depending on its current state.

Check your code, if necessary, with collapsible_toggle_waterbus.html in examples/ch08. JavaScript is intolerant of mistakes, so use the File Compare feature, as described in Chapter 2, if you're having problems. A missing period, quotation mark, parenthesis, or curly brace will prevent the function from working.

7. This works fine, but it's very inflexible, because it works only with CollapsiblePanel3. This is where passing an argument to a function makes it far more useful. The argument is a variable that goes between the parentheses at the end of the function name. You then use that variable inside the function to represent the actual value that's passed when the function is used. We're toggling the open and closed states of a panel, so let's call the variable panel.

   Change the function like this (the changes are in bold):

```
function togglePanel(panel)
{
  if (panel.isOpen()) {
    panel.close();
  } else {
    panel.open();
  }
}
```

8. Finally, you need to pass the ID of the panel you want to open as an argument to togglePanel() like this:

```
<a href="javascript:;" onclick="togglePanel(CollapsiblePanel3)">Water ➥
bus</a>
```

   Note that the ID is *not* in quotes because you're passing the object, and not a string.

9. Save collapsible.html, and test the page in a browser again. It should toggle the Water bus panel open and closed as before.

10. Now, the *real* test. Copy and paste the Water bus link, and change it like this:

```
<p><a href="javascript:;" onclick="togglePanel(CollapsiblePanel3)"> ➥
Water bus</a></p>
<p><a href="javascript:;" onclick="togglePanel(CollapsiblePanel4)"> ➥
Oyster card</a></p>
```

11. Save the page, and test the new link, which should toggle the Oyster card panel open and closed.

   Check your code, if necessary, against collapsible_toggle.html in examples/ch08.

That solves the problem of toggling a single panel open and closed. How about opening and closing all panels with a single click? Actually, this feature is already built into the external JavaScript file that controls collapsible panels, but you need to implement it manually. It's very easy, as the next exercise shows.

Opening and closing all collapsible panels simultaneously

This exercise shows you how to group collapsible panels so they can be opened or closed as a single unit. Each panel, however, can be opened or closed independently.

1. Continue working with the file from the preceding exercise. Alternatively, copy `collapsible_toggle.html` from `examples/ch08`, and save it as `collapsible.html` in `workfiles/ch08`.
2. To open and close all panels simultaneously, you need to wrap them in an outer `<div>`. Selecting multiple elements in Design view can be tricky, so the safest way to do this is in Code view. Insert a new `<div>` tag just before the first collapsible panel. It needs both an ID and a class. The ID can be anything you like, as long as it's unique on the page (I used `panelgroup`). The class must be `CollapsiblePanelGroup`. The amended code looks like this (it should be around line 40):

```
<p>Open all | Close all</p>
<div id="panelgroup" class="CollapsiblePanelGroup">
<div id="CollapsiblePanel1" class="CollapsiblePanel">
```

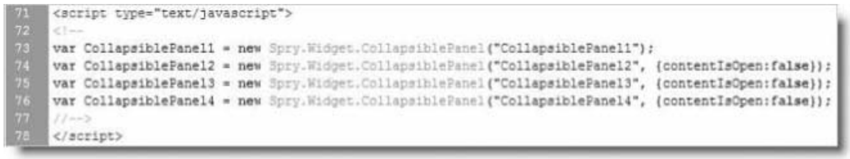
> *You can combine* `CollapsiblePanelGroup` *with other classes in the same* `class` *attribute, but you need to do this in Code view or the* Tag Inspector *panel, because Dreamweaver doesn't support assigning multiple classes through the Property inspector.*

3. Scroll to the end of the last panel, and insert a closing `</div>` tag. It should go immediately above the `<script>` block around line 70, like this:

```
more than two years).</p>
  </div>
</div>
</div>
<script type="text/javascript">
```

4. When you create a collapsible panel group like this, it's no longer necessary to initialize each panel individually. You just need to create an instance of the `CollapsiblePanelGroup` object.

   The `<script>` block at the bottom of the page currently looks like this:



```
71 <script type="text/javascript">
72 <!--
73 var CollapsiblePanel1 = new Spry.Widget.CollapsiblePanel("CollapsiblePanel1");
74 var CollapsiblePanel2 = new Spry.Widget.CollapsiblePanel("CollapsiblePanel2", {contentIsOpen:false});
75 var CollapsiblePanel3 = new Spry.Widget.CollapsiblePanel("CollapsiblePanel3", {contentIsOpen:false});
76 var CollapsiblePanel4 = new Spry.Widget.CollapsiblePanel("CollapsiblePanel4", {contentIsOpen:false});
77 //-->
78 </script>
```

5. Delete the code shown on lines 73–76 of the preceding screenshot, and replace it with this single line of code:

```
var panelgroup = new Spry.Widget.CollapsiblePanelGroup("panelgroup");
```

6. Save `collapsible.html`, and test the page in Live view or in a browser. The first thing you should notice is that all the panels are open when the page loads, but you can open and close them independently.

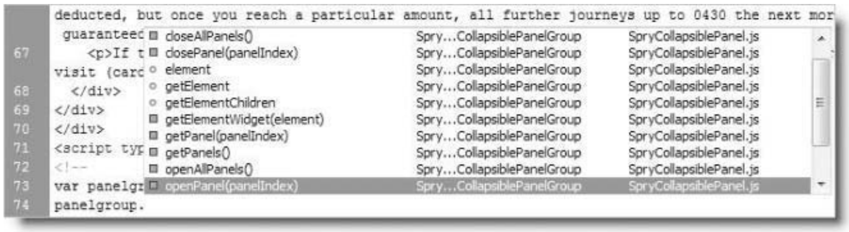
7. You probably don't want all of them open when the page loads, so amend the code at the bottom of the page like this:

```
var panelgroup = new Spry.Widget.CollapsiblePanelGroup("panelgroup", ➥
{contentIsOpen: false});
```

This passes the `contentIsOpen` property to the constructor and makes sure that all panels are closed when the page first loads.

8. What's that? You don't want them all closed? No problem. Remember that the code at the bottom of the page initializes Spry widgets when the page loads, so all you need to do is open one of the closed panels.

Insert a new line after the one you entered in the last step, and type panelgroup followed by a period. Since `panelgroup` is the variable to which you assigned the `CollapsiblePanelGroup` object, Dreamweaver displays code hints for the available properties and methods. Scroll down until you find openPanel(panelIndex), as shown here:



```
deducted, but once you reach a particular amount, all further journeys up to 0430 the next mor
 guaranteec
    <p>If t
visit (carc
  </div>
</div>
</div>
<script typ
<!--
var panelgr
panelgroup.
```

closeAllPanels()	Spry...CollapsiblePanelGroup	SpryCollapsiblePanel.js
closePanel(panelIndex)	Spry...CollapsiblePanelGroup	SpryCollapsiblePanel.js
element	Spry...CollapsiblePanelGroup	SpryCollapsiblePanel.js
getElement	Spry...CollapsiblePanelGroup	SpryCollapsiblePanel.js
getElementChildren	Spry...CollapsiblePanelGroup	SpryCollapsiblePanel.js
getElementWidget(element)	Spry...CollapsiblePanelGroup	SpryCollapsiblePanel.js
getPanel(panelIndex)	Spry...CollapsiblePanelGroup	SpryCollapsiblePanel.js
getPanels()	Spry...CollapsiblePanelGroup	SpryCollapsiblePanel.js
openAllPanels()	Spry...CollapsiblePanelGroup	SpryCollapsiblePanel.js
openPanel(panelIndex)	Spry...CollapsiblePanelGroup	SpryCollapsiblePanel.js

9. Double-click the code hint, or press Enter/Return to insert it. Then type the number of the panel you want to open (counting from zero), followed by a closing parenthesis and a semicolon. To open the first panel, the code looks like this:

```
panelgroup.openPanel(0);
```

10. Save `collapsible.html`, and test it again. This time, the first panel should slide open as the page loads (it might not render correctly in Live view, so test it in a browser).

11. As you can see in the preceding screenshot, the code hints for a `CollapsiblePanelGroup` object show that it has a `closeAllPanels()` method and an `openAllPanels()` one, too. So, to wire up the links to open and close all panels, all you need to do is create a dummy link on each one and add an `onclick` attribute to call the appropriate method on the `panelgroup` object. You have done this plenty of times before, so I'll just show the final code, which looks like this:

```
<p><a href="javascript:;" onclick="panelgroup.openAllPanels()">Open ➥
all</a> | <a href="javascript:;" onclick="panelgroup. ➥
closeAllPanels()">Close all</a></p>
```

12. Save the page, and test it. The panels now work both individually and as a group. There's just one problem: the `togglePanel()` function created in the preceding exercise no longer works because the individual objects identifying each panel no longer exist. Let's fix that.

13. To be able to toggle an individual panel open and closed, you need to know which panel group it belongs to and its position within the group. So, I have renamed the function `toggleGroupPanel()`, and the function will now take two arguments: `group` and `num`.

To find the individual panel, you first need to use the `getPanels()` method of the `CollapsiblePanelsGroup` object. This gets an array of all panels within the group. However, you can't just use the array index to get the panel. You need to pass the array element to the `getElementWidget()` method. Once you have identified the panel, the rest of the function remains the same. Here's the rewritten function with the amended parts highlighted in bold:

```
function toggleGroupPanel(group, num)
{
  var allPanels = group.getPanels();
  var panel = group.getElementWidget(allPanels[num-1]);
  if (panel.isOpen()) {
    panel.close();
  } else {
    panel.open();
  }
}
```

In the fourth line, I have subtracted 1 from the value of `num`, so the second argument passed to `toggleGroupPanel()` can use the more intuitive practice of counting the panels from one rather than zero.

14. Finally, amend the links that toggle the Water bus and Oyster card panels like this:

```
<p><a href="javascript:;" onclick="toggleGroupPanel(panelgroup, 3)"> ➥
Water bus</a></p>
<p><a href="javascript:;" onclick="toggleGroupPanel(panelgroup, 4)"> ➥
Oyster card</a></p>
```

Check your code, if necessary, against `collapsible_all.html` in `examples/ch08`.

A restriction with the `CollapsiblePanelsGroup` *object in Spry 1.6.1 appears to be that nothing else should be inside the outer* `<div>` *that's wrapped around the panels. Although everything works correctly to start with, the code rapidly gets confused and behaves erratically.*

So far, all the methods of opening panels have been confined to links on the same page. While that's useful, it's arguably more useful to be able to target a particular tab or panel to open when linked to from a different page. It can be done, but it requires part of the

Spry framework that's not included with Dreamweaver. I'll come back to that later in the chapter, but before that I'll show you how to combine different Spry effects to make custom effects of your own.

Using the Cluster object to combine effects

Spry effects bring together several complex actions to create a smooth transition onscreen. The secret weapon that makes this possible is the `Spry.Effect.Cluster` object, which determines whether to run each part of the effect simultaneously or in sequence. Since the built-in effects rely on the `Cluster` object, it's automatically at your disposal.

The `Cluster` object has many methods, but the following four are the ones that interest us:

- `call()`: This initiates the object. It expects two arguments: the effect's target element and an object literal containing any options.
- `addNextEffect()`: This chains effects in sequence. It takes an effect object as its sole argument.
- `addParallelEffect()`: This runs an effect in parallel with other effects. It takes an effect object as its sole argument.
- `start()`: This runs the effect. It takes no arguments.

To create a new effect, you need to extend the `Spry.Effect.Cluster` object. You do this by defining a function with the name of the new effect. Then you define a new JavaScript class using the same name and assigning its `prototype` object as `Spry.Effect.Cluster`. Finally, you assign the function as the constructor of the new class. It sounds more complicated than it really is. The basic syntax looks like this:

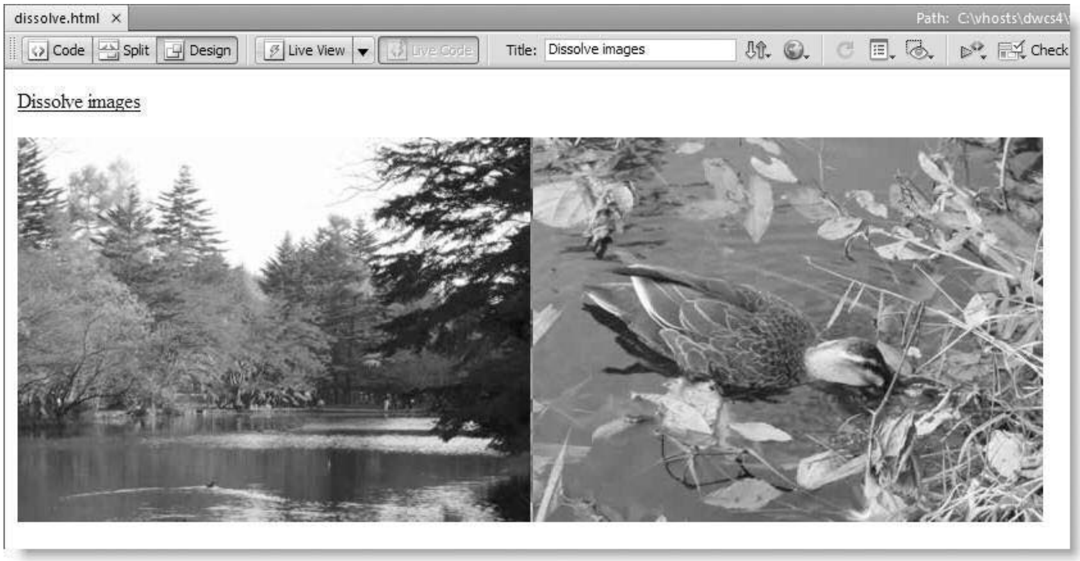
```
NewEffect = function(element, {options})
{
  Spry.Effect.Cluster.call(this, options);
  // details of effect go here
};
NewEffect.prototype = new Spry.Effect.Cluster();
NewEffect.prototype.constructor = NewEffect;
```

The best way to show you how to use the `Cluster` object is through a couple of practical examples. The next two exercises create a dissolve effect that can be used to fade one image into another, and an extension of the Spry highlight effect that makes a smooth transition to the final color.

Dissolving one image into another

This exercise demonstrates the use of the `addParallelEffect()` method of the `Cluster` object to fade out one image at the same time as another is faded in. Although images are used in this exercise, the effect could be applied to any elements on a page.

1. Copy `dissolve_start.html` from `examples/ch08`, and save it in `workfiles/ch08` as `dissolve.html`. The page contains a dummy link at the top and two images alongside each other inside a paragraph, as shown in the following screenshot:



   The two images will eventually be superimposed on each other. The image on the left has an ID called pond, and the other has an ID called duck. If you have a small monitor and the second image is pushed down below the first one, use two smaller images of your own.

2. Click the Live View button. The image of the duck should disappear.
3. Turn off Live view, and look in Code view to see why the duck vanished. The following `<style>` block embedded in the `<head>` of the page reduces the opacity of the duck image to zero when the page is displayed. In other words, whatever is behind it shows through.

```
<style type="text/css">
#duck {
  opacity: 0;
  filter: alpha(opacity=0);
}
</style>
```

   The `filter` property is nonstandard CSS but is required by Internet Explorer.

