7 USING SPRY DYNAMIC EFFECTS AND COMPONENTS

Dreamweaver was first released in 1997 at the height of the "browser wars" between Microsoft and Netscape. Both companies fought for dominance of the market by introducing new features that were frequently incompatible with those of their rivals. This presented a major headache for anyone trying to use JavaScript to add dynamic features, such as image replacement, rollovers, and validation of user input. What worked in Internet Explorer didn't work in Netscape, and vice versa. Dreamweaver came to the rescue of many web designers by creating prepackaged scripts called **behaviors** that resolved the inconsistencies and incompatibilities. Even though the browser wars are now part of Internet history, their legacy still lingers on. The version of JavaScript used by Microsoft Internet Explorer (JScript) still doesn't fully comply with standards laid down by the World Wide Web Consortium (W3C). So, designers and developers still need help with scripts that will work cross-browser.

Behaviors are still part of the Dreamweaver toolset, but they're showing their age. Designed to overcome problems caused by ancient browsers, such as Netscape 4, they lack the features offered by recently developed JavaScript **frameworks** (code libraries), such as Prototype (`http://www.prototypejs.org/`), script.aculo.us (`http://script.aculo.us/`), or jQuery (`http://jquery.com/`). Adobe's answer has been to develop Spry, an extensive JavaScript code library that you can download free of charge from `http://labs.adobe.com/technologies/spry/home.html`. You don't actually need Dreamweaver to use Spry; it's completely tool-independent. As with any JavaScript framework, there's a learning curve involved if you want to integrate Spry features into your web pages. However, the Spry learning curve is considerably shortened—or even eliminated—by Dreamweaver. As you saw in the previous chapter, you can insert a Spry menu bar into a page in seconds. What takes the time is customizing the CSS, not building the JavaScript to control the submenu flyouts—all that is generated automatically. In this chapter, we'll continue our exploration of Spry by using Spry effects and components.

A common dilemma with website design is too little space to display all the content that needs to be on a particular page. In common with other Ajax frameworks, Spry makes it easy to build components—such as accordions and tabbed and collapsible panels—that slot into a web page and give it a much more dynamic feel. The Spry tabbed panels and accordion (see Figure 7-1) are a series of interlinked panels, in which just one panel is open at a time. Tabbed panels use the intuitive metaphor of tabs like a card index, while the panels of an accordion slide up and down to reveal their contents. Spry collapsible panels look the same as an accordion, except that each panel is independent so they can be opened and closed in any combination.

From the user's point of view, all three are intuitive metaphors that shouldn't need any explanation. Equally important, from the developer's point of view, they are easy to insert and customize. All you have to do is supply the content and skin the components with CSS. If you struggled with the Spry menu bar in the previous chapter, you'll be pleased to know that the style sheets of these Spry widgets are a lot simpler to edit. Dreamweaver CS4 adds another space-saving widget, the Spry Tooltip, which displays hidden content when the user mouses over an image or other element.

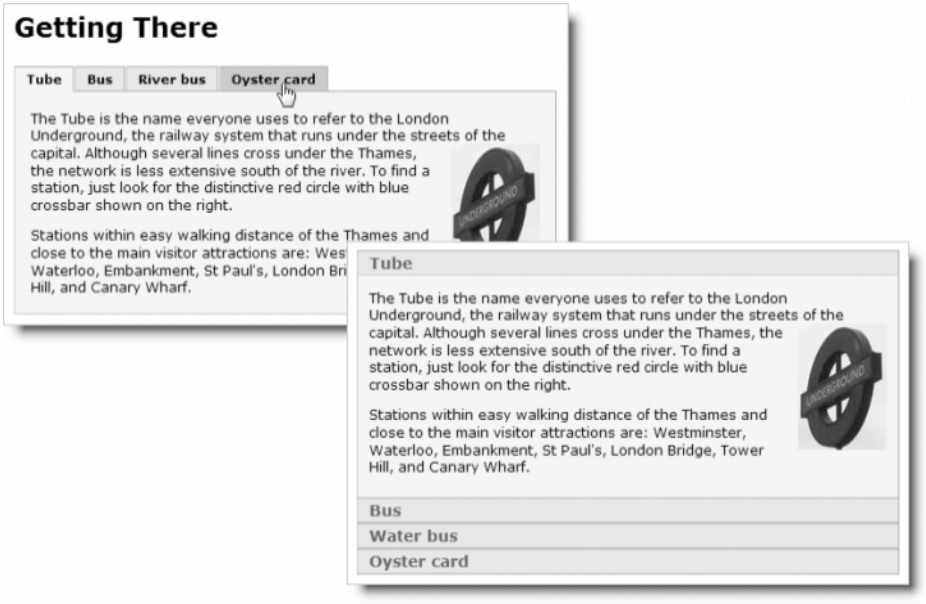


Figure 7-1. Tabbed panels and accordions are familiar website interfaces that users find easy to use.

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

- Applying Spry effects to different page elements
- Saving space with tabbed panels, accordions, and collapsible panels
- Selecting harmonious colors
- Styling user interface components
- Using the Spry Tooltip widget
- Removing Spry components cleanly from a page

Before diving into the user interface components, I will cover Spry effects. This is a series of dynamic effects, such as making page elements fade in and out, shrink, grow, and slide.

Animating page elements with Spry effects

Spry effects alter the look of a page element—or of the whole page itself—when a particular event occurs, such as the page loading, clicking a link, or mousing over an image. From a technical point of view, they manipulate the **Document Object Model (DOM)** of a web page. You don't need to know the intricate details of the DOM to use Spry effects, but it is important to understand the basic principle that underlies it.

DOM 101—why clean code matters

Figure 7-2 shows `fade.html` in `examples/ch07`, which contains a simple demonstration of the Spry Appear/Fade effect. Click the link at the top of the page, and the image of the Golden Pavilion in Kyoto fades out. Click it again, and the image fades back in.

Figure 7-2. The dynamic effect depends on JavaScript being able to identify the correct element to fade in and out.

Before adding the Spry effect to the page, the HTML looked like this:

```
<!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>Spry effects - fade in/out</title>
</head>
<body>
<p><a href="javascript:;">Fade image out/in</a></p>
<p><img src="../../images/kinkakuji.jpg" alt="Golden Pavilion" ➥
name="goldenpav" width="270" height="346" id="goldenpav" /></p>
</body>
</html>
```

The DOM sees this code in terms of the family tree shown in Figure 7-3.

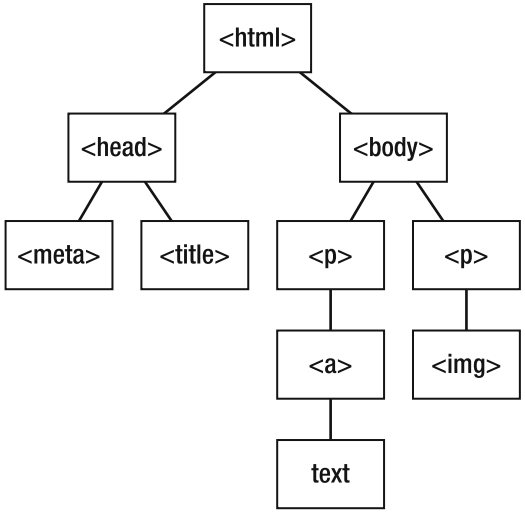


Figure 7-3. The contents of fade.html as seen by the DOM

For the Spry effect to fade the image in and out, it needs to pass the message from the text in the `<a>` tag in the first paragraph to the image in the second paragraph. The DOM acts as a sort of road map to make sure the message reaches the right destination. Either it can do it the hard way by following the hierarchy all the way up to the `<body>` tag and then drilling down to the `<img>` tag or it can do it the easy way with an `id` attribute. As you can see from the code listing, I have given the `<img>` tag an `id` attribute of `goldenpav`. Following the road map analogy, this acts as a signpost guiding Spry to the correct destination.