4. Create an external JavaScript file by selecting File ➤ New. In the Blank Page section of the New Document dialog box, select JavaScript as Page Type, and click Create. Save the new page as `clusters.js` in `workfiles/ch08`.
5. Following the basic syntax outlined earlier, let's call the new effect `Dissolve`. Add the following code to `clusters.js`:

```
Dissolve = function(elem1, elem2, duration)
{
  Spry.Effect.Cluster.call(this, {duration: duration});
};
Dissolve.prototype = new Spry.Effect.Cluster();
Dissolve.prototype.constructor = Dissolve;
```

Notice that I am using three arguments to be passed to the Dissolve effect: the first two are the IDs of the elements to be cross-faded, and the last one is for the duration in milliseconds. This is the only option, so it is passed to Spry.Effect.Cluster.call() as an object literal.

6. The Dissolve() function needs to instantiate two effects: one to reduce the opacity of the first element to zero and the other to increase the opacity of the second element from zero to fully opaque. The Spry effects library contains an object for precisely this purpose: Opacity. Amend the Dissolve() function definition like this:

```
Dissolve = function(elem1, elem2, duration)
{
  Spry.Effect.Cluster.call(this, {duration: duration});
  var fadeOut = new Spry.Effect.Opacity(elem1, 1, 0, {duration: ➥
duration, toggle: true});
  var fadeIn = new Spry.Effect.Opacity(elem2, 0, 1, {duration: ➥
duration, toggle: true});
};
Dissolve.prototype = new Spry.Effect.Cluster();
Dissolve.prototype.constructor = Dissolve;
```

The Opacity object takes four arguments: the target element, the starting opacity (1 is fully opaque, 0 is fully transparent), the ending opacity, and an object specifying any options. So, the Opacity object stored as fadeOut fades the first element from total opacity to total transparency, while fadeIn does the reverse to the second element. The same options are passed to both: they take the value of the duration property from the third argument passed to Dissolve() and set the toggle property to true. This last option reverses the effect the next time it is triggered.

7. With both effects stored as variables, you can now use the addParallelEffect() method to attach them to the target element (identified by this) as follows:

```
Dissolve = function(elem1, elem2, duration)
{
  Spry.Effect.Cluster.call(this, {duration: duration});
  var fadeOut = new Spry.Effect.Opacity(elem1, 1, 0, {duration: ➥
duration, toggle: true});
  var fadeIn = new Spry.Effect.Opacity(elem2, 0, 1, {duration: ➥
duration, toggle: true});
  this.addParallelEffect(fadeOut);
  this.addParallelEffect(fadeIn);
};
Dissolve.prototype = new Spry.Effect.Cluster();
Dissolve.prototype.constructor = Dissolve;
```

8. Save `clusters.js`, and switch back to `dissolve.html` in the Document window. The code you have just created is dependent on the `SpryEffects.js` external file, so both JavaScript files need to be attached to the HTML page.

9. A quick way to add external JavaScript files to a page is to display a representation of the page's <head> content in Design view. Select View ➤ Head Content, or press Ctrl+Shift+H/Shift+Cmd+H. This opens a section at the top of the Document window with icons representing HTML elements in the <head> of the page, as shown in Figure 8-2.

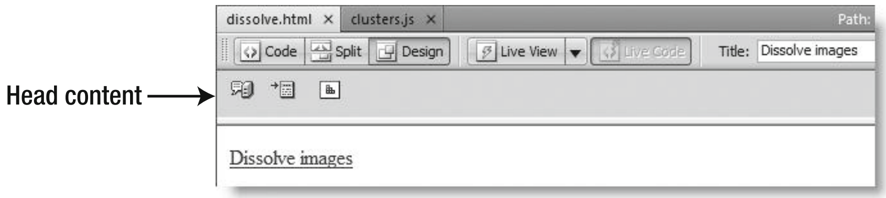
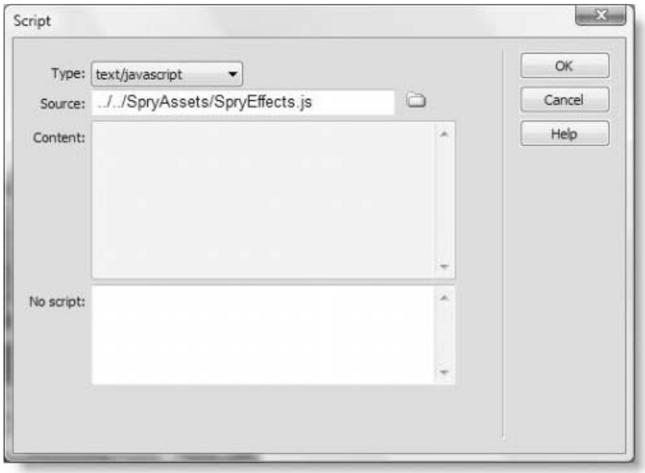


Figure 8-2. The <head> isn't visible in Design view, but you can inspect its contents by displaying them as icons.

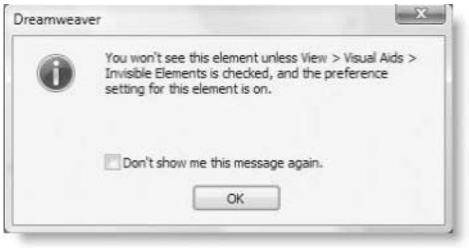
The icons are displayed in the same order as in the <head>, and you can drag them to the left or right to reposition them. You can also inspect and edit most elements by selecting an icon and viewing its contents in the Property inspector.

10. Click to the right of the last icon in the Head Content bar (it represents the embedded `<style>` block that you inspected in step 3). The Head Content bar should turn white to indicate that it has focus. Click the Script button in the Insert bar, or select Insert ➤ HTML ➤ Script Objects ➤ Script.

In the Script dialog box, click the folder icon alongside the Source field, navigate to `SpryAssets/SpryEffects.js`, and select it. Dreamweaver automatically selects text/javascript as the value for Type. Leave the Content and No script fields empty (these are for embedding JavaScript directly into the body of a page). The values should look like the following screenshot:


Script
Type: text/javascript
Source: ../../SpryAssets/SpryEffects.js
Content:
No script:

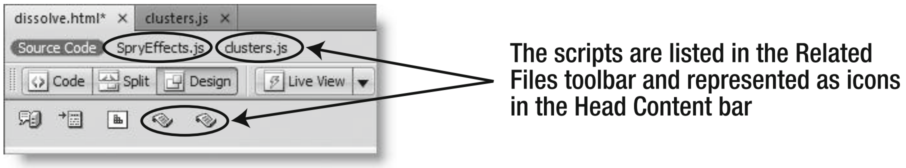
11. Click OK to close the Script dialog box. Dreamweaver will display the following message:



You won't see this element unless View > Visual Aids > Invisible Elements is checked, and the preference setting for this element is on.

Like the Content and No script fields, this applies only when you are embedding JavaScript directly into a page. You can safely ignore the message.

12. Repeat steps 10 and 11 to attach `clusters.js` to the page. There should now be two script icons in the Head Content bar, and both external files should be listed in the Related Files toolbar, as shown here:

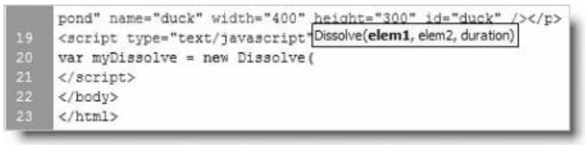

dissolve.html*
clusters.js
Source Code
SpryEffects.js
clusters.js
Code
Split
Design
Live View
The scripts are listed in the Related Files toolbar and represented as icons in the Head Content bar

13. Close the separate tab that contains `clusters.js`. You'll work with it through the Related Files feature from now on, so having two versions open in the Document window is likely to lead to confusion. You can also close the Head Content bar by selecting Head Content in the View menu or by pressing Ctrl+Shift+H/Shift+Cmd+H.

14. Switch to Code view, and create a `<script>` block at the foot of the page, just before the closing `</body>` tag. Create a `Dissolve` object like this:

```
var myDissolve = new Dissolve('pond', 'duck', 2000);
```

As soon as you type the opening parenthesis after Dissolve, Dreamweaver should display code hints for your newly defined effect like this:



```
pond" name="duck" width="400" height="300" id="duck" /></p>
<script type="text/javascript"
var myDissolve = new Dissolve(
</script>
</body>
</html>
```

This is Dreamweaver CS4's new code introspection at work.

15. Add an onclick event to the dummy link at the top of the page, and set it to apply the `start()` method to the effect you have just created like this (you refer to it through the variable in which it is stored):

```
<a href="javascript:;" onclick="myDissolve.start()">Dissolve images</a>
```

16. Save the page, and activate Live view. Click the Dissolve images link at the top of the page, and the two images should begin a simultaneous transition: the pond fading out and the duck fading in, as shown in Figure 8-3.

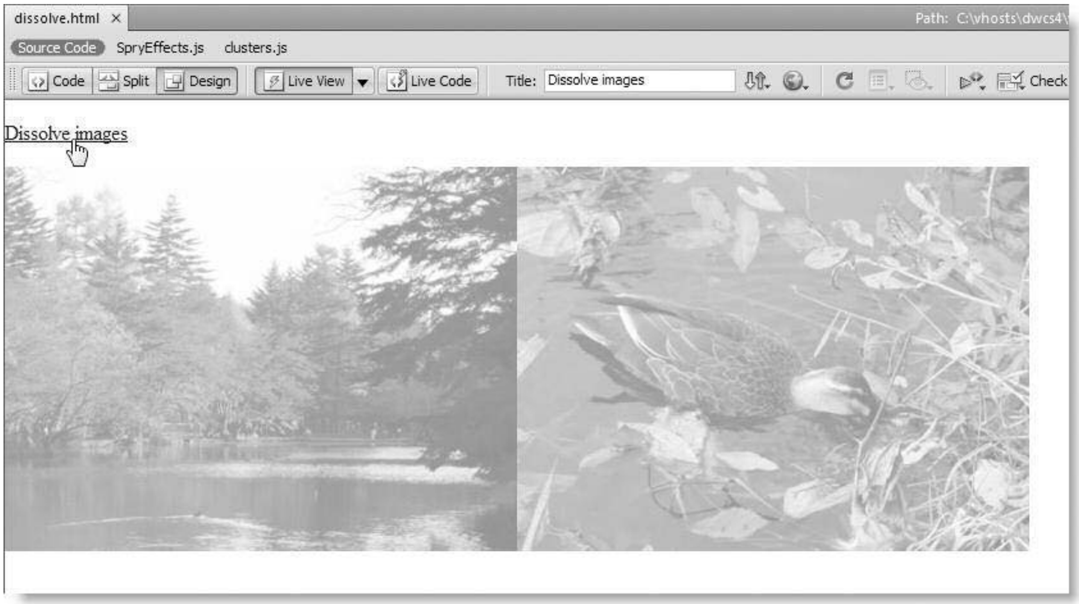


Figure 8-3. The new Dissolve effect switches the transparency of both images simultaneously.

17. If the effect doesn't work, load the page into a browser, such as Firefox, and use Tools ➤ Error Console (or a debugging extension, such as Firebug) to troubleshoot any JavaScript errors. You can also compare your files with `dissolve_01.html` and `clusters_01.js` in `examples/ch08`.

18. Amend the style rule for the duck image like this:

```
#duck {
  opacity: 0;
  filter: alpha(opacity=0);
  position: relative;
  left: -400px;
}
```

Both images are 400 pixels wide, so this simply moves the duck image the same distance to the left so that both images are superimposed. Note that this won't work if the browser window is less than 800 pixels wide, because the second image will drop down and be pushed too far left. If this happens, you might need to use absolute positioning instead.

If you test the page now, the images should dissolve from one to the other.

19. There's just one refinement that needs to be made to `clusters.js`. It's a good idea to set a default `duration` property. Then, the effect can be instantiated with just two arguments: the IDs of the elements you want to dissolve. Amend the code in clusters.js like this:

```
Dissolve = function(elem1, elem2, duration)
{
  var dur = 2000;
  if (duration != null) dur = duration;
  Spry.Effect.Cluster.call(this, {duration: dur});
  var fadeOut = new Spry.Effect.Opacity(elem1, 1, 0, {duration: dur, ➥
toggle: true});
  var fadeIn = new Spry.Effect.Opacity(elem2, 0, 1, {duration: dur, ➥
toggle: true});
  this.addParallelEffect(fadeOut);
  this.addParallelEffect(fadeIn);
};
Dissolve.prototype = new Spry.Effect.Cluster();
Dissolve.prototype.constructor = Dissolve;
```

The two new lines added at the top of the function create a variable, dur, with a default value of 2000. If the third variable passed to the Dissolve() constructor is omitted, it uses the default value. Note that the variable, dur, is now used as the value for the duration property in all the option objects.

20. Remove the duration from the code that instantiates the Dissolve object at the bottom of dissolve.html like this:

```
var myDissolve = new Dissolve('pond', 'duck');
```

21. Save both dissolve.html and clusters.js, and test them. The effect should now use the default duration of 2000 milliseconds. If you add a different value, it will use that instead.

Check your code, if necessary, against dissolve.html and clusters_dissolve.js in examples/ch08.

The next exercise shows how to create a custom effect that chains effects one after another. Rather than go through everything step by step, I'll just explain the main points, because the principles are the same as when running effects in parallel.

Creating a smooth highlight transition

The default Spry highlight effect uses three colors: a start color, the end color, and the color to which the background reverts at the end of the transition. I find this sudden switch at the end rather jarring, so this exercise creates a new effect that runs two color transitions in sequence.

1. Add the following code to clusters.js from the preceding exercise:

```
HighlightTransition = function(element, options)
{
  Spry.Effect.Cluster.call(this, options);
  var col1 = '#FFFFFF';
  var col2 = '#DCBD7D';
  var col3 = '#FFFFFF';
  var dur1 = 2000;
  var dur2 = 2000;
  if (options.col1 != null) col1 = options.col1;
  if (options.col2 != null) col2 = options.col2;
  if (options.col3 != null) col3 = options.col3;
  if (options.dur1 != null) dur1 = options.dur1;
  if (options.dur2 != null) dur2 = options.dur2;
  var transition1 = new Spry.Effect.Color(element, col1, col2, ➥
{duration: dur1, transition: Spry.sinusoidalTransition});
  var transition2 = new Spry.Effect.Color(element, col2, col3, ➥
{duration: dur2, transition: Spry.sinusoidalTransition});
  this.addNextEffect(transition1);
  this.addNextEffect(transition2);
};
HighlightTransition.prototype = new Spry.Effect.Cluster();
HighlightTransition.prototype.constructor = HighlightTransition;
```

This defines a new HighlightTransition object using the same syntax as before to extend the Spry.Effect.Cluster object. The important lines are highlighted in bold. They create two Spry Color objects and then add them to the current object using the addNextEffect() method. This runs the effects in sequence one after the other, instead of running them in parallel like the Dissolve effect.

The Spry Color object is another basic effect in the Spry effects library. It takes four arguments: the target element, the starting color, the end color, and an object literal with any options. I have used two options: the duration of the effect and the type of transition. The Spry.sinusoidalTransition starts slowly, speeds up in the middle, and then slows down again at the end. Table 8-1 lists the available transition options for Spry effects.

The first effect, stored as transition1, changes the background color of the target element from col1 to col2, and the second effect (transition2) changes the background color from col2 to col3.

The rest of the code sets defaults for all the colors and durations. This means you need set only those options that you want to change from the default, although you must set at least one option for the effect to work.

Table 8-1. Transition options for Spry effects

Transition	Description
Spry.linearTransition	Progresses evenly throughout
Spry.circleTransition	Rapid start followed by a long easing
Spry.fifthTransition	Similar to Spry.linearTransition but eases toward the end
Spry.growSpecificTransition	Starts gently, then dips back before rapid finish

Transition	Description
`Spry.pulsateTransition`	Rapid pulsation between start and finish values, ending with finish value
`Spry.sinusoidalTransition`	Starts slowly, speeds up, then eases toward the end
`Spry.squareTransition`	Starts slowly and gradually speeds up
`Spry.squarerootTransition`	Starts quickly and gradually eases

2. Copy `highlight_transition_start.html` from `examples/ch08`, and save it as `highlight_transition.html` in `workfiles/ch08`.
3. Link `SpryEffects.js` and `clusters.js` to `highlight_transition.html` in the same way as in steps 10–12 of the preceding exercise.
4. The image has 20 pixels of padding that can be used as a test for the new highlight effect. The image's ID is `goldenpav`, so add the following code to the bottom of the page to initialize a `HighlightTransition` object:

   ```
   <script type="text/javascript">
   var myHighlight = new HighlightTransition('goldenpav', {dur2: 1000});
   </script>
   ```

   The options must be passed to the constructor method as an object literal, using the same names as in `HighlightTransition` definition in step 1 (`col1`, `col2`, `col3`, `dur1`, and `dur2`). You must pass at least one option to the constructor in this way. This example changes the duration of the second color change from the default 2000 milliseconds to 1000.
5. Add an `onclick` attribute to the dummy link at the top of the page to trigger the effect like this:

   ```
   <p><a href="javascript:;" onclick="myHighlight.start()">Highlight ➥
   image</a></p>
   ```
6. Test the page. The image should be surrounded by a golden brown border that fades in and out smoothly. Experiment with other colors and durations.

   Check your code, if necessary, with `highlight_transition.html` and `clusters.js` in `examples/ch08`.

Using Spry utilities

As I explained in Chapter 7, Spry is software neutral. You can download the latest copy of the Spry framework from Adobe Labs at `http://labs.adobe.com/technologies/spry/home.html` and use it with any script editor. At the time of this writing, the current version is 1.6.1, which is the same as Dreamweaver CS4, although newer versions will be posted when available. In addition to the same external JavaScript files that Dreamweaver uses, the Spry framework contains a lot of documentation and samples. If you're interested in getting the most out of Spry, it's well worth downloading. The drawback for inexperienced developers is that most examples assume a good understanding of JavaScript. Often the explanation of how something works is lurking in comments in the source code.

The full Spry framework also includes several useful files that are missing from Dreamweaver. Two of the most useful are `SpryDOMUtils.js`, which makes it easy to manipulate the DOM (see Chapter 7), and `SpryURLUtils.js`, which lets you pass options to Spry objects through a URL—essential for opening a specific panel from a link on a different page.

To continue with the exercises in this section, you need to download the most recent version of the Spry framework from `http://labs.adobe.com/technologies/spry/home.html` and unzip the compressed file. The `Readme.html` and `docs.html` files contain links to all the documentation and samples. I'll leave you to explore them at your leisure. The files you need for the following exercises are in the `includes` folder. Copy `SpryDOMUtils.js` and `SpryURLUtils.js` to the `SpryAssets` folder in the site you're using for this book.

Passing information to a Spry widget through a URL

When you link from one page to another, you can pass information to the target page by adding parameters to the end of the URL. There are two ways of doing this:

- A query string: This is a series of name/value pairs following a question mark, like this: `?variable1=value1&variable2=value2`. Each name is separated from its value by an equal sign, and each pair is separated by an ampersand (in XHTML, the ampersand needs to be embedded in the link as `&amp;`).
- A fragment identifier: This is the hash (or pound) symbol followed by the name of an ID or anchor tag, indicating the section of the page you want the browser to go to, for example, `#thisSection`.

The `SpryURLUtils.js` file contains a method called `getLocationParamsAsObject()`, which extracts this information from a URL. You can then pass this information to the code that initializes the Spry widget when the page loads.

Opening a tab or accordion panel from another page

To open a specific tab or panel in a Spry widget on a different page, you need to pass the information as a query string. For example, to open the second accordion panel, you would add this to the end of the URL: `?panel=1`. If the panel is identified by an ID, you pass the ID as the value instead, for example, `?panel=waterbus`.

To open a specific tab or panel—and go straight to it—you need to combine both methods like this: `?panel=waterbus#waterbus`.

It's important to get the order right. The query string must come before the fragment identifier. If you put them the other way round, both sets of information will be ignored.

In the page that contains the Spry widget, you use the `getLocationParamsAsObject()` method in `SpryURLUtils.js` like this:

```
var params = Spry.Utils.getLocationParamsAsObject();
```

This stores the query string as an object called `params`, enabling you to pass the values it contains to the widget's constructor method. Since the page might be accessed directly, the values passed to the constructor need to use the JavaScript conditional (or ternary) operator like this:

```
{defaultTab: params.tab ? params.tab : 0}
```

If the URL used to access the page has a query string that contains a variable called `tab`, its value will be held in `params.tab`. This rather cryptic piece of code means "If `params.tab` exists, assign its value to `defaultTab`; but if `params.tab` doesn't exist, use `0` instead."

That's the theory. Now, let's get coding.

Preparing the target page

This exercise demonstrates how to open a specific tab of a tabbed panels widget from a link in another page. The same technique applies to an accordion.

1. Copy `tabbed_start.html` from `examples/ch08` to `workfiles/ch08`, and rename it `tabbed_other.html`.
2. Attach `SpryURLUtils.js` by adding it to the `<head>` of `tabbed_other.html`. If you're not sure how to do this, use the same technique as described in steps 10 and 11 of the "Dissolving one image into another" exercise earlier in the chapter.
3. Switch to Code view, and add the following code block inside the `<head>` section. It doesn't matter where it goes, but it must come after the `<script>` tag that attaches `SpryURLUtils.js` to the page. Spry code hints should help you get the spelling and combination of uppercase and lowercase correct.

   ```
   <script type="text/javascript">
   var params = Spry.Utils.getLocationParamsAsObject();
   </script>
   ```

   This calls the `getLocationParamsAsObject()` method from `SpryURLUtils.js`, which converts all the information passed to the page through the URL into a JavaScript object called `params`. You can now use `params` to retrieve the values from the URL.
4. Scroll down to the bottom of the page until you come to the code that initializes the tabbed panels. It currently looks like this:

   ```
    var TabbedPanels1 = new Spry.Widget.TabbedPanels("TabbedPanels1");
   ```

5. To open a specific panel, you need to pass a second argument to the constructor method. As explained in "Initializing a Spry object" earlier in this chapter, this needs to be in the form of an object literal. For a tabbed panels widget, the

property that controls the default panel is called defaultTab. For an accordion, it's defaultPanel.

If the value of the tab or panel you want to open is passed through the URL, it will be a property of the params object you created in step 3. You can call the properties sent through the URL anything you like, but it makes sense to use tab for a tabbed panels widget and panel for an accordion. So, the selected value will be params.tab or params.panel.

However, you need to take into account the likelihood that nothing is passed through the URL (for example, when a user accesses the page directly). So, change the code in step 4 like this:

```
var TabbedPanels1 = new Spry.Widget.TabbedPanels("TabbedPanels1",
{defaultTab: params.tab ? params.tab : 0});
```

If you're using an accordion, the code should look like this:

```
var Accordion1 = new Spry.Widget.Accordion("Accordion1",
{defaultPanel: params.panel ? params.panel : 0});
```

This uses the conditional (ternary) operator, which is the same in both JavaScript and PHP, to determine the value assigned to defaultTab or defaultPanel. When used like this with an object literal, the conditional operator can seem confusing because it also uses a colon. The first colon is part of the object literal syntax and separates the object property from its value. The second colon is part of the conditional operator, which comprises a question mark and a colon.

If the expression to the left of the question mark equates to true, the value immediately to the right of the question mark is used. However, if the expression equates to false, the value following the colon is used instead.

So if params.tab or params.panel has a value, it will equate to true, and its value will be assigned to the defaultTab or defaultPanel property. If params.tab or params.panel doesn't have a value, 0 is used instead, making the first tab or panel the default.

6. Tabs and panels can be identified either by their index (position within the widget counted from zero) or by an ID. When linking from another page, it's safer to use an ID in case the order of tabs/panels changes. Instructions on how to add an ID were given in the exercises on creating links from the same page earlier in this chapter.

   For the purposes of this exercise, give the third panel an ID of waterbus.

7. Save tabbed_other.html, and test it in a browser. The first tab should be displayed when the page loads.

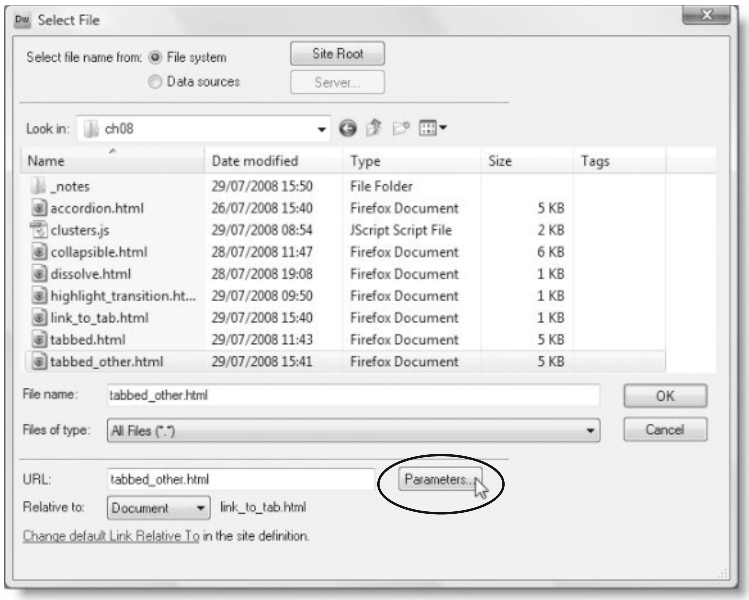
   You can check your code, if necessary, against tabbed_other.html in examples/ch08.

That finishes the changes to the target page. There is no need to create named anchors for the tabbed panels or accordion, because you can use the ID Dreamweaver automatically assigns to each set of tabbed panels or accordion.

Creating the link from the other page

All that's necessary now is to create a link to the target page using a query string, as described at the beginning of this section.

1. Create a new HTML page, and save it as `link_to_tab.html` in `workfiles/ch08`.
2. Type some text in the page to use as a link to the Water bus tab of `tabbed_other.html`.
3. Highlight the text you plan to use as a link, and select the HTML view of the Property inspector. You can type the link and query string directly into the Link field. However, if you prefer to let Dreamweaver create the correct syntax for you, click the folder icon to the right of the Link field.
4. In the Select File dialog box, select `tabbed_other.html`, and click the Parameters button, as shown in the following screenshot:



Select File

Select file name from: (●) File system ( ) Data sources

Look in: ch08

Name	Date modified	Type	Size	Tags
_notes	29/07/2008 15:50	File Folder		
accordion.html	26/07/2008 15:40	Firefox Document	5 KB	
clusters.js	29/07/2008 08:54	JScript Script File	2 KB	
collapsible.html	28/07/2008 11:47	Firefox Document	6 KB	
dissolve.html	28/07/2008 19:08	Firefox Document	1 KB	
highlight_transition.ht...	29/07/2008 09:50	Firefox Document	1 KB	
link_to_tab.html	29/07/2008 15:40	Firefox Document	1 KB	
tabbed.html	29/07/2008 11:43	Firefox Document	5 KB	
tabbed_other.html	29/07/2008 15:41	Firefox Document	5 KB	

File name: tabbed_other.html

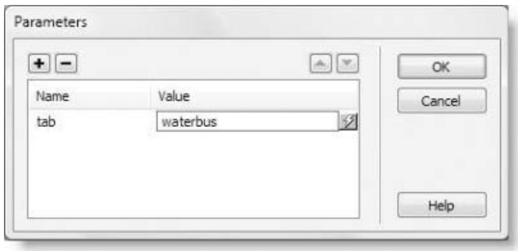
Files of type: All Files (*.*)

URL: tabbed_other.html

Relative to: Document link_to_tab.html

Change default Link Relative To in the site definition.

5. In the Parameters dialog box, enter tab in the Name field. Then use the Tab key or mouse to open the Value field, and enter waterbus, as shown here:



Parameters

Name	Value
tab	waterbus

On this occasion, the query string consists of a single name/value pair, but a query string can contain several pairs. Use the plus and minus keys to add or remove name/value pairs. You can also change their order with the up and down arrows.

6. Click OK to close the Parameters dialog box, and then click OK again (Choose on a Mac) to close the Select File dialog box.

   The value in the Link field of the Property inspector should now look like this:

   ```
   tabbed_other.html?tab=waterbus
   ```

7. Save `link_to_tab.html`, and load it in a browser. Click the link. This time, when `tabbed_other.html` loads, the Water bus tab should be displayed instead of the first tab.

   Check your code, if necessary, against `link_to_tab.html` and `tabbed_other.html` in `examples/ch08`.

Opening a collapsible panel from another page

The principle behind opening a collapsible panel through a URL is identical to opening a tab or accordion panel. The main difference is that each panel is independent. Its open or closed state is determined by the `contentIsOpen` option. Consequently, the ID is not important when sending a query string. All you need is a name to identify the panel and to give it a value of `true` or `false`.

You can see a working example of this in `collapsible_other.html` and `link_to_collapsible.html` in `examples/ch08`. The query string in `link_to_collapsible.html` looks like this:

```
<a href="collapsible_other.html?oyster=true">Oyster Card</a>
```

The code that initializes the fourth collapsible panel in `collapsible_other.html` looks like this:

```
var CollapsiblePanel4 = new Spry.Widget.CollapsiblePanel( ➥
"CollapsiblePanel4", {contentIsOpen: params.oyster ? ➥
params.oyster : false});
```

This means that if the URL contains a variable called `oyster`, its value will be used for the `contentIsOpen` option. Otherwise, `contentIsOpen` is set to `false`.

You could, in fact, dispense with a value for `oyster` and use this instead:

```
var CollapsiblePanel4 = new Spry.Widget.CollapsiblePanel( ➥
"CollapsiblePanel4", {contentIsOpen: params.oyster ? ➥
true : false});
```

Selecting and manipulating page elements with Spry.$$

If you thought Spry was just about widgets and effects, think again. In common with other JavaScript frameworks like Prototype and jQuery, Spry uses CSS selectors to manipulate the DOM and change the look or behavior of targeted page elements. Table 8-2 describes the selectors supported by Spry 1.6.1. If you're familiar with either Prototype or jQuery, you'll immediately recognize them. They're based on the proposed selectors for CSS3 (`http://www.w3.org/TR/css3-selectors`). Although CSS3 is still a long way from becoming a reality, the selectors have basically been agreed upon, so learning them for use with a JavaScript framework serves a dual purpose.

While the Spry selector utility matches Prototype and jQuery in its ability to select elements on a page, it currently has only ten methods (listed in Table 8-3) that manipulate the DOM. They're mainly useful for changing the CSS styles of an element in response to a JavaScript event.