For all this to work smoothly, the road map needs to be clear. Tags need to be properly nested, and you should give `id` attributes to elements that you want the DOM and/or CSS to handle in a particular way. Dreamweaver makes this task a lot easier by warning you when tags are incorrectly nested (a message usually appears in the Property inspector, and the invalid code is highlighted in yellow). But remember: Dreamweaver is only a tool. It will do what you tell it. Even if Dreamweaver keeps your code perfectly valid, you should be aware of what's going on in Code view. If you clutter up your page with unnecessary `<span>` and `<div>` tags, the DOM road map becomes harder to navigate, resulting in pages that are sluggish and not user-friendly.

The other thing to remember is that an `id` attribute cannot be used more than once on the same page. Inexperienced web designers sometimes think, "I want all these elements to work the same way, so I'll give them all the same ID." It might work with CSS, but it won't work with Spry or any other JavaScript that relies on DOM manipulation.

> *An ID should be a unique identifier. Never use the same ID for more than one element on a page.*

Dreamweaver CS4 makes it a lot easier to assign an ID to page elements through the Property inspector. Most elements now have an ID field on the left of the Property inspector (for text elements, it's in the HTML view of the Property inspector). When assigning an ID, you should also bear the following rules in mind:

- Use only alphanumeric characters, hyphens, or underscores.
- Do not use spaces or punctuation.
- Never begin with a number.

Applying a Spry effect

Spry effects are event-driven, so you need to decide which event to use and which element to trigger it. For example, you could attach a Spry effect to the <body> element of a page and use the `onload` event to trigger it as soon as the page finishes loading. Other common choices are using a text link that triggers the effect when it's clicked, or using an image to trigger an effect when the mouse passes over it. The other decision you need to make is which element to apply the effect to. Dreamweaver refers to this as the **target** element.

All Spry effects are grouped with the original Dreamweaver behaviors, which are in a slightly different location from previous versions. The Behaviors panel has been combined with the Tag Inspector panel, and you access its features by clicking the Behaviors button, as shown in Figure 7-4.

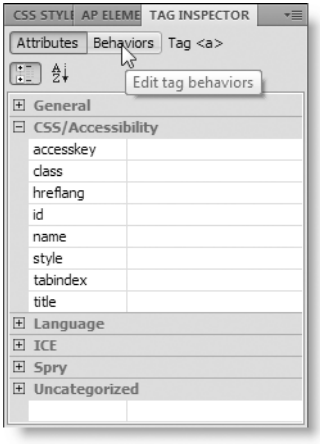


Figure 7-4.
Dreamweaver behaviors and Spry effects are now in the Tag Inspector panel.

Like the Property inspector, the Tag Inspector panel is context-sensitive. Its options depend on where the insertion point is currently located in the Document window. Click in an element or select it in the Document window before using the Tag Inspector panel. To open the panel, double-click its tab or, if you're using iconic mode, click its icon. You can also use Window ➤ Tag Inspector or press F9/Shift+Opt+F9. There are also menu and keyboard

shortcuts that take you directly to the Tag Inspector panel in Behaviors mode: Window ➤ Behaviors or Shift+F4 (the keyboard shortcut is the same on both Windows and Mac).

> *Don't confuse the* Tag Inspector *panel in* Behaviors *mode with the* Server Behaviors *panel. Dreamweaver uses* behaviors *to mean JavaScript-driven features. Server behaviors use server-side code, such as PHP. You'll work with the* Server Behaviors *panel a lot in the second half of this book.*

To apply a Spry effect, select the element that will be used to trigger the effect, open the Tag Inspector panel in Behaviors mode, click the plus (+) button, and select the effect from the Effects submenu, as shown in Figure 7-5.

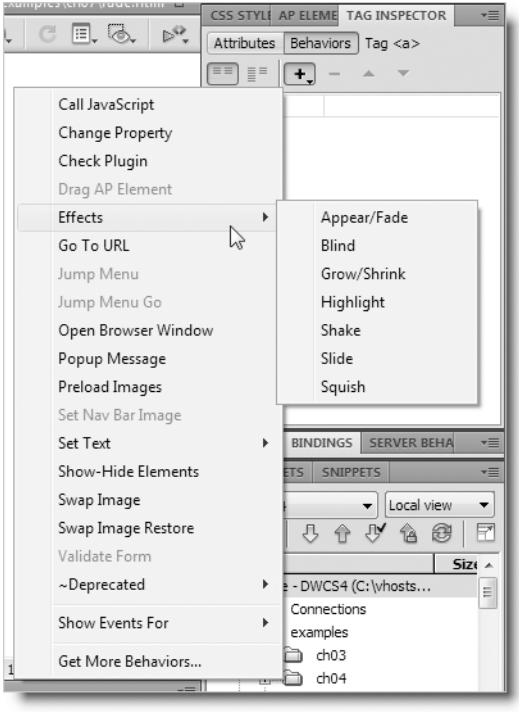


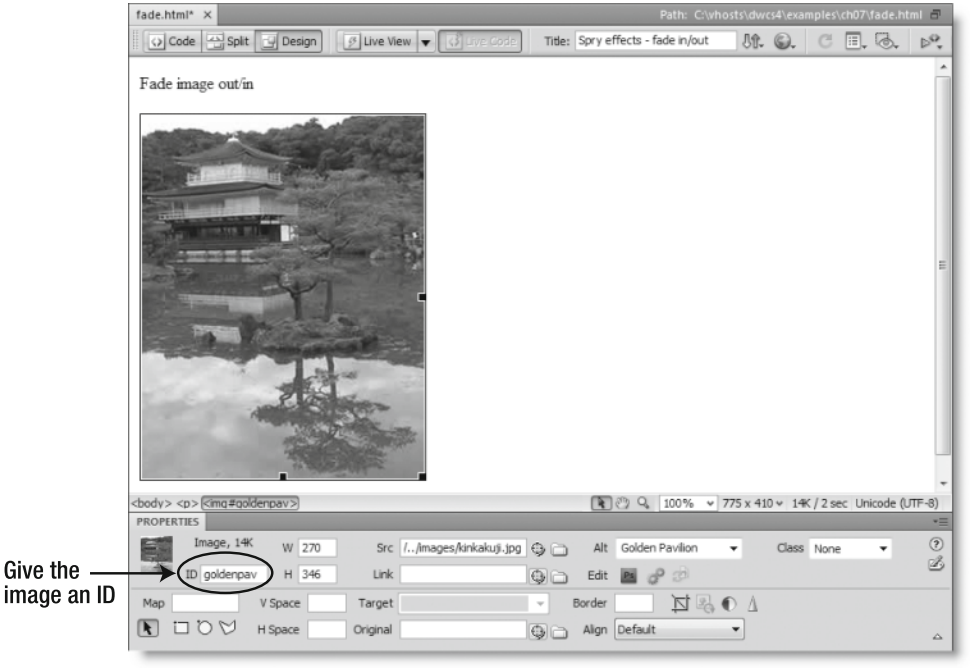
Figure 7-5.
The Spry effects are grouped with the original Dreamweaver JavaScript behaviors.

Fading an image

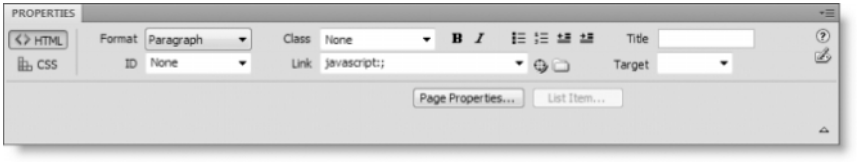
This brief exercise shows how to apply the Spry Appear/Fade effect to an image. The basic procedure for applying all Spry effects is identical, so once you understand the technique, you can quickly apply any effect.