The Spry selector utility uses the Prototype convention of two dollar signs to select elements but avoids conflict with other frameworks by prefixing them with `Spry`. The following code selects all elements that use a class called `optional`:

```
Spry.$$('.optional')
```

As a simple example of how you can use the selector utility, you can create a function to toggle the display of selected elements on and off by creating a class called `hideMe` with the property `display: none` like this:

```
function toggleOpts()
{
  Spry.$$('.optional').toggleClassName('hideMe');
}
```

Table 8-2. CSS selectors supported by Spry.$$, as of Spry 1.6.1

Pattern	Meaning	Example
`*`	Any element.	`Spry.$$(*)`
`E`	An element of type `E`, e.g., an HTML tag.	`Spry.$$('div')`
`E.class`	An `E` element with a specified class (the element is optional).	`Spry.$$('img.floatleft')` `Spry.$$('.floatleft')`
`E#id`	An `E` element with a specified ID (the element is optional).	`Spry.$$('div#nav')` `Spry.$$('#nav')`
`E F`	An `F` element descendant of an `E` element, e.g., all links in unordered lists.	`Spry.$$('ul a')`
`E > F`	An `F` element that is a direct child of an `E` element.	`Spry.$$('p > a')`
`E + F`	An `F` element immediately preceded by an `E` element (an adjacent sibling), e.g., the first paragraph after a level 1 heading.	`Spry.$$('h1 + p')`

Continued

Table 8-2. Continued

Pattern	Meaning	Example
`E ~ F`	All F elements preceded by having the same parent as an E element, e.g., all paragraphs at the same level as a level 1 heading that precedes them. Other elements may intervene.	`Spry.$$('h1 ~ p')`
`E[foo]`	An E element with a foo attribute, e.g., all links with a `title` attribute. Do not use `E[class]` as a bug in Internet Explorer adds a class attribute to every element.	`Spry.$$('a[title]')`
`E[foo="bar"]`	An E element with a foo attribute exactly equal to "bar".	`Spry.$$('img[width="50"]')`
`E[foo^="bar"]`	An E element with a foo attribute that begins with the string "bar".	`Spry.$$('img[title^="Art"]')`
`E[foo$="bar"]`	An E element with a foo attribute that ends with the string "bar".	`Spry.$$('a[href$=".pdf"]')`
`E[foo*="bar"]`	An E element with a foo attribute that contains the substring "bar".	`Spry.$$('p[class*="left"]')`
`E[foo~="bar"]`	An E element with a foo attribute that comprises a list of space-separated values, one of which is exactly equal to "bar".	`Spry.$$('p[class~="warn"]')`
`E:first-child`	An E element that is the first child of its parent, e.g., the first row in a table.	`Spry.$$('tr:first-child')`
`E:last-child`	An E element that is the last child of its parent.	`Spry.$$('tr:last-child')`
`E:only-child`	An E element that is the only child of its parent, e.g., an image wrapped in a `<div>`.	`Spry.$$('img:only-child')`
`E:first-of-type`	An E element that is the first sibling of its type, e.g., the first cell in a table row.	`Spry.$$('td:first-of-type')`
`E:last-of-type`	An E element that is the last sibling of its type, e.g., the last cell in a table row.	`Spry.$$('td:last-of-type')`
`E:only-of-type`	An E element that is the only sibling of its type.	`Spry.$$('img:only-of-type')`
`E:nth-child(n)`	An E element that is the nth child of its parent (see main text for an explanation).	

Pattern	Meaning	Example
`E:nth-last-child(n)`	An E element that is the nth child of its parent, counting from the last one.	
`E:nth-of-type(n)`	An E element that is the nth sibling of its type.	
`E:nth-last-of-type(n)`	An E element that is the nth sibling of its type, counting from the last one.	
`E:empty`	An E element that has no children (including text nodes).	`Spry.$$('td:empty')`
`E:not(s)`	An E element that does not match simple selector s, e.g., everything except a paragraph.	`Spry.$$('*:not(p)')`
`E:checked`	An E element that is checked (radio buttons or checkboxes).	`Spry.$$('input:checked')`
`E:disabled`	A form E element that is disabled.	`Spry.$$('input:disabled')`
`E:enabled`	Form elements that are not explicitly disabled.	`Spry.$$('input:enabled')`
`E[hreflang|="en"]`	An E element with an `hreflang` attribute that has a hyphen-separated list of values beginning with "en".	`Spry.$$('link[hreflang|="en"]')`

Attribute selectors do not permit spaces around the operators. For example, the following is incorrect:

```
Spry.$$('a[href $= ".pdf"]') // WRONG
```

It must be like this:

```
Spry.$$('a[href$=".pdf"]')   // RIGHT
```

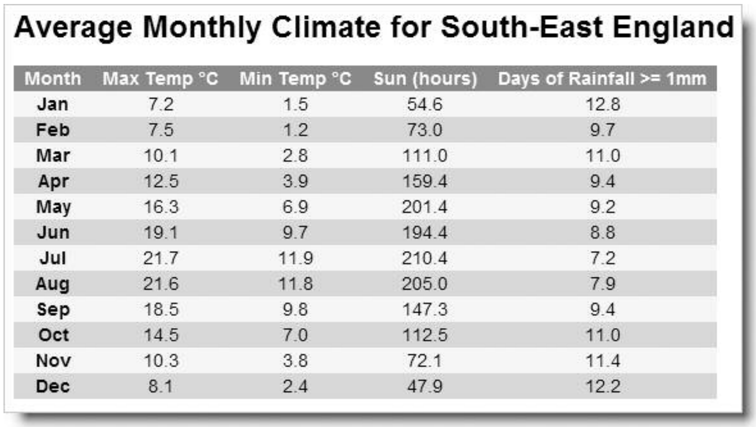
The `nth-child` selectors are designed to select elements in a repeating pattern. The simplest way to use them is for odd and even elements like this:

```
tr:nth-child(odd)   // picks odd rows
tr:nth-child(even)  // picks even rows
```

The following function (in odd_even.html in examples/ch08) adds class names to odd and even table rows:

```
function init() {
  Spry.$$('tr:nth-child(odd)').addClassName('odd');
  Spry.$$('tr:nth-child(even)').addClassName('even');
  Spry.$$('tr:first-child').removeClassName('odd'). ➥
addClassName('headerRow');
}
```

The function runs when the page loads and produces striped table rows, as shown in Figure 8-4. The final line uses the first-child selector to remove the odd class from the first row and apply a different class. Spry selector utility methods can be chained in the same way as with other JavaScript libraries.



Average Monthly Climate for South-East England

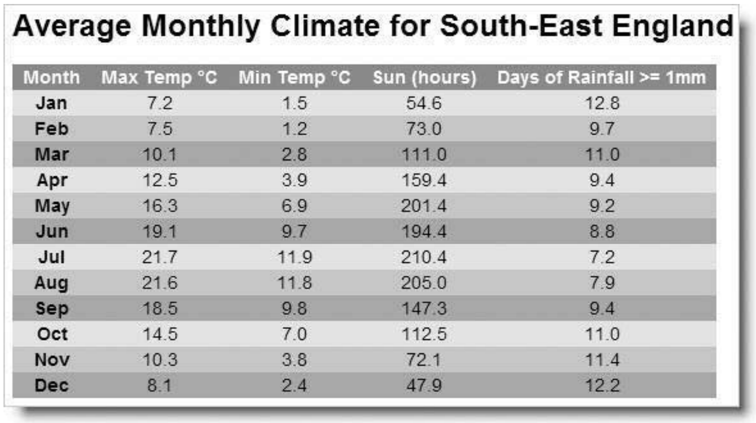
Month	Max Temp °C	Min Temp °C	Sun (hours)	Days of Rainfall >= 1mm
Jan	7.2	1.5	54.6	12.8
Feb	7.5	1.2	73.0	9.7
Mar	10.1	2.8	111.0	11.0
Apr	12.5	3.9	159.4	9.4
May	16.3	6.9	201.4	9.2
Jun	19.1	9.7	194.4	8.8
Jul	21.7	11.9	210.4	7.2
Aug	21.6	11.8	205.0	7.9
Sep	18.5	9.8	147.3	9.4
Oct	14.5	7.0	112.5	11.0
Nov	10.3	3.8	72.1	11.4
Dec	8.1	2.4	47.9	12.2

Figure 8-4. The alternating background colors are applied automatically to odd and even rows.

You can achieve even more ambitious effects with nth-child by using the formula *an+b*, where *a* and *b* are both numbers. The first number represents how many elements are in the repeat sequence. The second number identifies the element that you want to select within the sequence. So if you want a repeating pattern of three, the formula works like this:

```
tr:nth-child(3n+1)  // picks rows 1, 4, 7, etc
tr:nth-child(3n+2)  // picks rows 2, 5, 8, etc
tr:nth-child(3n+3)  // picks rows 3, 6, 9, etc
```

You can see the effect in Figure 8-5 and nth-child.html in examples/ch08.



Average Monthly Climate for South-East England

Month	Max Temp °C	Min Temp °C	Sun (hours)	Days of Rainfall >= 1mm
Jan	7.2	1.5	54.6	12.8
Feb	7.5	1.2	73.0	9.7
Mar	10.1	2.8	111.0	11.0
Apr	12.5	3.9	159.4	9.4
May	16.3	6.9	201.4	9.2
Jun	19.1	9.7	194.4	8.8
Jul	21.7	11.9	210.4	7.2
Aug	21.6	11.8	205.0	7.9
Sep	18.5	9.8	147.3	9.4
Oct	14.5	7.0	112.5	11.0
Nov	10.3	3.8	72.1	11.4
Dec	8.1	2.4	47.9	12.2

Figure 8-5. Using the nth-child selector targets repeating elements in a user-defined sequence.

Table 8-3. Methods used by the Spry selector utility

Method	Argument(s)	Description
`addClassName()`	`class`	Adds the specified class to all selected elements. The argument should be in quotes.
`addEventListener()`	`event, handler, capture`	Adds a listener for the specified event. The first argument should be a string consisting of the event name (without "on"). The second argument is the name of the function to be used as the event handler. The final argument is a Boolean (`true or false`) that specifies whether the handler should respond in the capture phase. Internet Explorer does not support the capture phase, so you should normally use `false`.
`forEach()`	`function`	Runs the specified function on each selected element.
`removeAttribute()`	`attribute`	Removes the specified attribute from the selected elements. The name of the attribute should be in quotes.
`removeEventListener()`	`event, handler, capture`	Removes the specified event listener. The arguments are the same as for `addEventListener()`.
`removeClassName()`	`class`	Removes the specified class. The class name should be in quotes.

Continued

Table 8-3. Continued

Method	Argument(s)	Description
`setAttribute()`	`attribute`, `value`	Adds the attribute and value to all selected elements. Both arguments should be in quotes.
`setProperty()`	`property`, `value`	Sets a property on the selected object(s). Both arguments should be in quotes.
`setStyle()`	`style`	Sets the specified styles on the selected elements. The argument should be a string consisting of CSS property/value pairs separated by semicolons.
`toggleClassName()`	`class`	Removes the specified class if it already exists on the selected elements. Otherwise, adds it. The class name should be in quotes.

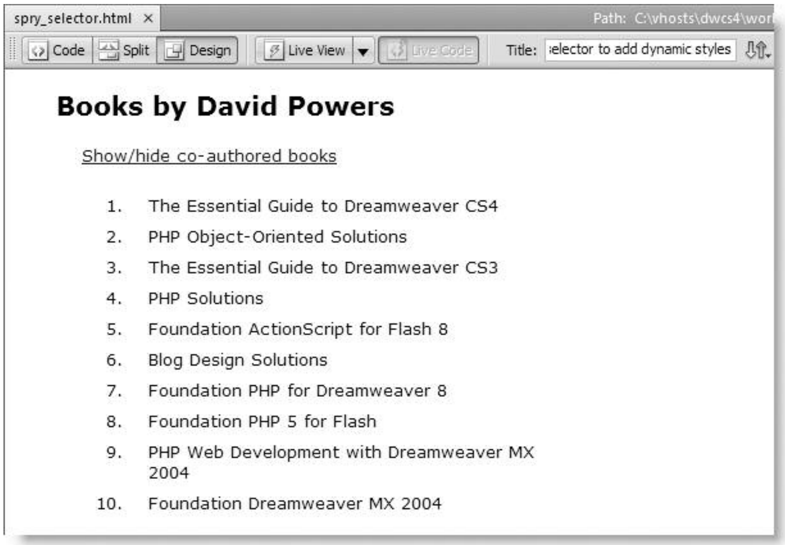
I have included Tables 8-2 and 8-3 to whet the appetite of readers who already have some experience with JavaScript and encourage them to delve deeper into the Spry application programming interface (API). If you're new to JavaScript, all this might seem like impenetrable gobbledygook, but you should have little difficulty implementing the code in the following exercise.

To get up to speed on JavaScript, I suggest you read an up-to-date introductory text, such as Beginning JavaScript with DOM Scripting and Ajax: From Novice to Professional *by Christian Heilmann (Apress, ISBN: 978-1-59059-680-7). Do not read anything published before, say, 2005. The whole approach to JavaScript has changed radically since the early days of the Web. It's important not to get stuck with outdated concepts and techniques.*

Styling on the fly with the Spry selector

This exercise uses the `Spry.$$` selector to style alternate items in an ordered list with a different background color. It also uses a class selector to toggle on and off the display of certain items. The page also gracefully degrades in a browser that has JavaScript disabled.

1. Copy `spry_selector_start.html` from `examples/ch08`, and save it in `workfiles/ch08` as `spry_selector.html`. The page looks like the following screenshot.


spry_selector.html
Path: C:\vhosts\dwcs4\wor
Code
Split
Design
Live View
Live Code
Title: elector to add dynamic styles

Books by David Powers

Show/hide co-authored books

1. The Essential Guide to Dreamweaver CS4
2. PHP Object-Oriented Solutions
3. The Essential Guide to Dreamweaver CS3
4. PHP Solutions
5. Foundation ActionScript for Flash 8
6. Blog Design Solutions
7. Foundation PHP for Dreamweaver 8
8. Foundation PHP 5 for Flash
9. PHP Web Development with Dreamweaver MX 2004
10. Foundation Dreamweaver MX 2004

It contains an ordered list of books that I have written for friends of ED and Apress over the past few years. Some of the books were coauthored with other writers. The dummy link at the top of the page will be used to hide and display those books.

2. Open Code view. You'll see that, in addition to a few style rules to improve the look of the text, there are three classes embedded in the `<head>` of the page: `odd`, `even`, and `hideMe`.

The only class that's added to any of the HTML tags is `coauthored`, but there are no style rules for the `coauthored` class. That's because you're going to use that class to identify the books that will be hidden or displayed when the link is clicked at the top of the page.

3. To use the `Spry.$$` selector, you need to attach `SpryDOMUtils.js` to the page `<head>`. You should be familiar with doing this by now, but refer to steps 9–11 of the "Dissolving one image into another" exercise if you're still unsure.

4. Let's start off by giving the list items an alternating background color. Add the following `<script>` block to the `<head>` anywhere after the `<script>` tag that links `SpryDOMUtils.js` to the page (code hints will help you a lot with the typing):

```
<script type="text/javascript">
function init()
{
  Spry.$$('li:nth-child(odd)').addClassName('odd');
  Spry.$$('li:nth-child(even)').addClassName('even');
}
</script>
```

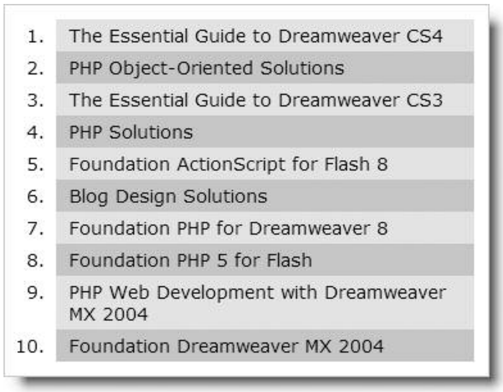
This uses the `nth-child` structural pseudo-selector to select odd and even `<li>` tags and adds the appropriate class to each one.