1. Create a new subfolder called `ch07` in the `workfiles` folder.
2. Create a new HTML document called `fade.html`, and save it in `workfiles/ch07`.

3. Type some text to act as a trigger, and press Enter/Return. This wraps the text in paragraph tags and creates a new paragraph.
4. Insert an image in the new paragraph. I used `kinkakuji.jpg`, but you can use any image in the images folder. It's not necessary for this exercise to give the image alternate text, but you should get in the habit of doing so.
5. With the image selected in the Document window, give it a unique identity by entering a name in the ID field of the Property inspector. Press Enter/Return to register the change. Your page should now look similar to this:



6. Select the text in the first paragraph, and convert it into a dummy link by entering javascript:; in the Link field of the HTML view of the Property inspector, as shown here:



Using `javascript:;` rather than # for a dummy link prevents the page from jumping to the top if you have a lot of content on the page.

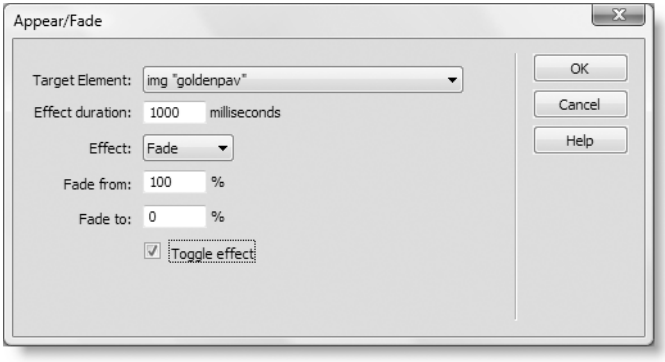
7. Press Enter/Return or Tab to ensure that the link is created. This is important, because you can't apply the Spry effect until the trigger element exists.

> *Values entered in the Property inspector and other panels are not added to the underlying code until the focus moves to another part of the UI. Pressing Enter/Return registers the new value in the underlying code. Pressing the Tab key or clicking elsewhere in the Document window has the same effect.*

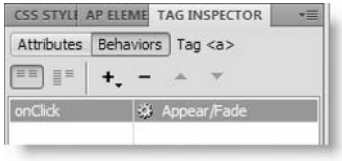
8. Open the Tag Inspector panel in Behaviors mode, click the plus button to activate the Behaviors menu, as shown in Figure 7-5, and select Effects ➤ Appear/Fade.

9. The dialog box that opens sets the options for the effect. The Target Element drop-down menu lists all elements that can be used as targets (usually elements that have an ID). The other options are self-explanatory. The Toggle effect checkbox at the bottom of the dialog box lets you run the effect in reverse when the trigger event is repeated. So, in the case of this effect, it makes the target element fade back into view when the event is triggered again.

Choose the image as the target element, and select the option to toggle the effect. Check your settings with the following screenshot, and click OK:



10. The effect is now listed in the Tag Inspector panel, as shown here:



The event that will be used to trigger the effect is displayed on the left. You can change this value by clicking it to open a menu of the available events. However, the default value, onClick, is fine in this case (although Dreamweaver displays the event with an uppercase *C*, the underlying code inserts `onclick` all in lowercase if you have selected an XHTML document type).

To change any of the effect's settings, make sure the trigger element is selected in the Document window, and double-click the listing shown in the preceding screenshot (or right-click, and select Edit Behavior from the context menu).

11. Save `fade.html`. If this is the first time you have used a Spry effect in the current site, Dreamweaver displays a message telling you that it has copied `SpryEffects.js` to your Spry assets folder (the default name is `SpryAssets`). Click OK.

12. Click the Live View button in the Document toolbar, and then click the text link in `fade.html`. The image should fade out over one second. Click the link again, and the image should fade back in.

    If necessary, you can check your code against `fade.html` in `examples/ch07`.

The basic procedure for applying the original Dreamweaver JavaScript behaviors is the same. Since many of them are rather old, I don't plan to cover them in this book. You can find a description of each one in Dreamweaver Help (F1 or Help ➤ Dreamweaver Help) ➤ Applying JavaScript behaviors ➤ Applying built-in Dreamweaver behaviors.

Exploring the available effects

Table 7-1 summarizes what each Spry effect does and which target elements it can be used with. Appear/Fade and Highlight can be used with almost any tag, but the others are more restricted. The complete list of supported target elements is reproduced mainly for reference. Most effects can be applied only to a block element, such as a heading, paragraph, or `<div>`. Appear/Fade, Highlight, and Shake can be applied directly to an `<img>` tag. If in doubt, wrap the target element in a `<div>`, and assign it an ID.

Table 7-1. Spry effects and supported target elements

Effect	Action	Supported targets	Not supported
Appear/Fade	Fades an element in or out	Most tags	`applet`, `body`, `iframe`, `object`, `tbody`, `th`, `tr`
Blind	Reveals or conceals an element, like pulling a window blind up or down	`address`, `applet`, `center`, `dir`, `dd`, `div`, `dl`, `dt`, `form`, `h1–6`, `li`, `menu`, `p`, `pre`, `ol`, `ul`	Any other tag
Grow/Shrink	Grows or shrinks an element to either the center or top left	`address`, `applet`, `center`, `dd`, `dir`, `div`, `dl`, `dt`, `form`, `img`, `menu`, `p`, `pre`, `ol`, `ul`	Any other tag
Highlight	Applies a color transition to the element's background	Most tags	`applet`, `body`, `frame`, `frameset`, `noframes`

Effect	Action	Supported targets	Not supported
Shake	Shakes an element horizontally for half a second	address, applet, blockquote, dd, dir, div, dl, dt, fieldset, form, h1–6, hr, iframe, img, li, menu, object, p, pre, ol, table, ul	Any other tag
Slide	Slides an element up or down to conceal or reveal it	blockquote, center, dd, div, form, img	Any other tag
Squish	Collapses or expands an element to or from its upper-left corner	address, applet, center, dd, dir, div, dl, dt, form, img, menu, p, pre, ol, ul	Any other tag

The dialog box for each effect is very similar, and all share the following common settings:

- Target Element: Dreamweaver automatically identifies every element on the page that the effect can be applied to. Select the element from the drop-down list. Unless the effect is being applied to the trigger element, the target must have an ID. In the case of the Shake and Squish effects, this is the only setting.
- Effect duration: This is the length of the effect, measured in milliseconds. The default setting is 1000—in other words, one second.
- Effect: The available options depend on the effect but normally specify the direction in which the target element will move.
- Toggle effect: Selecting this option reverses the effect the next time the event is triggered.

The best way to learn how to use Spry effects is to experiment with them. However, the hints in the following sections should help you.

Appear/Fade

This effect can be applied to just about any element on a page, and it affects everything inside the target element. Making an element fade to nothing does not alter the layout of the page. An empty space remains where the element originally was.

The <body> tag cannot be used as the target element of this effect. To get the whole page to fade in after it finishes loading, wrap the entire contents of the page in a . Use the <body> tag as the trigger, set the as the target element, and set the event to onLoad. You can see this in fade_in.html in examples/ch07. A called container has been selected as the target element, the effect duration set to 3000 (3 seconds), and the effect set to Appear.

Blind

This is very similar to Slide, except that Blind acts like a mask scrolling up or down in front of the target element, whereas Slide moves the whole target element. Blind up results in the target element disappearing from the bottom; with Blind down, the target element is normally hidden, and the mask moves down to reveal it. Content below the target element moves up and down in time with the effect.

Images need to be wrapped in a block element such as a paragraph or `<div>` to use Blind. Use the block element as the target. For an example, see `blind.html` in `examples/ch07`.