8

5. What you have just created is a function, so you need to trigger it to run when the page loads. Either you can put a call to the function in a `<script>` block at the bottom of the page, as Dreamweaver does with the calls to the widget constructors, or you can add it to the `<body>` tag as an `onload` event. Let's take the latter course, so amend the `<body>` tag like this:

```
<body onload="init()">
```

6. Switch to Design view, and activate Live view. The list should now look like this:



1. The Essential Guide to Dreamweaver CS4
2. PHP Object-Oriented Solutions
3. The Essential Guide to Dreamweaver CS3
4. PHP Solutions
5. Foundation ActionScript for Flash 8
6. Blog Design Solutions
7. Foundation PHP for Dreamweaver 8
8. Foundation PHP 5 for Flash
9. PHP Web Development with Dreamweaver MX 2004
10. Foundation Dreamweaver MX 2004

The list items now have alternating background colors—certainly a lot easier than adding the odd and even classes manually to each item, because the same code works however many items are in the list. In fact, it works for any list on a page. Also, by changing the selector from `li` to `tr`, you could easily apply this to a table with many rows.

7. Now let's wire up the link that toggles the display of coauthored books. Switch back to Code view, and add the following function definition inside the same `<script>` block as in step 4:

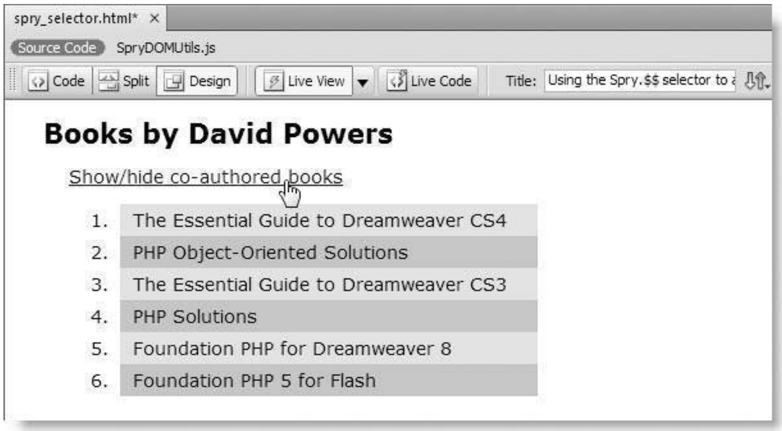
```
function showCoauthored()
{
  Spry.$$('li.coauthored').toggleClassName('hideMe');
}
```

This selects all `<li>` elements with the class `coauthored` and toggles the `hideMe` class on and off. As described in Table 8-3, the `toggleClassName()` method adds a class if it's absent and removes it if it's already applied to an element. So, this will have the effect of adding or removing a style rule that sets the element's `display` property to `none`.

8. Add it to the dummy link at the top of the page with the `onclick` attribute like this:

```
<p><a href="javascript:;" onclick="showCoauthored()">Show/hide
co-authored books</a></p>
```

9. Switch to Design view, and activate Live view. Click the link at the top of the page. The list of books should display only those books I wrote on my own, as shown in Figure 8-6.



Books by David Powers

Show/hide co-authored books

1. The Essential Guide to Dreamweaver CS4
2. PHP Object-Oriented Solutions
3. The Essential Guide to Dreamweaver CS3
4. PHP Solutions
5. Foundation PHP for Dreamweaver 8
6. Foundation PHP 5 for Flash

Figure 8-6. The contents of the list have been dynamically altered without needing to reload the page.

Click the link again, and the full list is restored.

10. There's one final improvement: the link should be visible only when JavaScript is enabled. Switch off Live view, and position your cursor inside the link at the top of the page. Select the <p> tag in the Tag inspector at the bottom of the Document window, and choose hideMe from the Class drop-down menu in the HTML view of the Property inspector. The link will disappear.

11. You want the link to be visible when JavaScript is enabled, so you can use the `Spry.$$` selector to remove the `hideMe` class. Amend the `init()` function like this:

```
function init()
{
  Spry.$$('li:nth-child(odd)').addClassName('odd');
  Spry.$$('li:nth-child(even)').addClassName('even');
  Spry.$$('p.hideMe').removeClassName('hideMe');
}
```

This removes the `hideMe` class from any paragraph that has the `hideMe` class.

12. Save and test the page again. Check your code, if necessary, against `spry_selector.html` in `examples/ch08`.

This has been only a brief example of what you can do with `SpryDOMUtils.js`, but I hope it will encourage you to experiment more. Working your way through the samples included with the Spry framework download should give you further ideas.

Reducing download times with smaller files

One drawback with using a JavaScript library is the size of the files. Spry effects make your pages livelier, but they add 77KB to the download size. That's quite a lot of code just to add one or two pleasing effects, particularly if some of your users are still on dial-up connections. Even if your target audience uses broadband, file size remains a consideration because bigger files consume more bandwidth, and on a popular site, that can cost you or your clients a lot of money.

However, it's not quite as bad as it sounds. JavaScript files are stored in the user's browser cache, so they are normally downloaded only the first time they are required. Still, if you're concerned about the size of the Spry external files, you can replace them with smaller versions. If you download the full Spry framework from Adobe Labs, as described earlier, the ZIP file contains two folders, `includes_minified` and `includes_packed`. These contain versions of the library files that have been compressed to reduce their size. The files have exactly the same names as the versions installed by Dreamweaver, so all you need to do is swap your existing files for ones of the same name from either `includes_minified` or `includes_packed`. The two folders use different techniques to reduce file size, but those in `includes_packed` are considerably smaller. To give just one example, the version of `SpryEffects.js` installed by Dreamweaver is 77KB, the one in `includes_minified` is 62KB, whereas the one in `includes_packed` is just 29KB. On a popular site, the bandwidth savings could be considerable.

Creating unobtrusive JavaScript

If implemented skillfully, CSS separates a page's content from instructions about how it should be presented. This has inspired many developers to apply the same principle to JavaScript, separating behavior from structure. "Wouldn't it be better," the argument goes, "to add JavaScript to a page only if the browser is capable of handling it?"

Since JavaScript lets you manipulate the DOM, you can. This is a technique known as **unobtrusive JavaScript**. Instead of embedding `onclick` and other event handling attributes in the HTML code, unobtrusive JavaScript uses DOM manipulation to add them on the fly in just the same way as the previous exercise added the `odd` and `even` classes to the list items.

The difficulty with unobtrusive JavaScript is that it requires a lot of careful planning. Because you can't see the features being added to the HTML code, you need to work out exactly how everything can be added dynamically.

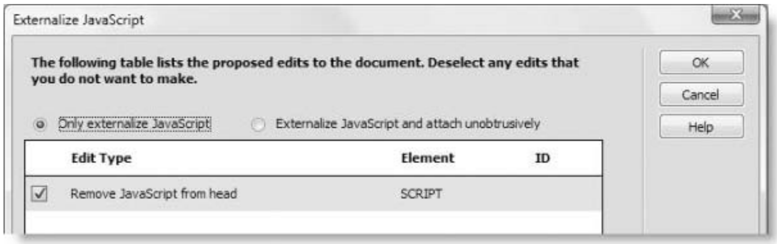
Using the JavaScript Extractor to externalize scripts

Dreamweaver CS4 has come up with a feature designed to take all the guesswork out of creating unobtrusive JavaScript: the JavaScript Extractor. This works on the simple principle that you embed the JavaScript elements in a page in the normal way. Once you're happy with the way the page works, you extract the JavaScript and externalize it. The drawback with this is that it's like squeezing toothpaste from a tube: it's easy to do, but don't try getting it back in afterward. . .

Moving JavaScript to an external file

This exercise demonstrates how to use the JavaScript Extractor using spry_selector.html from the preceding section.

1. Because the JavaScript Extractor cannot restore JavaScript once it has been removed from a page, it's always a good idea to create a new copy of the file that you want to work on. Save spry_selector.html from the previous exercise (or from examples/ch08) as spry_unobtrusive.html in workfiles/ch08.
2. Close the original file and work with spry_unobtrusive.html.
3. Select Commands ➤ Externalize JavaScript. Dreamweaver analyzes the page and opens the following dialog box:


Externalize JavaScript

The following table lists the proposed edits to the document. Deselect any edits that you do not want to make.

[✓ Only externalize JavaScript] Externalize JavaScript and attach unobtrusively

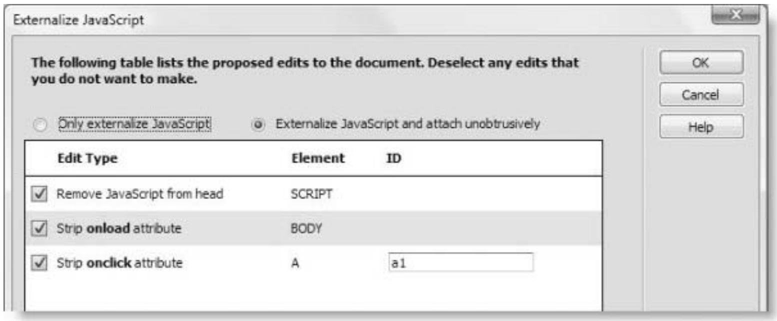
	Edit Type	Element	ID
✓	Remove JavaScript from head	SCRIPT	

The radio buttons at the top of the dialog box offer the following two options:

- Only externalize JavaScript: This simply moves functions to an external file and attaches the file to the page.
- Externalize JavaScript and attach unobtrusively: This attempts to move everything and creates the necessary external script to add inline event handlers, such as onclick, through DOM manipulation.

With the first option selected, Dreamweaver finds only the function definitions in the <head> of spry_unobtrusive.html.

4. Select the second radio button. Dreamweaver displays a warning that behaviors will no longer be editable through the Behaviors panel (this includes Spry effects). When you click OK to dismiss the warning, the Externalize JavaScript dialog box changes to this:


Externalize JavaScript

The following table lists the proposed edits to the document. Deselect any edits that you do not want to make.

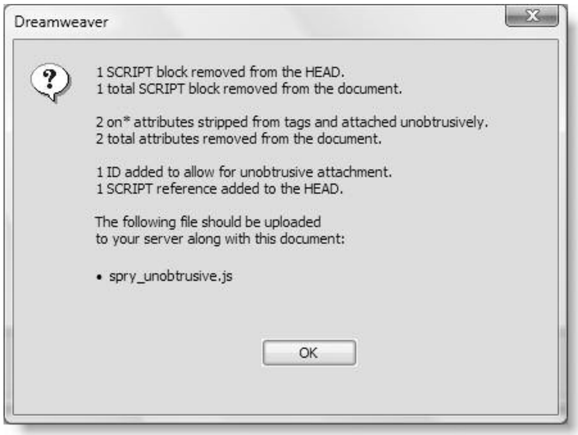
Only externalize JavaScript [✓ Externalize JavaScript and attach unobtrusively]

	Edit Type	Element	ID
✓	Remove JavaScript from head	SCRIPT	
✓	Strip **onload** attribute	BODY	
✓	Strip **onclick** attribute	A	a1

Dreamweaver lists all the JavaScript that it can find in the page. The checkbox alongside each proposed edit lets you decide whether to implement a particular suggestion. In this case, each edit is selected by default. However, Dreamweaver automatically deselects any scripts that use `document.write`, because these cannot be externalized.

To be able to manipulate the DOM, Dreamweaver automatically creates IDs for inline elements that don't already have them. As you can see in the preceding screenshot, it says it will add `a1` as the ID for the `onclick` attribute. If you want to change the ID, the field is editable.

5. Click OK when you're happy with your selections. Dreamweaver then presents you with a report of what it has done, like this:



```
1 SCRIPT block removed from the HEAD.
1 total SCRIPT block removed from the document.

2 on* attributes stripped from tags and attached unobtrusively.
2 total attributes removed from the document.

1 ID added to allow for unobtrusive attachment.
1 SCRIPT reference added to the HEAD.

The following file should be uploaded
to your server along with this document:

• spry_unobtrusive.js
```

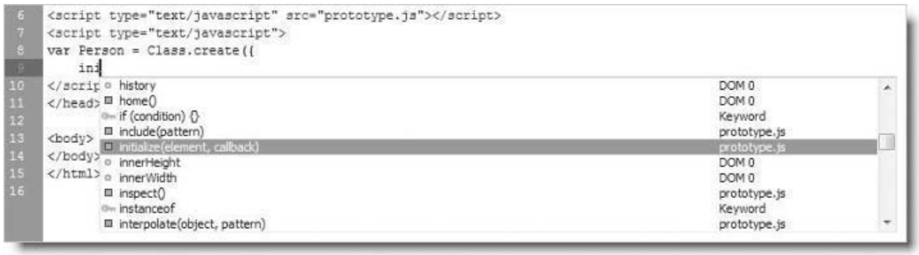
The important thing about this report is the last section, which tells you the name of the external JavaScript file that it has created. You must upload this to your website. Otherwise, none of the JavaScript will work.

The external file is given the same name as the file you have just extracted the JavaScript from, except with a `.js` filename extension. If a file with that name already exists, Dreamweaver adds a number just before the filename extension.

The external JavaScript file is created in the same folder, but you can move it to a dedicated scripts folder through the Files panel. If you move the file, don't forget to update the links when Dreamweaver prompts you.

Using other JavaScript libraries

Adobe realizes that not everyone will want to use Spry, so support for all flavors of JavaScript has been greatly improved in Dreamweaver CS4. As explained in Chapter 1, Dreamweaver now provides code hints for all the main data types and the DOM. More significantly, Dreamweaver constantly analyzes the JavaScript attached to a page, providing code hints for custom functions and classes. This includes popular JavaScript frameworks, such as Prototype and jQuery, as shown in Figure 8-7.



```
<script type="text/javascript" src="prototype.js"></script>
<script type="text/javascript">
var Person = Class.create({
    ini
</scrip
</head>

<body>
</body>
</html>
```

```
history                        DOM 0
home()                         DOM 0
if (condition) {}              Keyword
include(pattern)               prototype.js
initialize(element, callback)  prototype.js
innerHeight                    DOM 0
innerWidth                     DOM 0
inspect()                      prototype.js
instanceof                     Keyword
interpolate(object, pattern)   prototype.js
```

Figure 8-7. Dreamweaver's code introspection provides code hints for Prototype and other JavaScript libraries.

Code hints normally pop up when you type a period or opening parenthesis. You can also trigger them by pressing Ctrl+spacebar (the combination is the same on Windows and Mac).

Code hints generated by code introspection are available only in pages that are directly attached to the file that contains the function or class definition. For example, if you attach `prototype.js` to a page, you get Prototype hints in that page. However, if you attach an external file to the same page, you don't get any Prototype hints in the external file. Spry's code hints, on the other hand, are hardwired into Dreamweaver, so they're available in any page. Consequently, if you want to use JavaScript libraries other than Spry—and you want code hints—you need to attach the external library directly to the page where you create your JavaScript. The simple way to do this is to build your JavaScript in the <head> of the page and then use the JavaScript Extractor, as described in the previous section, to export it to an external file. That's how I created `jquery_selector.html` and `jquery_selector.js` in `examples/ch08`.

The other drawback with external libraries is that the level of hinting is determined by the structure of those libraries. Although you get hints for all the methods available to a `Spry.$$` selector, similar hints are not generated for the Prototype $$ or jQuery $ selectors. Let's hope this situation will be improved either in a future version of Dreamweaver or by the release of a third-party extension to provide code hints for all the main frameworks.

Talking of third-party extensions, perhaps the best support of all for other JavaScript libraries comes through Adobe's decision to release the Web Widgets Software Development Kit (SDK). This enables JavaScript developers to package web widgets as Dreamweaver extensions. Prior to the release of Dreamweaver CS4, Adobe contacted the teams behind jQuery (`http://jquery.com/`) and the Yahoo! User Interface (YUI) Library (`http://developer.yahoo.com/yui/`) and asked them to adapt some of their widgets so they can be easily installed in Dreamweaver. Other leading developers are also being encouraged to package JavaScript widgets for Dreamweaver. To find out what widgets are available, open the Extend Dreamweaver control on the Application bar, as shown in Figure 8-8, and select Browse for Web Widgets.

8

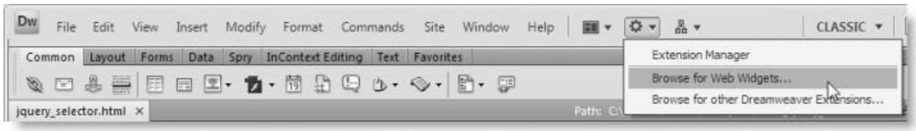

Dw
File Edit View Insert Modify Format Commands Site Window Help
CLASSIC
Common Layout Forms Data Spry InContext Editing Text Favorites
Extension Manager
Browse for Web Widgets...
Browse for other Dreamweaver Extensions...
jquery_selector.html
Path:

Figure 8-8. The Extend Dreamweaver control on the Application bar is your gateway to JavaScript widgets.

As long as you're connected to the Internet, this takes you directly to a dedicated web widget section on the Adobe Exchange. Choose the widgets you want, download, and install them.

The next section walks you through the installation process for all Dreamweaver extensions. Then, to round out the chapter, I'll show you how to use two of the new web widgets, the jQuery Dialog and the YUI calendar.

Installing Dreamweaver extensions

One of the main reasons for Dreamweaver's enduring dominance as the leading website development program is its extensibility. Extensions created by third-party developers add new functionality to the program. Some extensions are quite simple. Others are much more powerful and are designed to take your productivity to a whole new level. For example, Cartweaver (`http://www.cartweaver.com`), the PHP version of which was created by my partner in crime on this book, Tom Muck, greatly simplifies the construction of a fully featured ecommerce site. The following is a short—and by no means exhaustive—list of some of the most respected third-party developers (the more sophisticated extensions, such as Cartweaver, are sold on a commercial basis, but many others are free):

- Community MX (`http://communitymx.com/`)
- DMXzone (`http://dmxzone.com/`)
- Kaosweaver (`http://kaosweaver.com/`)
- Project Seven (`http://www.projectseven.com`)
- Tom Muck (`http://tom-muck.com/`)

Adobe has also taken the decision to focus some aspects of Dreamweaver functionality in extensions, rather than make them part of the core product. This makes it easier to update that functionality between releases of the program itself. So, you're likely to see more extensions in the future.

Regardless of whether an extension is free or commercial, the method of installation is identical and is done through the Adobe Extension Manager.

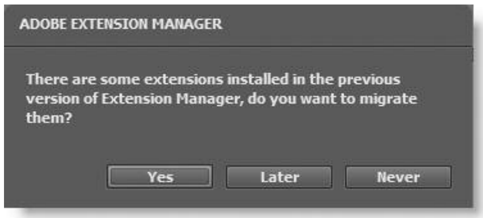
Using the Adobe Extension Manager

In previous versions of Dreamweaver, the Extension Manager was installed automatically. However, the CS4 installer now gives you the option not to install many of the shared programs, such as Device Central, Bridge, and the Extension Manager. If you accepted the default selection of programs when installing Dreamweaver CS4, you should see Adobe Extension Manager CS4 listed among the programs in the Windows Start menu or in your `Applications` folder on a Mac. If it's not there, you need to install it from your Dreamweaver or Creative Suite DVD. You should also ensure that you have the Adobe Integrated Runtime (AIR) installed, because the Extension Manager is now an AIR application (AIR is included in the default Dreamweaver installation).

You can launch the Extension Manager in several ways, but perhaps the quickest way is by selecting Extension Manager in the Extend Dreamweaver control on the Application bar (see Figure 8-8). If you have hidden the Application bar on a Mac, alternative ways of opening the Extension Manager are by selecting Commands ➤ Manage Extensions or Help ➤ Manage Extensions. You can also open the program directly from the Windows Start menu or the `Applications` folder on a Mac. As if that weren't enough, you can usually also launch the Extension Manager by double-clicking the `.mxp` file of the extension you want to install.

Migrating extensions from a previous version

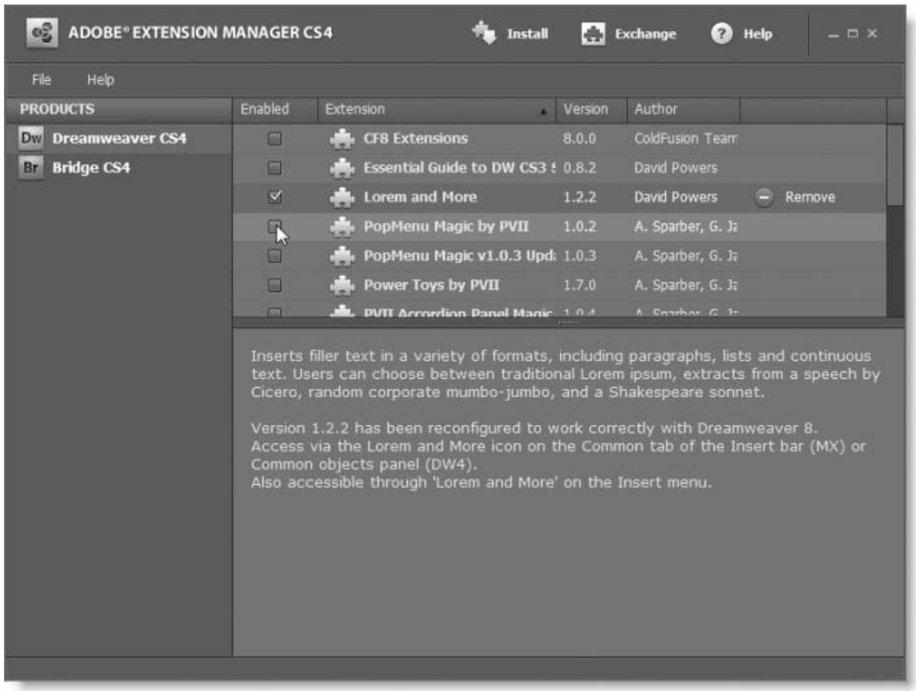
If you're upgrading from an earlier version of Dreamweaver, you'll immediately notice that the Extension Manager looks completely different. However, most of its functionality is unchanged. The first time you launch the Extension Manager, it detects any extensions installed in a previous version of Dreamweaver on the same computer account and presents you with the following options:


ADOBE EXTENSION MANAGER

There are some extensions installed in the previous version of Extension Manager, do you want to migrate them?

If you click Yes, the Extension Manager copies details of existing extensions to your CS4 configuration folder. It then tells you to relaunch the Extension Manager. Migrating extensions like this does not automatically enable them in CS4. You need to do that manually for each one, because some older extensions might not be compatible. However, it's a useful way to preserve functionality between versions.

To enable an extension, put a check mark in the Enabled checkbox to the left of the extension name, as shown in Figure 8-9. Some extensions require you to restart Dreamweaver, but you don't need to do so until you have selected all those you want to migrate. However, it's a wise policy to install extensions only one at a time, because this makes it easier to detect which extension is responsible if Dreamweaver starts behaving erratically. Sometimes changes to Dreamweaver make older extensions incompatible with the latest version.



Enabled	Extension	Version	Author
☐	CF8 Extensions	8.0.0	ColdFusion Tean
☐	Essential Guide to DW CS3 !	0.8.2	David Powers
✓	Lorem and More	1.2.2	David Powers
☐	PopMenu Magic by PVII	1.0.2	A. Sparber, G. Ja
☐	PopMenu Magic v1.0.3 Upd:	1.0.3	A. Sparber, G. Ja
☐	Power Toys by PVII	1.7.0	A. Sparber, G. Ja
☐	PVII Accordion Panel Magic	1.0.4	A. Sparber, G. Ja

Inserts filler text in a variety of formats, including paragraphs, lists and continuous text. Users can choose between traditional Lorem ipsum, extracts from a speech by Cicero, random corporate mumbo-jumbo, and a Shakespeare sonnet.

Version 1.2.2 has been reconfigured to work correctly with Dreamweaver 8.
Access via the Lorem and More icon on the Common tab of the Insert bar (MX) or Common objects panel (DW4).
Also accessible through 'Lorem and More' on the Insert menu.

Figure 8-9. The Extension Manager provides a simple interface to add and remove Dreamweaver extensions.

Installing an extension

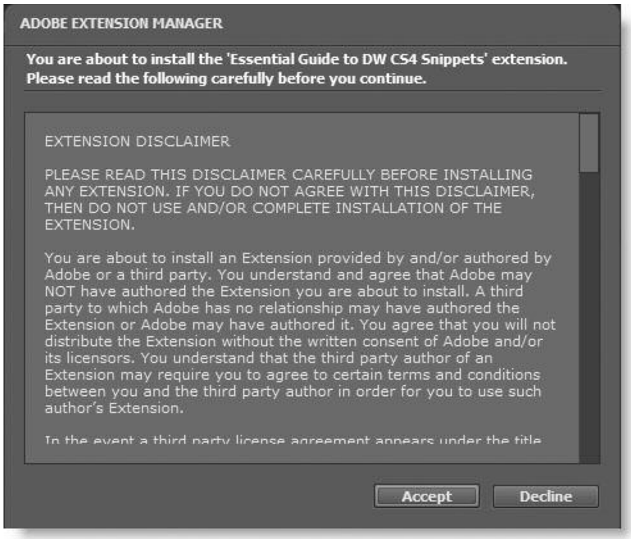
The Extension Manager is shared by several Creative Suite programs, so it's important to check that you have the correct program selected in the Products column on the left of the Extension Manager (see Figure 8-9). If you launched the Extension Manager from Dreamweaver CS4, it should automatically select the correct program.

Unlike previous versions, Extension Manager CS4 cannot be used to manage extensions in older programs. It recognizes only programs in Creative Suite 4.

Installing an extension involves the following simple steps:

1. Click the Install button at the top of the Extension Manager.
2. In the Select Extension to Install dialog box, navigate to the folder where you downloaded the extension, select the extension's `.mxp` file, and click Open.

3. You're then presented with a disclaimer notice that tells you Adobe does not offer technical support for the extension and that any license is between you and its creator. In addition to a standard disclaimer that applies to all extensions, there might also be a license specific to the extension. You must click Accept to proceed with the installation.



You are about to install the 'Essential Guide to DW CS4 Snippets' extension.
Please read the following carefully before you continue.

EXTENSION DISCLAIMER

PLEASE READ THIS DISCLAIMER CAREFULLY BEFORE INSTALLING ANY EXTENSION. IF YOU DO NOT AGREE WITH THIS DISCLAIMER, THEN DO NOT USE AND/OR COMPLETE INSTALLATION OF THE EXTENSION.

You are about to install an Extension provided by and/or authored by Adobe or a third party. You understand and agree that Adobe may NOT have authored the Extension you are about to install. A third party to which Adobe has no relationship may have authored the Extension or Adobe may have authored it. You agree that you will not distribute the Extension without the written consent of Adobe and/or its licensors. You understand that the third party author of an Extension may require you to agree to certain terms and conditions between you and the third party author in order for you to use such author's Extension.

In the event a third party license agreement appears under the title

4. The extension now installs. Small extensions install almost instantaneously. Larger ones may take several minutes. Extensions created by the same developer often share common files, so you might see warnings that an older or newer version of a particular file already exists. Click Yes to replace older versions and No if the existing version is newer.

   When the process is complete, the Extension Manager will tell you whether you need to restart Dreamweaver. This usually happens with extensions that need to rebuild part of the menu system. The pane at the bottom of the Extension Manager provides a brief description of the extension and how to use it (see Figure 8-9).

Some commercial extensions require registration or activation. Follow the instructions onscreen the first time you launch Dreamweaver after installing such an extension.

Removing an extension

Removing an extension is easy. Just launch the Extension Manager, and click the Remove button alongside the name of the extension you want to remove (see Figure 8-9).

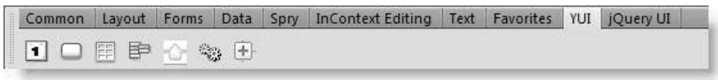
If you don't want to remove an extension completely from Dreamweaver, disable it temporarily by deselecting the Enabled checkbox alongside the extension name. Just select the checkbox again when you want to restore the extension.

Extensions install files in your personal configuration folder, so they are visible only to the current user account. If there is more than one user account on the computer, the extension needs to be installed separately in each one. Because extensions make changes to your configuration files, you should install extensions only from sources that you can trust.

Right, after that brief detour, let's get on with the jQuery and YUI web widgets.

Using jQuery and YUI web widgets

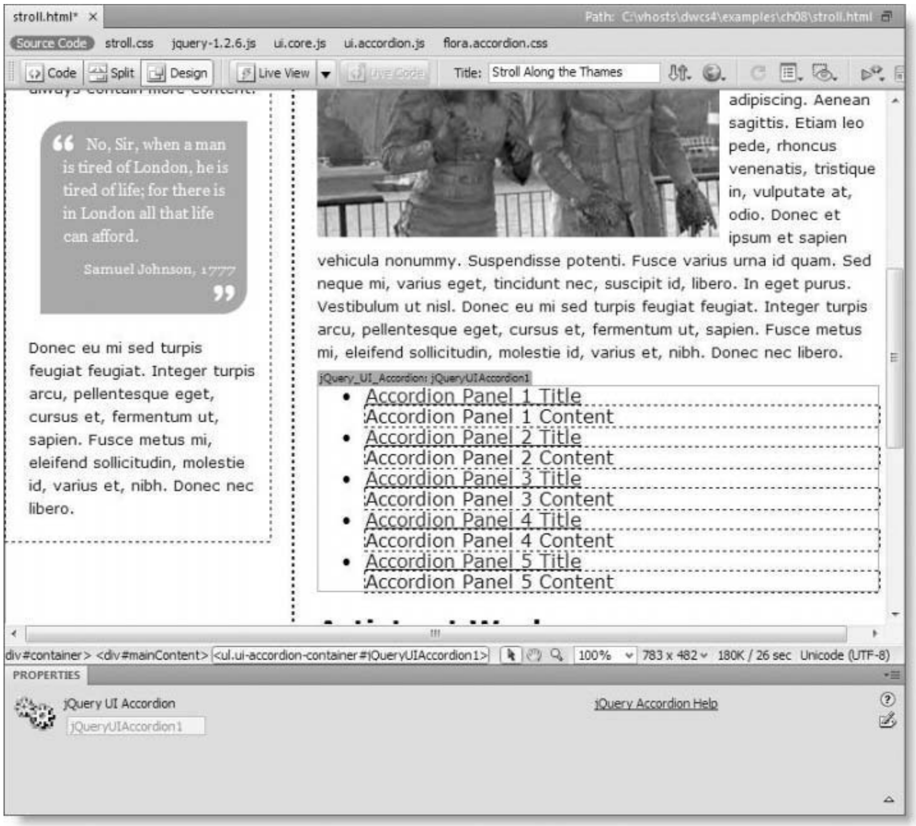
After downloading the extensions from the Adobe Exchange and installing them as described in the previous section, you need to restart Dreamweaver. The jQuery and YUI web widgets are then accessible through their own tabs on the Insert bar, as shown in the following screenshot, or submenus added at the bottom of the Insert menu. The icons and menu listings appear in the same order as you install each widget.