Grow/Shrink

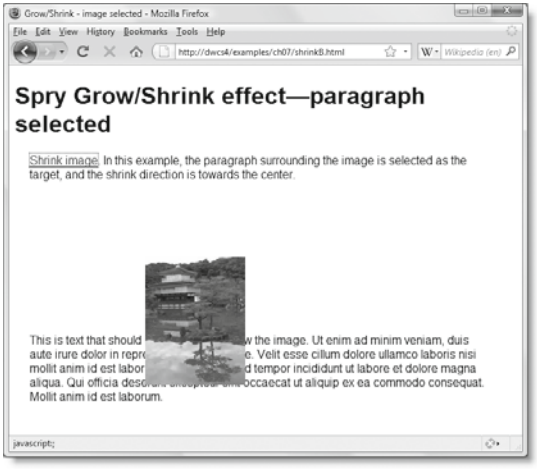
This effect works with a wide range of block elements and images, but it can have unexpected results (see Figure 7-6), so you need to test your pages and CSS carefully when using it.

There are two options for the direction of movement: to and from the center of the target element (see Figures 7-6A and 7-6B) or to and from its top-left corner (see Figures 7-6C and 7-6D). Grow/Shrink can be applied directly to an image or its containing element. Each screenshot shows what happens when the target element is shrunk to 50 percent of its original size but in a variety of circumstances. (You can test the results in `shrinkA.html`, `shrinkB.html`, `shrinkC.html`, and `shrinkD.html` in `examples/ch07`.)

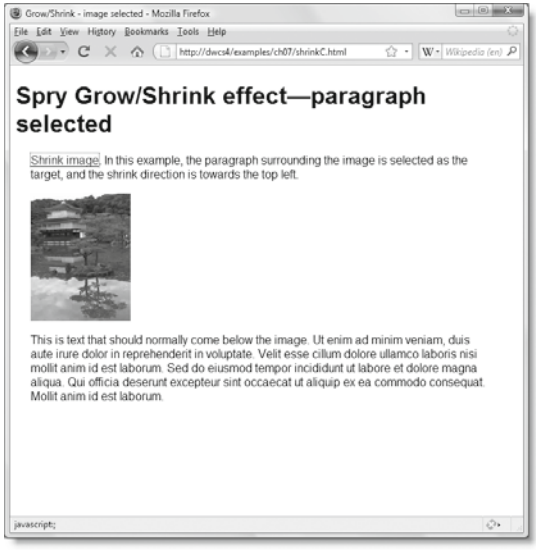
- Figure 7-6A shows what happens when the image itself is selected as the target element and shrunk to its center. Any content below the target element moves up, but the image moves down, resulting in an overlap. The same happens if the effect is applied to a surrounding element with the same width and height as the image.
- Figure 7-6B shows what happens if the effect is applied to a surrounding block element with no fixed height and is shrunk to its center: the parent element and its contents shrink together but move to the center of the page.
- Figure 7-6C shows what happens if the effect is set to move to the top left and is applied to the surrounding `<div>`, regardless of whether the `<div>` has fixed dimensions. The same happens if the image is selected as the target but *only* if the surrounding `<div>` has no height.
- Figure 7-6D shows the gap created by applying the effect directly to the image and shrinking it to its top-left corner when the surrounding `<div>` has a fixed height. The text remains in its original position, much further down the page.

Test your layout carefully if you use this effect.

A: Image shrunk to center



B: Container with no height shrunk to center



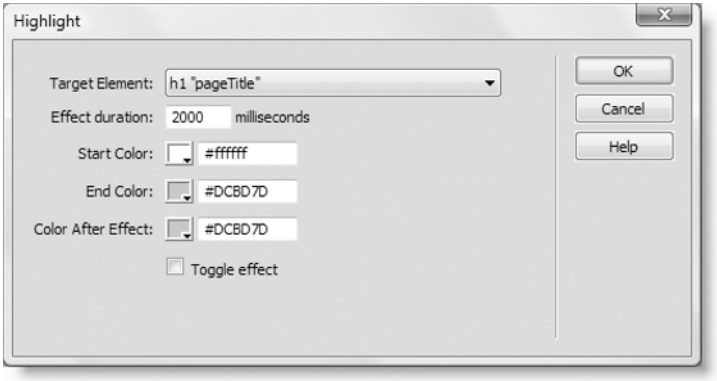
C: Image or container shrunk to top left

D: Image shrunk to top left; container has height

Figure 7-6. The Grow/Shrink Spry effect can produce unexpected changes to your layout (see text for details).

Highlight

Highlight changes the background color of the target element. As the following screenshot shows, the Highlight dialog box has three color settings: Start Color, End Color, and Color After Effect. You can set these either by typing the hexadecimal color value in the text field (preceded by #) or by clicking the color picker to the left of the text field.



The meanings of Start Color and End Color are what you would expect. Effect duration sets the time taken (in milliseconds) to transition from one color to the other—2000 (or 2 seconds) seems to be the optimal choice—and the transition follows a visually pleasing curve. Color After Effect is the color to which the background is set after the transition, and it cuts in immediately. You need to choose this color carefully. I find it's best to set this value either to the same as Start Color or End Color. Otherwise, the transition appears unnaturally abrupt. You can see an example in `highlight_text.html` in `examples/ch07`.

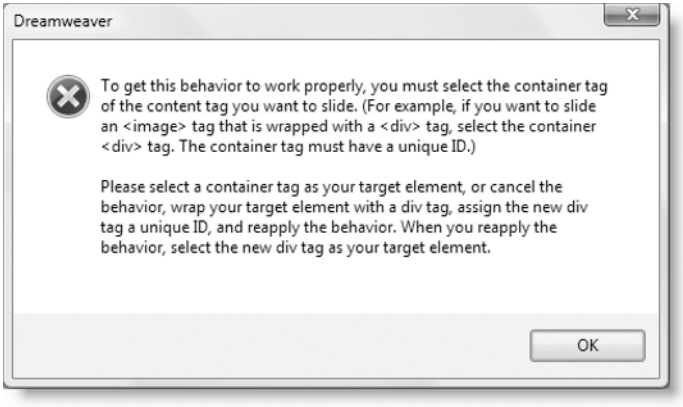
When Highlight is applied directly to an image, there must be padding around the image for the background color to be visible. Adding only margins to the image has no effect, because background color does not affect the margin of an element. See `highlight_padding.html` and `highlight_margin.html` in `examples/ch07`.

Shake

This is my least favorite effect. It has only one option: the target element, which it shakes horizontally for half a second. It might be appropriate in advanced Ajax contexts to indicate that an element has been updated asynchronously, but it would be more useful if you could set the speed and duration of the movement. The danger is that it will become the modern equivalent of the `<blink>` tag—mercilessly abused because it looks "cool." Use with care. Depending on your layout, this effect sometimes spawns a horizontal scrollbar in the browser. There's an example in `shake.html` in `examples/ch07`.

Slide

Slide is similar to Blind, but rather than a mask moving over the target element, the element itself moves. As Table 7-1 shows, this effect can be applied to only a small range of block elements or images. You cannot apply the Slide effect directly to the element you want to slide in and out of view. Instead, the target element must be a `<div>` wrapped around it. Although that's straightforward, what makes matters slightly complicated is that the Slide effect is very picky about the elements it accepts immediately inside the wrapper. The child element of the wrapper `<div>` *must* be one of the following: `<blockquote>`, the deprecated `<center>` element, `<dd>`, `<form>`, `<img>`, or another `<div>`. If the child element is anything else, you get this warning:



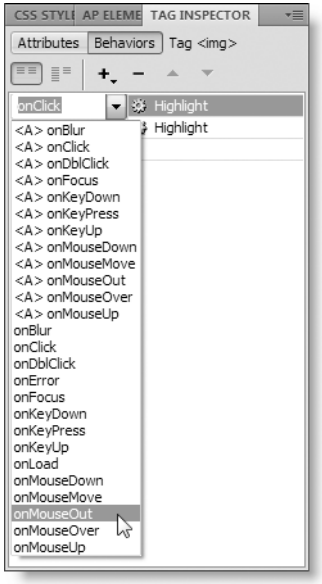
The image of the Golden Pavilion in `slide.html` in `examples/ch07` is wrapped in a `<div>`, not a paragraph. If you want to use a paragraph with the Slide effect, you must wrap the paragraph in two `<div>` tags and use the outer one as the target element.