Both jQuery and YUI have packaged several of their best widgets for Dreamweaver, including calendars and sliders. The jQuery collection also includes an accordion and tabbed panels, which you might want to use in preference to the Spry versions described in Chapter 7, particularly if you're already at home with jQuery and want to incorporate other jQuery features into the widgets. A quick look at the jQuery accordion demonstrates the difference between the Spry widgets that are a core part of Dreamweaver CS4 and the third-party widgets.

To install a web widget, just position your cursor where you want to insert the widget, and click its icon on the Insert bar or select it from the Insert menu. Figure 8-10 shows a default jQuery UI Accordion widget inserted in `stroll.html`, the sample page that I showed you how to create in Chapter 5.

As you can see from the files listed in the Related Files toolbar in Figure 8-10, the jQuery accordion widget comes complete with three external JavaScript files, including the basic jQuery library, and a style sheet. However, no styles are applied in Design view, and the Property inspector simply has a link to online help. To see what the widget will look like when the page is deployed on the Web and to use the Code Navigator to inspect the widget's CSS, you need to turn on Live view (see Figure 8-11).



“No, Sir, when a man is tired of London, he is tired of life; for there is in London all that life can afford.
Samuel Johnson, 1777”

Donec eu mi sed turpis feugiat feugiat. Integer turpis arcu, pellentesque eget, cursus et, fermentum ut, sapien. Fusce metus mi, eleifend sollicitudin, molestie id, varius et, nibh. Donec nec libero.

adipiscing. Aenean sagittis. Etiam leo pede, rhoncus venenatis, tristique in, vulputate at, odio. Donec et ipsum et sapien vehicula nonummy. Suspendisse potenti. Fusce varius urna id quam. Sed neque mi, varius eget, tincidunt nec, suscipit id, libero. In eget purus. Vestibulum ut nisl. Donec eu mi sed turpis feugiat feugiat. Integer turpis arcu, pellentesque eget, cursus et, fermentum ut, sapien. Fusce metus mi, eleifend sollicitudin, molestie id, varius et, nibh. Donec nec libero.

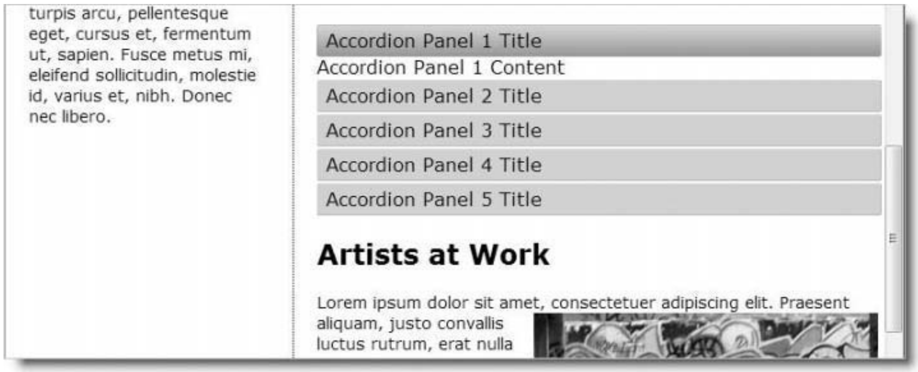
jQuery_UI_Accordion: jQueryUIAccordion1

- Accordion Panel 1 Title
  Accordion Panel 1 Content
- Accordion Panel 2 Title
  Accordion Panel 2 Content
- Accordion Panel 3 Title
  Accordion Panel 3 Content
- Accordion Panel 4 Title
  Accordion Panel 4 Content
- Accordion Panel 5 Title
  Accordion Panel 5 Content

div#container> <div#mainContent> <ul.ui-accordion-container#jQueryUIAccordion1> 100% 783 x 482 180K / 26 sec Unicode (UTF-8)

jQuery UI Accordion
jQueryUIAccordion1

Figure 8-10. The jQuery UI accordion widget appears unstyled in Design view.


turpis arcu, pellentesque eget, cursus et, fermentum ut, sapien. Fusce metus mi, eleifend sollicitudin, molestie id, varius et, nibh. Donec nec libero.

Accordion Panel 1 Title
Accordion Panel 1 Content
Accordion Panel 2 Title
Accordion Panel 3 Title
Accordion Panel 4 Title
Accordion Panel 5 Title

Artists at Work

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Praesent aliquam, justo convallis luctus rutrum, erat nulla

Figure 8-11. Turn on Live view to see what third-party widgets will look like in the finished page.

In spite of the lack of styling in Design view, using one of these web widgets is a huge time-saver. All the necessary files are attached and stored in a dedicated jQuery or YUI folder ready to be uploaded to your website. Inserting a widget also creates the necessary code to initialize it. However, instead of placing the initialization script at the bottom of the page, as Spry does, the third-party widgets insert it immediately after the HTML portion of the widget. Selecting the turquoise tab at the top-left of the widget and pressing Delete removes the widget, its contents, the initialization script, and all links to dependent files.

Adding content to the jQuery accordion is simply a matter of substituting the placeholder text, so it's one of the easiest third-party widgets to use. Other widgets require a knowledge of jQuery or the YUI Library API. Using jQuery and the YUI Library API is beyond the scope of this book, but the following sections give you a brief taster of what's possible. If you have a basic understanding of JavaScript, it doesn't take long to achieve impressive results.

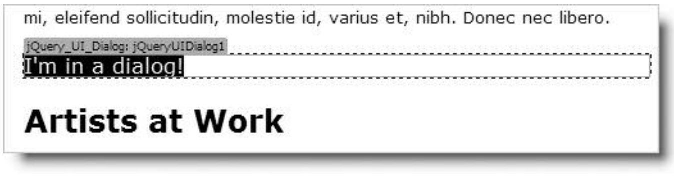
Inserting a jQuery UI dialog widget

The jQuery UI dialog widget (`http://docs.jquery.com/UI/Dialog`) creates modeless and modal floating windows and dialog boxes. A **modeless** window is a pop-up window **that** permits access to the originating page, whereas a **modal** one blocks access until the pop-up window is closed. In combination with a modal window, the dialog widget makes it easy to dim the rest of the page so that the user's concentration is focused on the content of the pop-up—a technique that has become popular with image galleries (see Figure 8-13).

Displaying a larger image with a dialog widget

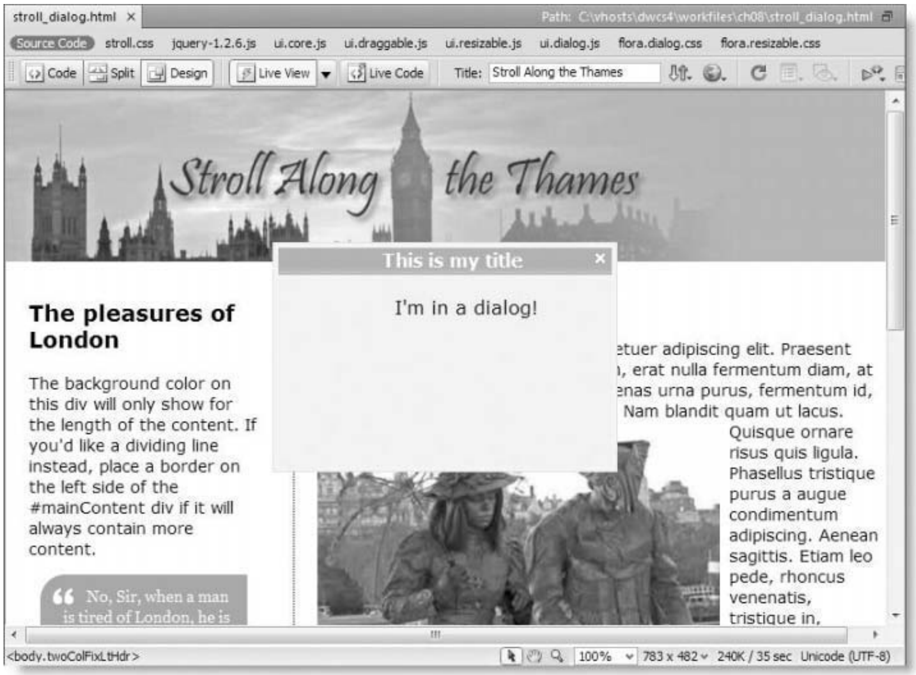
The following exercise uses the jQuery UI dialog widget to display a larger version of `living_statues.jpg` in `stroll.html`. Initially, the widget will be physically inserted into the page, but it will then be converted to use unobtrusive JavaScript so the page degrades gracefully in browsers that have JavaScript turned off. The exercise uses some basic jQuery techniques, but you should be able to follow the instructions even if you have never used jQuery before.

1. Copy `stroll.html` from `examples/ch08`, and save it as `stroll_dialog.html` in `workfiles/ch08`. Also copy `stroll.css` to the same folder.
2. Position your cursor at the end of the first paragraph, just before the Artists at Work heading. Insert a jQuery UI dialog widget from the Insert bar or Insert menu. A widget with some placeholder text is inserted in the page like this:


mi, eleifend sollicitudin, molestie id, varius et, nibh. Donec nec libero.
jQuery_UI_Dialog: jQueryUIDialog1
I'm in a dialog!
Artists at Work

3. Save `stroll_dialog.html`. Dreamweaver presents you with a dialog box informing you that it's copying dependent files to your site. These are all located in a dedicated folder called `jQuery.ui-1.5.2` in the site root (the name of the folder is likely to change when new versions are released).

4. Click the Live View button or load the page into a browser to view the default dialog widget (see Figure 8-12). The dialog box loads immediately. It's both resizable and draggable, and it closes when you click the close button at the top-right of the dialog box. It's not very practicable in its default state, but it doesn't take much effort to change.



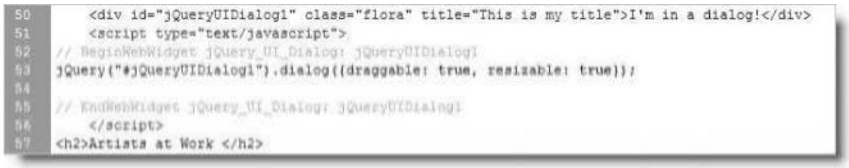
The background color on this div will only show for the length of the content. If you'd like a dividing line instead, place a border on the left side of the #mainContent div if it will always contain more content.

No, Sir, when a man is tired of London, he is

...etuer adipiscing elit. Praesent ..., erat nulla fermentum diam, at ...enas urna purus, fermentum id, Nam blandit quam ut lacus. Quisque ornare risus quis ligula. Phasellus tristique purus a augue condimentum adipiscing. Aenean sagittis. Etiam leo pede, rhoncus venenatis, tristique in,

Figure 8-12. The default widget displays a dialog box in the center of the page as soon as it loads.

5. Close the dialog box, and deactivate Live view. Switch to Code view to examine the code inserted by the widget. It's just above the second heading and looks like this:



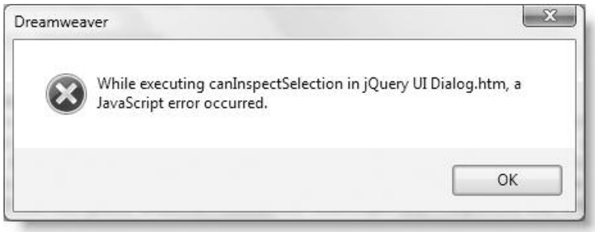
```
    <div id="jQueryUIDialog1" class="flora" title="This is my title">I'm in a dialog!</div>
    <script type="text/javascript">
// BeginWebWidget jQuery_UI_Dialog: jQueryUIDialog1
jQuery("#jQueryUIDialog1").dialog({draggable: true, resizable: true});

// EndWebWidget jQuery_UI_Dialog: jQueryUIDialog1
    </script>
<h2>Artists at Work </h2>
```

As you can see, the dialog box is simply a `<div>`. The text in the dialog box title bar is taken from the `title` attribute of the `<div>`, and the content of the `<div>` determines what is displayed inside the dialog box.

The code shown on line 53 of the preceding screenshot initializes the widget. To avoid conflicts with other JavaScript libraries, it uses the `jQuery()` function instead of the shorthand `$()` notation.

6. You're going to use the dialog box to display a larger version of `living_statues.jpg`, so replace the `title` attribute shown on line 50 with Living Statues on the South Bank.
7. Delete the placeholder text between the `<div>` tags, and with your cursor between the empty tags insert `living_statues_680.jpg` from the `images` folder. Add some alternative text when prompted to do so.
8. Enclose the entire `<div>` in *single* quotes, cut it to your clipboard, and paste it as the argument to the jQuery function in place of `"jQueryUIDialog1"`. You might see the following warning when you try to select the code, but you can safely ignore it:



While executing canInspectSelection in jQuery UI Dialog.htm, a JavaScript error occurred.

The code inside the `<script>` block should now look like this:

```
// BeginWebWidget jQuery_UI_Dialog: jQueryUIDialog1
jQuery('<div id="jQueryUIDialog1" class="flora" title="Living Statues ➥
on the South Bank"><img src="../../images/living_statues_680.jpg" ➥
width="680" height="449" alt="Living Statues" /></div>').dialog( ➥
{draggable: true, resizable: true});
// EndWebWidget jQuery_UI_Dialog: jQueryUIDialog1
```

9. If you save the page and test it now, the dialog box still appears immediately. It remains the same size, but you can resize it to see the larger image. By using the code for the `<div>` as the argument to `jQuery()`, the `<div>` and its contents are now being generated on the fly by JavaScript. This means the larger image won't be loaded in a browser that has JavaScript disabled.
10. The jQuery UI `dialog()` constructor method takes an object literal containing the options you want to set. At the moment, the `options` object has two properties: `draggable` and `resizable`, both of which are set to `true`. Let's set two more, `height` and `width`, so the image fits the dialog box. Amend the object literal like this:

```
{draggable: true,
 resizable: true,
 height: 515,
 width: 720}
```

Although adding newlines to JavaScript statements usually causes them to malfunction, you can use newlines in objects for ease of reading without causing problems.

11. To make the dialog box modal, all you need to do is add `modal: true` to the `options` object like this:

```
{draggable: true,
 resizable: true,
 height: 515,
 width: 720,
 modal:true}
```

12. To dim the background, you also need to use the `overlay` property, which expects its values as an object, so you nest it within the `options` object like this:

```
{draggable: true,
 resizable: true,
 height: 515,
 width: 720,
 modal:true,
 overlay: {
   opacity: 0.5,
   background: 'black'
 }
}
```

13. Test the page to make sure everything is working as expected. You should see the larger image displayed fully inside a modal dialog box, with the rest of the window dimmed (see Figure 8-13 on the next page).