Squish

Squish collapses the target element from the bottom-right corner toward the top left until it disappears completely and is very easy to apply. The Squish dialog box has only one setting: the target element. Any content below the target element moves up to fill the gap, as demonstrated in `squish.html` in `examples/ch07`. Unlike other Spry effects, there's no toggle option in the Dreamweaver dialog box, and you can't specify the start and end sizes of the target element.

Applying multiple events to a trigger element

You're not limited to applying a single event to the trigger element for a Spry effect or behavior. In the examples of the Highlight effect, I have applied the `onmouseover` and `onmouseout` events to the image. The first event applies the Highlight effect when you



mouse over the image. The second event applies the same effect in reverse. To apply multiple events to the same trigger, just apply the effect again, and select a different event from the drop-down menu in the Tag Inspector panel, as shown here.

> *Dreamweaver often seems reluctant to let you change the trigger event from* onClick*. I usually find it accepts the change the second time you select the new event.*

When you select an image, the drop-down menu contains a duplicate set of events preceded by <A>, as shown in the screenshot alongside. This option inserts the event handler in a pair of <a> tags wrapped around the image. This is necessary for some older browsers that don't recognize event handlers attached directly to an image.

If you choose different event handlers, the order that behaviors or effects are listed doesn't matter. However, you may need to change the order when you use the same event handler for more than one behavior. This sometimes happens when adding several behaviors to the <body> tag to be executed when the page first loads. You do this by selecting an event in the Tag Inspector panel and moving it up or down the list with the up and down arrows at the top of the panel.

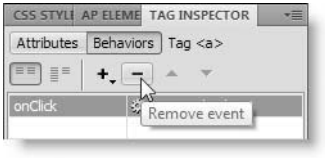
Removing effects and behaviors cleanly

A question I often see in online forums is "Why does my browser report errors on the page?" Frequently, the answer is that an effect or behavior has been removed, but the event handler that triggers it has been left behind. Another cause is the removal of a page element, such as an image or a , that an effect or behavior is attempting to find. If you treat Dreamweaver purely as a WYSIWYG tool, you're likely to end up with similar problems. If you remove an element that triggers an effect or behavior or is the target of one, you must do it in the correct manner.

Removing an effect or behavior involves three simple steps, as follows:

1. Select the page element that the effect or behavior is applied to.
2. Select the effect or behavior in the Tag Inspector panel.
3. Click the minus (–) button, as shown in the following screenshot:

Instead of clicking the minus button, you can right-click and select Delete Behavior. You can even just press Delete (but make sure the behavior is selected in the Behaviors panel first).



Everything is removed cleanly, preventing errors from popping up later in your page. However, `SpryEffects.js` is *not* deleted from the `SpryAssets` folder, in case it's needed by other pages. The link to the external JavaScript file is also preserved if it's required by other effects in the page.

Restoring a deleted effect or behavior

If you delete a behavior by mistake, you can restore it by pressing Ctrl+Z/Cmd+Z or by selecting Edit ➤ Undo Remove Behavior (Edit ➤ Undo on a Mac). This always undoes the last action. By default, Dreamweaver remembers your last 50 steps. So, you can continue pressing Ctrl+Z/Cmd+Z to restore a deleted effect or behavior if you change your mind after doing something else (although you lose those changes too).

An alternative way to undo several steps is to use the History panel. The History panel is not displayed by default but is automatically added to the bottom of the panel groups the first time you open it (Window ➤ History). The keyboard shortcut (Shift+F10) is available on Windows only. To learn more about the History panel, open Help (F1), and select Adding content to pages ➤ Automating tasks ➤ Use the history panel.

You can change the number of steps that can be undone by altering Maximum number of history steps in the General category of the Preferences panel (Edit/Dreamweaver ➤ Preferences). Resist the temptation to increase this number by a significant amount, because it is memory intensive. The default 50 is the optimal level.

Another useful way of retracing your steps is the Revert command on the File menu. This undoes all changes in a document and restores it to the last saved state.

Conserving space with Spry UI components

Dreamweaver CS4 comes with four Spry user interface components or widgets designed to solve the problem of putting a lot of information at the user's fingertips without creating interminably long pages: tabbed panels, accordion panels, collapsible panels, and—new to this version—tooltips. Several features are common to working with all Spry widgets. If you worked through the previous chapter about the Spry menu bar, they should be familiar to you, but it's worth repeating them here:

- Always save your page in a Dreamweaver site before inserting a Spry widget. Dreamweaver prompts you if you forget.
- After inserting a widget, save the page to link the external JavaScript file and style sheet, and copy them to the site's Spry assets folder (see "Setting other site options" in Chapter 2). All instances of a widget in a site share the same files, so they are copied only when inserting the first instance. *You must upload these files to your remote server when deploying your site on the Internet. The Spry widgets won't work without them.*

Dreamweaver attaches the widget's style sheet immediately above the closing </head> tag. If your page has style rules embedded in conditional comments, move the link to the style sheet above the conditional comments.

- Dreamweaver inserts a block of JavaScript at the bottom of the page to initialize the widget when the page loads.
- To see the widget's details in the Property inspector, hover your mouse pointer over the widget in Design view, and click the tab at the top left of the surrounding border.

Although the Spry UI components are great space savers, the contents of hidden panels are loaded at the same time as the rest of the page. Don't put lots of heavy graphics in these widgets or overuse them on any individual page. The external JavaScript file and style sheet for each widget add about 20KB to a page but are stored in the browser's cache after loading the first time.

Building a tabbed interface

Tabbed panels use the common metaphor of tabs at the top of folders in a filing cabinet. Click the tab, and the associated content is displayed in the panel beneath. It's a clean, intuitive way of storing a lot of content in a relatively small space. The example in Figure 7-7 has four tabs, so the total space required to display the information is one fourth of what it would normally be.

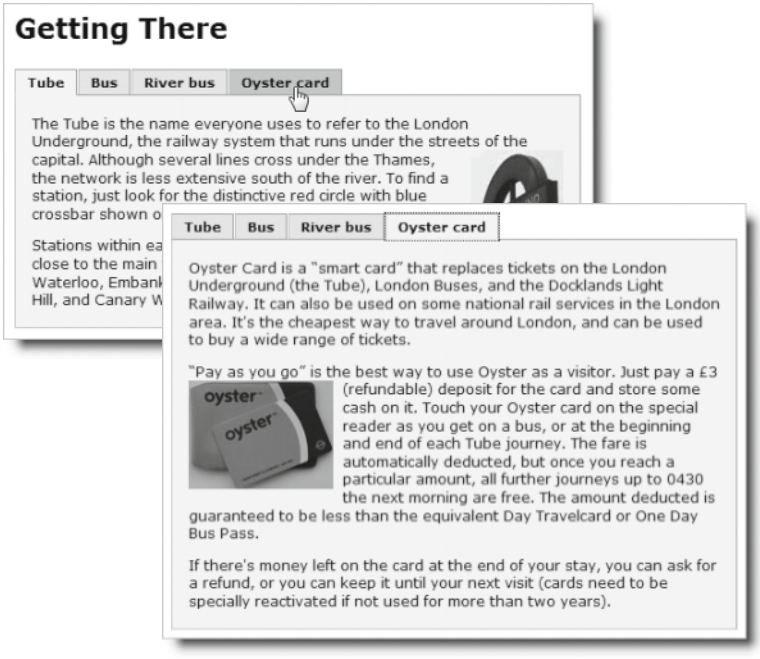


Figure 7-7. Tabbed panels are a great way of presenting related information in a confined space.

The Spry tabbed panels widget takes only the click of a button to insert, and it degrades gracefully in old browsers or if JavaScript is turned off. The panels expand to display their contents if the browser cannot handle the JavaScript. The accordion, collapsible panels, and tooltip expand in a similar way, making all four user interface widgets accessible.

Let's take a look at the anatomy of a tabbed panels widget.

Examining the structure of the tabbed panels widget

You can insert a Spry tabbed panels widget in three ways: from the Spry tab of the Insert bar, from the Layout tab of the Insert bar, or by choosing Insert ➤ Spry ➤ Spry Tabbed Panels. This creates a default two-tab widget (see Figure 7-8) at the current insertion point in the page.

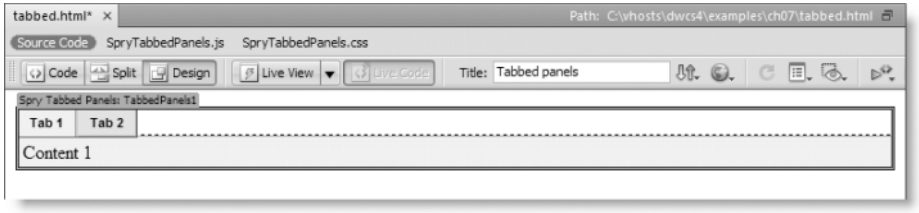


Figure 7-8. The default tabbed panels widget contains two tabs styled with a neutral gray interface.

As you can see in Figure 7-8, the Related Files toolbar displays the names of two dependent files (`SpryTabbedPanels.js` and `SpryTabbedPanels.css`). Until you save the page, these are stored in a temporary folder, so you should always save the page immediately after inserting a Spry widget for Dreamweaver to update the links and copy the dependent files to the site's Spry assets folder.

The tabbed panels are controlled by JavaScript and CSS, but unlike the Spry menu bar, there's no option on the Property inspector to toggle the CSS on and off. However, if you switch to Code view, the underlying HTML looks like this:

```
<div id="TabbedPanels1" class="TabbedPanels">
  <ul class="TabbedPanelsTabGroup">
    <li class="TabbedPanelsTab" tabindex="0">Tab 1</li>
    <li class="TabbedPanelsTab" tabindex="0">Tab 2</li>
  </ul>
  <div class="TabbedPanelsContentGroup">
    <div class="TabbedPanelsContent">Content 1</div>
    <div class="TabbedPanelsContent">Content 2</div>
  </div>
</div>
```

The whole widget is wrapped in a `<div>`; the tabs are an unordered list, and the panels are in a nested `<div>`. Each individual panel is also a `<div>`, nested one level further down. The only element that has an ID is the overall wrapper `<div>`. Dreamweaver automatically calls the first tabbed panels widget on a page `TabbedPanels1` and numbers subsequent instances `TabbedPanels2`, and so on. Everything else is controlled by classes. Although

each element has a class assigned to it explicitly in the underlying code, other classes are generated dynamically by the external JavaScript file. Table 7-2 explains what each class is for. In common with all user interface widgets, the class names are long but descriptive.

Table 7-2. The classes used to style the tabbed panels widget

Class	Type	Purpose
`TabbedPanels`	Explicit	Eliminates margin and padding surrounding the widget and clears any preceding floats. *This class must always have an explicit width*. The default value is `100%` to fill all available space.
`TabbedPanelsTabGroup`	Explicit	Removes margin and padding from the tabs as a group.
`TabbedPanelsTab`	Explicit	Styles the individual tabs. Uses relative positioning to shift the tabs 1 pixel down and gives the bottom border the same color as the top border of `TabbedPanelsContentGroup`. This creates the illusion that the tabs are being drawn behind the content panel. Two non-standard properties (`-moz-user-select` and `-khtml-user-select`) are set to `none` to prevent users from selecting the text in Firefox, Mozilla, and Konqueror.
`TabbedPanelsTabHover`	Dynamic	Controls the rollover look of the tabs.
`TabbedPanelsTabSelected`	Dynamic	Sets the background color and bottom border of the currently selected tab to the same as the `TabbedPanelsContentGroup` to create the illusion that the tab is part of the panel.
`TabbedPanelsContentGroup`	Explicit	Ensures that the panels sit beneath the tabs. Sets the background and border colors for the panels.
`TabbedPanelsContent`	Explicit	Styles the content of an individual panel. By default, only adds `4px` padding.
`TabbedPanelsContentVisible`	Dynamic	Empty style rule that can be used to give a different style to the currently visible panel.

The `<li>` *tags contain the* `tabindex` *attribute, which makes the code invalid according to the W3C specifications. Although Spry generates classes dynamically, Internet Explorer doesn't support setting* `tabindex` *through JavaScript, so this was the compromise adopted to make it possible to navigate the panels with the Tab key. If W3C validation is vital to you, remove the* `tabindex` *attributes. However, this will make your page less accessible to assistive technology for the disabled and keyboard users. Occasionally bending the rules like this makes sense and has no adverse effect in any browser.*

Editing a tabbed panels widget

The Property inspector has only three settings for the tabbed panels widget (see Figure 7-9): ID, number and order of panels, and the default panel. The Customize this widget link opens Dreamweaver Help at the page listing the style settings.

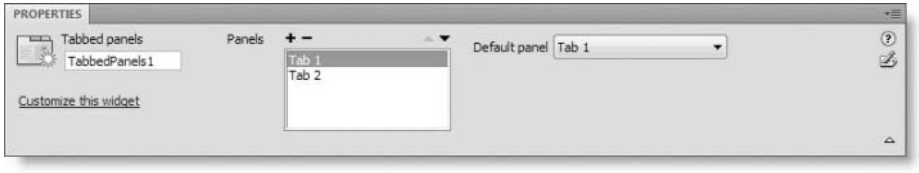


Figure 7-9. The Property inspector for the tabbed panels widget is very simple.

Use the plus (+) and minus (–) buttons to add or remove panels, and use the up and down arrows to reorder them. The name of each panel changes when you edit the tabs in Design view. The Default panel drop-down menu on the right determines which panel is open when the page first loads.

You can open a tab or panel for editing in Design view in two ways, as follows:

- Bring up the details of the widget in the Property inspector, and select the panel name in the Panels list.
- Position your mouse pointer over the right side of the tab until an eye icon appears, as shown in Figure 7-10, and click.

Figure 7-10. Click the eye icon at the right side of a tab to reveal its associated panel for editing.

Each panel is a `<div>`, so you can insert anything you like: text, images, and so on.

Inserting and editing a tabbed panels widget

Roll up your sleeves, and insert a tabbed panels widget into `stroll.html`. To make it easier to dip into individual chapters, the files in `examples/ch07` use the version of `stroll.html` from Chapter 5 without the Spry menu bar, because it involves fewer dependent files.

1. Copy `stroll.html` and `stroll.css` from `examples/ch07` to `workfiles/ch07`. Update links if prompted by Dreamweaver. Save a copy of `stroll.html` as `stroll_tabbed.html`.

   Note that in Dreamweaver CS4, using File ➤ Save As (Ctrl+Shift+S/Shift+Cmd+S) opens the renamed version in a separate tab and gives it focus in the Document window. In previous versions, Dreamweaver closed the original document and displayed the new one in the same tab. Later in the chapter, you'll make fresh copies of `stroll.html` to experiment with other widgets, but you don't need it at the moment, so you can close it if you prefer to keep as few documents open as necessary.