14. To prevent the dialog box from opening automatically when the page loads, you need to set the `autoOpen` property of the `options` object to `false`. You also need a reference to the dialog box so that it can be opened when the user clicks the smaller image. Add the `autoOpen` property, and assign the whole declaration to a variable called `bigImage`. The complete code should look like this:

```
var bigImage = jQuery('<div id="jQueryUIDialog1" class="flora" ➥
title="Living Statues on the South Bank"><img ➥
src="../../images/living_statues_680.jpg" width="680" height="449" ➥
alt="Living Statues" /></div>').dialog({
  draggable: true,
  resizable: true,
  height: 515,
  width: 720,
  modal: true,
  overlay: {
    opacity: 0.5,
    background: 'black'
  },
  autoOpen:false
});
```

15. You can now attach an `onclick` event handler dynamically to the smaller image, which can be identified using the following attribute selector:

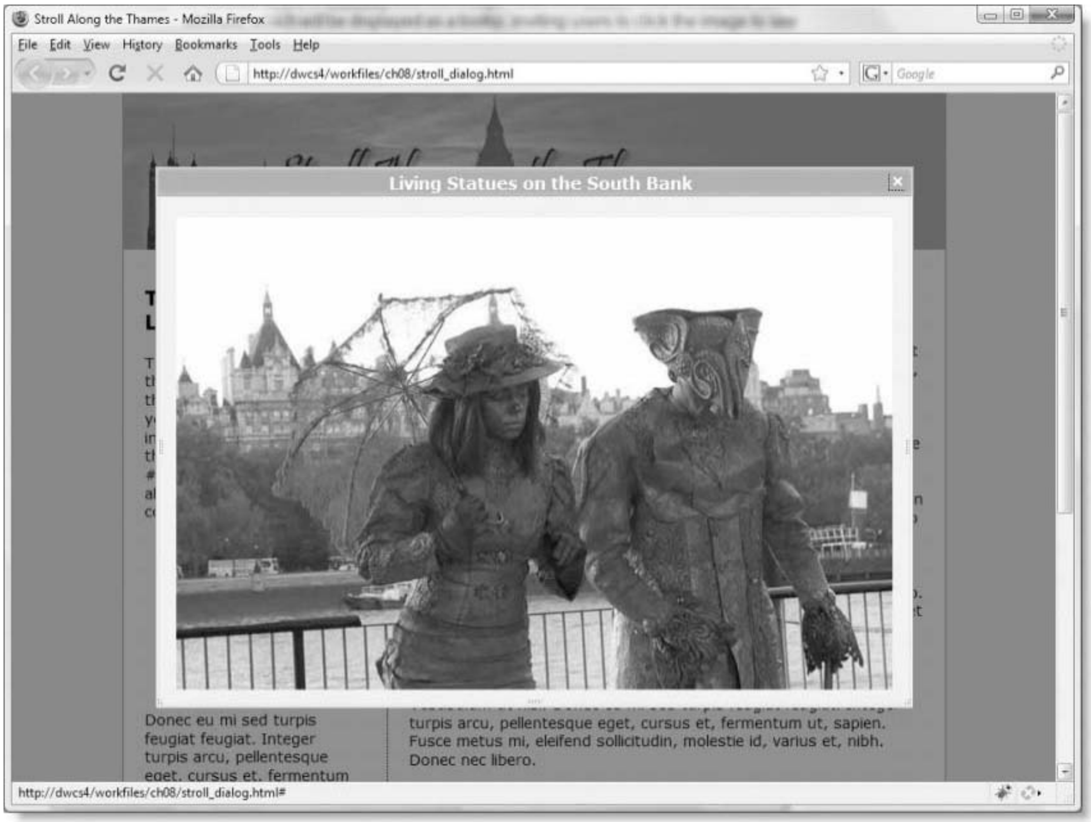
```
jQuery('img[src$=living_statues.jpg]')
```

This looks for an image with a `src` attribute that ends with `living_statues.jpg`. Add the following code immediately after the code in step 14:

```
jQuery('img[src$=living_statues.jpg]').css('cursor', 'pointer')
.attr('title', 'Click for a larger image')
.click(function(e){bigImage.dialog('open')});
```

In typical jQuery fashion, this chains several methods and applies them to `living_statues.jpg`. First, the `css()` method converts the cursor to a hand pointer whenever anyone mouses over the image. Then the `attr()` method adds a `title` attribute, which will be displayed as a tooltip, inviting users to click the image to see a larger version. Finally, the `click()` method is passed a function that references the dialog box using the variable `bigImage` and passes `'open'` as an argument to its `dialog()` method.

16. Save `stroll_dialog.html`, and test it. When you mouse over `living_statues.jpg`, the cursor should turn to a hand and display a tooltip inviting you to view a larger image. Click, and you should see a much bigger version centered in a dialog box with the rest of the window dimmed, as shown in Figure 8-13.


Stroll Along the Thames - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://dwcs4/workfiles/ch08/stroll_dialog.html
Google
Living Statues on the South Bank
http://dwcs4/workfiles/ch08/stroll_dialog.html#

Donec eu mi sed turpis feugiat feugiat. Integer turpis arcu, pellentesque eget, cursus et, fermentum

turpis arcu, pellentesque eget, cursus et, fermentum ut, sapien. Fusce metus mi, eleifend sollicitudin, molestie id, varius et, nibh. Donec nec libero.

Figure 8-13. The dialog widget displays the larger image and dims the rest of the page.

17. Finally, to tidy up the page and remove the JavaScript from the middle of the HTML, cut the script block and paste it into the <head> of the document after the links to the jQuery external files (or put it in an external file of its own, linked to the page after the other jQuery files). Once you move the script outside the body of the page, you need to wrap the script in a jQuery document ready handler like this:

```
jQuery(function() {
  var bigImage = jQuery('<div id="jQueryUIDialog1" class="flora" ➥
title="Living Statues on the South Bank"><img ➥
src="../../images/living_statues_680.jpg" width="680" height="449" ➥
alt="Living Statues" /></div>').dialog({
    draggable: true,
    resizable: true,
    height: 515,
    width: 720,
    modal: true,
    overlay: {
      opacity: 0.5,
      background: 'black'
    },
    autoOpen:false
  });
  jQuery('img[src$=living_statues.jpg]').css('cursor', 'pointer')
  .attr('title', 'Click for a larger image')
  .click(function(e){bigImage.dialog('open')});
});
```

I have used `jQuery()` instead of the shorthand `$()`, but you can use `$()` if you're not mixing jQuery with other JavaScript libraries that use the same shorthand.

Check your code, if necessary, against `stroll_dialog.html` in `examples/ch08`.

Selecting dates with a YUI calendar

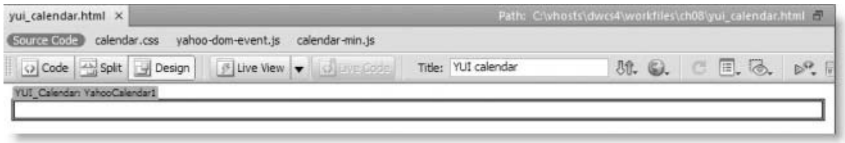
The YUI Library is a massive collection of utilities, controls, and components written in JavaScript. Just to give you a taste of the type of things available, I have chosen the YUI Calendar, which is one of the first web widgets to have been released for Dreamweaver. Inserting a calendar requires nothing more than clicking its icon in the YUI tab of the Insert bar or selecting it from the Insert menu and saving the external files to your site. However, you need to write your own JavaScript functions to do anything with selected dates.

Displaying the selected date

This exercise shows how to capture the date selected in a YUI calendar and display it as a JavaScript alert.

1. Create a new page called `yui_calendar.html` in `workfiles/ch08`, and insert a YUI Calendar widget from the Insert bar or Insert menu.

2. Save the page to copy the external JavaScript files and style sheet to your site. Dreamweaver stores them in a dedicated folder called YUI.
3. When you look at the page in Design view, you might be distinctly underwhelmed, because all you get is a turquoise border and tab with nothing inside.



4. Click the Live View button, and everything comes to life, with the current month and date selected, as shown in Figure 8-14 (so now you know when I wrote this part of the book). The calendar is fully functional in the sense that you can move back and forth through the months and select dates, but nothing happens when you select a particular date. It's up to you to add that functionality yourself.

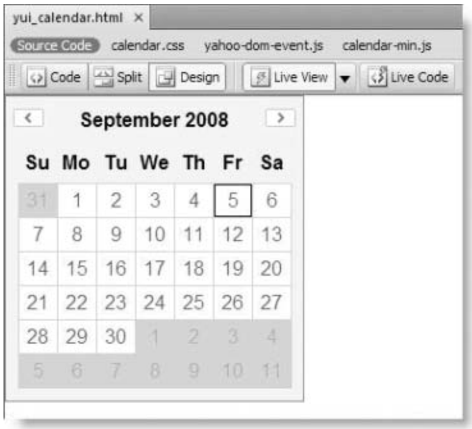
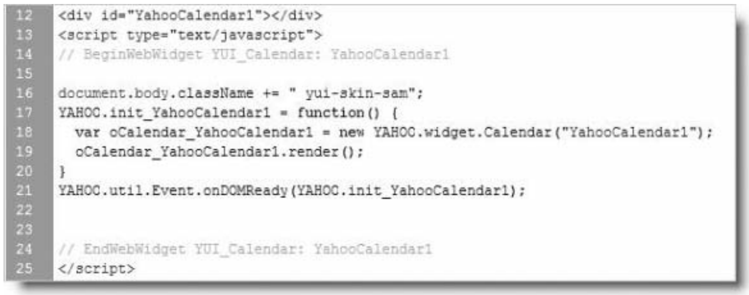


Figure 8-14. The YUI calendar is generated entirely dynamically by JavaScript.

5. Deactivate Live view, and switch to Code view. As you can see in the following screenshot, the calendar is an empty `<div>`, and there are just a few lines of script. The code shown on lines 17–20 initializes the calendar, assigning it to a variable called `oCalendar_YahooCalendar1`. The code on line 21 loads the calendar into the page when the DOM is ready.



```
12  <div id="YahooCalendar1"></div>
13  <script type="text/javascript">
14  // BeginWebWidget YUI_Calendar: YahooCalendar1
15
16  document.body.className += " yui-skin-sam";
17  YAHOO.init_YahooCalendar1 = function() {
18    var oCalendar_YahooCalendar1 = new YAHOO.widget.Calendar("YahooCalendar1");
19    oCalendar_YahooCalendar1.render();
20  }
21  YAHOO.util.Event.onDOMReady(YAHOO.init_YahooCalendar1);
22
23
24  // EndWebWidget YUI_Calendar: YahooCalendar1
25  </script>
```

6. When you select one or more dates in the calendar, it dispatches an event called selectEvent, which contains the selected date(s) as a multidimensional array in the format [[YYYY, MM, DD], [YYYY, MM, DD] . . .]. So, you can define an event handler function to capture the selection. You need to add it inside the initialization function like this:

```
YAHOO.init_YahooCalendar1 = function() {
  function selectHandler(type, args, obj)
  {
    var dates = args[0];
    var date = dates[0];
    var months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'June', 'July', ➥
'Aug', 'Sep', 'Oct', 'Nov', 'Dec'];
    var year = date[0], month = months[date[1]-1], day = date[2];
    alert('Selected date is : ' +  month + ' ' + day + ', ' + year);
  }
  var oCalendar_YahooCalendar1 = new YAHOO.widget.Calendar( ➥
"YahooCalendar1");
  oCalendar_YahooCalendar1.render();
}
```

The event handler needs to take three arguments: the type of event, the arguments dispatched by the event, and the object that was the event's target. The function needs the first and third arguments to know what to expect, but all you're interested in is extracting the value of the arguments passed by the event.

The selectEvent dispatches a single multidimensional array of dates, so there's only one argument, which can be extracted as args[0] and is assigned to a variable called dates.

For the purposes of this exercise, you want to extract just the first date in the dates array. This can be identified as dates[0] and is assigned to a variable called date.

Since each date is in itself an array in the format [YYYY, MM, DD], you can extract the day as date[2], the month as date[1], and the year as date[0].

To avoid confusion with different national conventions regarding date formats, I have created an array of month names. JavaScript counts arrays from zero, so you get the month name by subtracting one from the month number like this: months[date[1]-1].

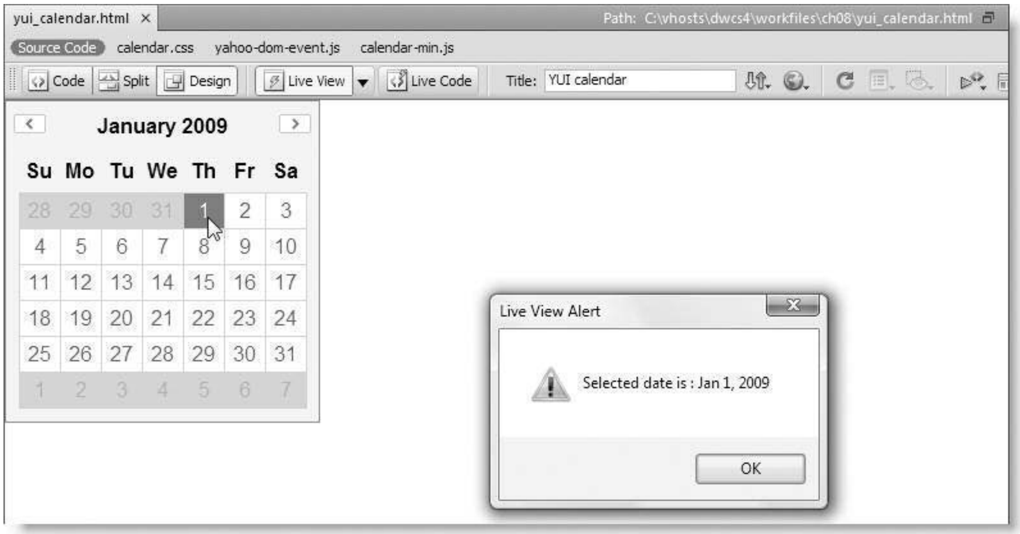
Finally, the function passes the selected date to an alert.

7. The event handler function needs to be registered to listen for the selectEvent by using the subscribe() method after the calendar object has been instantiated like this:

```
var oCalendar_YahooCalendar1 = new YAHOO.widget.Calendar( ➥
"YahooCalendar1");
oCalendar_YahooCalendar1.selectEvent.subscribe(selectHandler, ➥
oCalendar_YahooCalendar1, true);
oCalendar_YahooCalendar1.render();
```

The `subscribe()` method takes three arguments: the event handler function, the object, and the Boolean variable `true`.

8. Save `yui_calendar.html`, and test it in Live view or a browser. Select a date in the calendar, and you should see its value displayed in a JavaScript alert, as shown in Figure 8-15.



Selected date is : Jan 1, 2009

Figure 8-15. The event handler extracts and formats the selected date.

Check your code, if necessary, against `yui_calendar.html` in `examples/ch08`.

Of course, displaying the date as a JavaScript alert serves no practical value. The purpose of this exercise has been to demonstrate how to create an event handler to respond to the selection of dates. You can use the data gathered by the event handler for a variety of things, including populating date fields in online forms or triggering a request to display events related to that date. Your ability to do that depends on your JavaScript skills.

Chapter review

This has been very much a hands-on chapter, digging into the mysteries of JavaScript, Spry, and other web widgets. However, it has barely managed to scratch the surface of a vast subject. Spry, jQuery, and the YUI Library have many enthusiastic fans, but JavaScript remains an uphill struggle for many others. While the web widgets are an attractive addition, they are not integrated into Dreamweaver to the same extent as Spry. Their principal advantage is that they speed up the deployment of sophisticated UI components by bringing together all the necessary external files, installing them, and creating the initialization script with a single mouse click. After that, it's up to you. I hope this chapter has whetted your appetite to experiment further with the framework(s) of your choice.

In the next chapter, we take an in-depth look at creating online forms, which lay the foundation for much of the rest of this book. Forms are the principal way of communicating with a database. You'll also continue your exploration of Spry, because Dreamweaver incorporates an impressive set of validation widgets that check user input before submitting it to the server for processing.