   > *If you make any changes to a file before using* File ➤ Save As*, the changes are saved* only *in the renamed file, and the original file is restored to its last saved state.*

2. Scroll down to the end of the first block of text in the `mainContent` `<div>` (just above the Artists at Work heading). Press Enter/Return to insert a new paragraph. Type Getting There, and convert it to a heading by selecting Heading 2 from the Format drop-down menu on the left of the Property inspector.
3. With your cursor at the end of the new heading, click the Spry Tabbed Panels button on the Spry tab of the Insert bar (or use the Layout tab or Insert menu as described earlier). You should now have a tabbed panels widget in the middle of the page, as shown in Figure 7-11.

Figure 7-11. By default, the tabbed panels widget fills the available horizontal space.

As long as your cursor is at the beginning or end of an existing element when you insert a widget, Dreamweaver correctly places the widget outside the existing element. If your cursor is anywhere else, Dreamweaver splits the existing element by creating closing and opening tags and inserting the widget between them.

4. Save `stroll_tabbed.html`, and click OK if prompted to copy the dependent files (this happens only the first time you create a tabbed panel widget in a site).
5. Rename `SpryTabbedPanel.css` in the Spry assets folder as `SpryTabbedPanel_stroll.css`, and update the links when prompted. Move the link to `SpryTabbedPanel_stroll.css` above the conditional comments in the `<head>` of the page. There won't be any conflicts of style rules, but this is a good habit to adopt.
6. Place your cursor inside the first tab, delete Tab 1, and type Tube.
7. Open `getting_there.doc` in `examples/ch07`, and copy the paragraphs labeled Tube to your clipboard. If you can't open a Word document, the text is in `getting_there.txt`, but Dreamweaver won't do the automatic formatting in the next step.
8. Highlight Content 1 in the tabbed panels widget, and paste the contents of your clipboard into the panel. If you set the Copy/Paste options in the Preferences panel using the settings shown in Figure 3-7 in Chapter 3, Dreamweaver should automatically preserve the paragraph structure from the Word document. Otherwise, press Ctrl+Shift+V/Shift+Cmd+V or select Edit ➤ Paste Special, and select Text with structure plus basic formatting (bold, italic) and Clean up Word paragraph spacing (Paste Special is described in Chapter 3).

   If you used the plain text in `getting_there.txt`, you need to format it manually as paragraphs with the Format drop-down menu in the HTML view of the Property inspector. Dreamweaver places a `<br />` tag between the paragraphs, so you need to split them by pressing Enter/Return and then remove the extra line created by the `<br />` tag.
9. Position your cursor inside the second tab, and rename it Bus.
10. Open the second panel for editing by selecting it in the Property inspector (click the turquoise tab at the top left of the widget, if necessary) or clicking the eye icon as shown earlier in Figure 7-10. Copy the Bus paragraphs from `getting_there.doc`, and paste them in place of the placeholder text in the second panel.
11. Click the turquoise Spry Tabbed Panels tab at the top left of the widget to bring up its details in the Property inspector, and click the plus button in the Property inspector to add two more panels. Rename them Water bus and Oyster Card, and replace the placeholder text in each panel with the copy from `getting_there.doc`.
12. With the Oyster Card panel open, insert `oystercard.jpg` from the `images` folder at or near the beginning of the second paragraph. Enter Oyster Card as the Alternate text when prompted.
13. To make the text wrap around the image, with the image still highlighted, select fltlft from the Class drop-down menu in the HTML view of the Property inspector.

14. Open the first panel (Tube) for editing, and insert `underground.jpg` at the beginning of the first paragraph. Set Alternate text to Underground station sign and Class to fltrt.

15. Click the Live View button in the Document toolbar. The bottom half of the page should look like Figure 7-12. Click the various tabs to display the other panels. You'll see that the height of the panels expands and contracts depending on the amount of content. All content below the tabbed panels is repositioned according to the height of the selected panel, so you need to be careful when incorporating this widget in a design where the layout needs to be pixel perfect.

    Check your code if necessary with `stroll_tabbed.html` in `examples/ch07`.

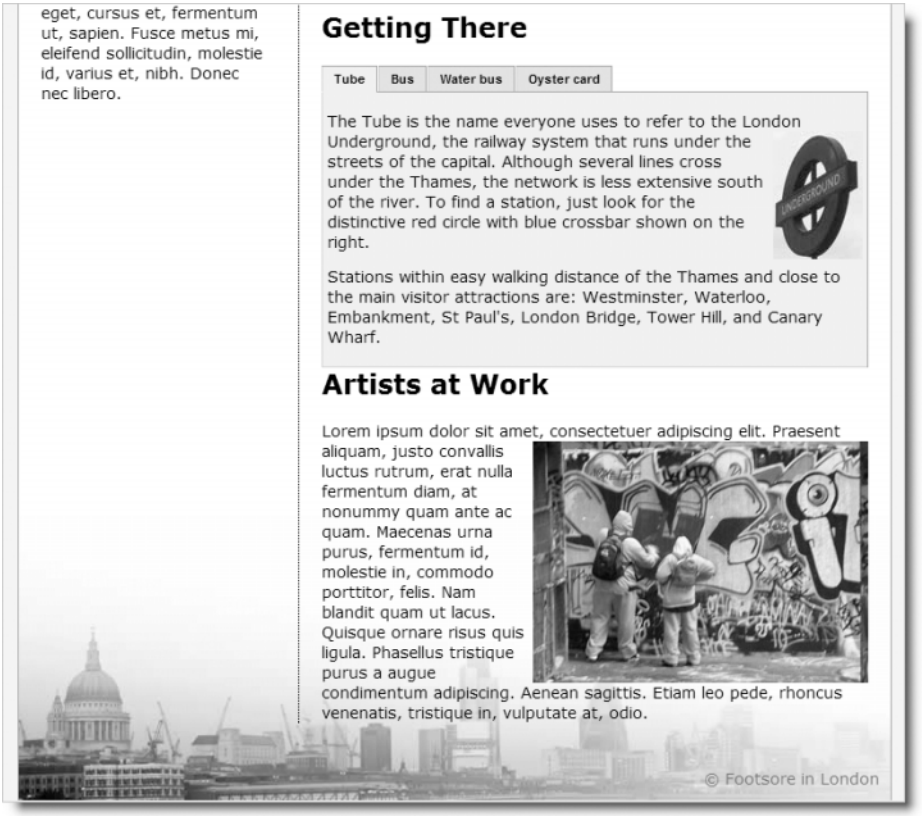


Figure 7-12. Even without customizing the styles, Spry tabbed panels look at home in most pages.

The neutral gray styling fits in easily with many designs, so you could leave it as it is. However, I don't imagine that you'll let me get away with that, so let's restyle the panels. The bottom of the panels is too close to the following headline, so that needs fixing, too.

Selecting harmonious colors

The tabbed panels style sheet uses four shades of gray, ranging from light (#EEE) to dark (#999). I decided to use as my base colors the pink (#F8F1EB) from the page background and the russet (#C99466) border down both sides of the `container <div>`. Table 7-3 lists the colors that I finally decided on.

Table 7-3. Conversion chart for Dreamweaver defaults and substituted colors

Default color	Replacement	Applies to
Pale gray (#EEE)	Light pink (#FAF3ED)	Panel background color and selected tab
Light gray (#DDD)	Darker pink (#F2E1D2)	Nonselected tabs
Medium gray (#CCC)	Light brown (#DFBD9F)	Tabs on rollover and lighter borders
Dark gray (#999)	Russet (#C99466)	Darker borders

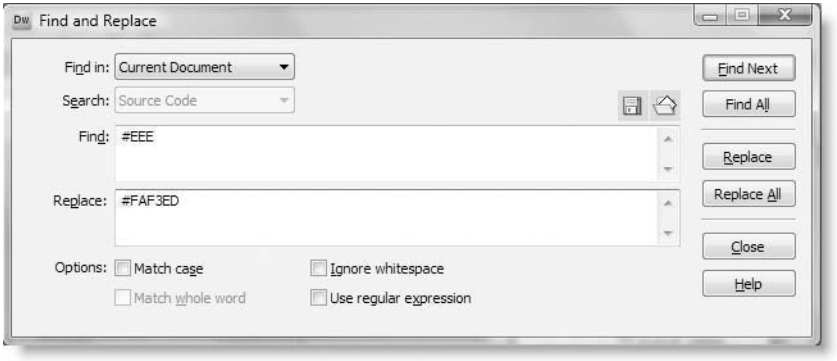
To simplify customization of a Spry widget, make a similar chart of the default colors and your chosen replacements. You can then go through the style rules quite quickly to make the substitutions.

Styling a tabbed panels widget

Let's style the tabbed panels using the color scheme outlined in Table 7-3. Continue working with `stroll_tabbed.html` from the previous exercise.

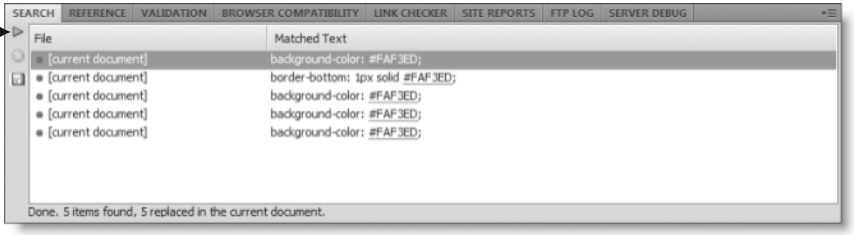
1. Select SpryTabbedPanels_stroll.css in the Related Files toolbar, and take a quick look at the rules it contains. In addition to copious comments describing the role of each selector, there are a lot of properties. Editing this style sheet with either the CSS Styles panel or the Code Navigator is a lot of work. Surely there's a simpler way? There most certainly is: the Find and Replace dialog box (see Chapter 5 for a detailed description of how to use it).
2. With SpryTabbedPanels_stroll.css still selected in the Related Files toolbar and the style sheet open in Split view, launch the Find and Replace dialog box (Edit ➤ Find and Replace or Ctrl+F/Cmd+F).
3. In the Find and Replace dialog box, set Find in to Current Document. The Search option will be grayed out because Source Code is the only option inside a style sheet. Enter #EEE in the Find field, and enter #FAF3ED in the Replace field. Make

sure all the options at the bottom of the dialog box are deselected. Your settings should look like this:



4. Click Replace All. The Results panel should open and report that it has made five substitutions, as shown in the next screenshot:

Click this arrow to relaunch Find and Replace



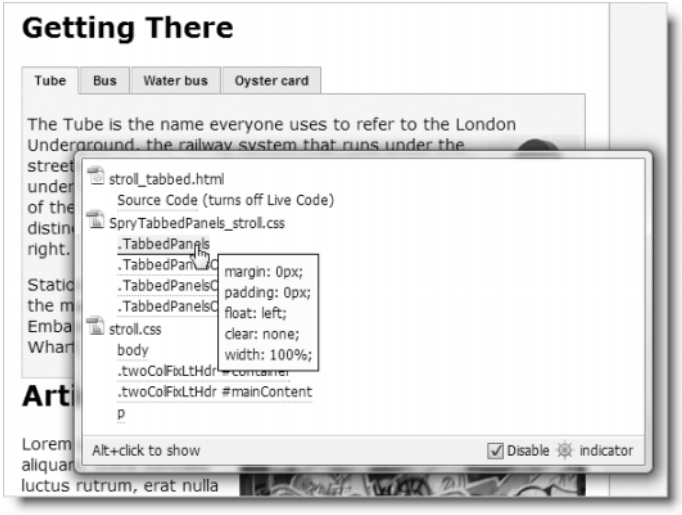
5. You now need to substitute the other colors. Use the right-facing green arrow at the top left of the Results panel, as indicated in the preceding screenshot, to relaunch the Find and Replace dialog box (the Search menu becomes selectable when you open the dialog box this way, but it should remain set to Source Code). Repeat steps 3 and 4 three times using the following values taken from Table 7-3:

- Find: #DDD Replace: #F2E1D2
- Find: #CCC Replace: #DFBD9F
- Find: #999 Replace: #C99466

6. Close the Results panel, and test the tabbed panels in Live view. They now look more in harmony with the page, but some fine-tuning still needs to be done to margins and padding.

7. The `TabbedPanels` class controls the horizontal and vertical space around the tabbed panels, as well as their overall width. As Figure 7-12 shows, there's no gap between the bottom of the panel and the following heading. So, you need to adjust the `margin` property.

Hold down the Alt key (or Opt+Cmd on a Mac), and click anywhere inside the tabbed panels to bring up the Code Navigator. Click the link for the `.TabbedPanels` selector, as shown here:



8. Change the `margin` property from `0px` to `0 0 15px 0`. This adds a 15-pixel margin on the bottom but leaves the other sides with a 0-pixel margin.

   If you want to constrain the width of the panels, this is where you should edit the `width` property. However, do *not* delete the `width` property, because it's required for the widget to display correctly in Internet Explorer.

9. Press F5 to refresh Design view. There should now be a nice offset between the bottom of the tabbed panels and the following heading.

10. Now let's improve the look of the text. Turn off Live view if it's still active, and bring up the Code Navigator by holding down Alt/Opt+Cmd and clicking in one of the tabs. Click the link for the third selector (`.TabbedPanelsTab`).

    The `font` property uses the shorthand version like this:

    ```
    font: bold 0.7em sans-serif
    ```

    Change it to this:

    ```
    font: bold 0.7em Verdana, Geneva, sans-serif;
    ```

    A neat way of doing this is to highlight sans-serif and press Ctrl+space (the combination is the same on Windows and Mac). This brings up code hints with a list of suggested font families, as shown here:

Use your down arrow key to select the fonts you want, and press Enter/Return to insert them. You can also double-click your choice, but this is trickier and often results in part of the original selection remaining in the code.

11. The one final improvement is to reduce the size of the text and add some horizontal padding to the paragraphs. Position your cursor anywhere in one of the paragraphs in the tabbed panels widget, and launch the New CSS Rule dialog box. If you use the icon at the bottom right of the CSS Styles panel or the CSS view of the Property inspector, the New CSS Rule dialog box suggests this horrendous dependent selector:

```
.twoColFixLtHdr #container #mainContent #TabbedPanels1 ➥
.TabbedPanelsContentGroup ➥
.TabbedPanelsContent.TabbedPanelsContentVisible p
```

Click the Less Specific button five times to reduce the selector to this:

```
.TabbedPanelsContent.TabbedPanelsContentVisible p
```

Then edit the Selector Name field manually to this:

```
.TabbedPanelsContent p
```

12. Set Rule Definition to SpryTabbedPanels_stroll.css, and click OK.
13. In the Type category of the CSS Rule Definition dialog box, set Size to 75%. Then select the Box category, deselect Same for all in the Padding section, set Right and Left to 10px, and click OK.
14. Test the page in Live view and a browser. The contents of the tabbed panels should now look more compact but with more breathing space on either side. If necessary, compare your style sheet with SpryTabbedPanels_stroll_horiz.css in the SpryAssets folder.

Converting to vertical tabs

The tabbed panels style sheet also contains a default set of rules that let you change the orientation of the tabs. Instead of running across the top, you can have them running down the left side of the panel. Table 7-4 describes the purpose of each selector.

Table 7-4. Style rules for vertical tabs

Selector	Type	Notes
.VTabbedPanels .TabbedPanelsTabGroup	Explicit	Vertical tabs are displayed in a column. This selector sets the background color, border, height, and width of the column. The height (default 20em) needs to be the same as in .VTabbedPanels .TabbedPanelsTabGroup. Don't use a pixel height unless the panels contain elements of fixed dimensions, such as images.

Selector	Type	Notes
`.VTabbedPanels .TabbedPanelsTab`	Explicit	Works in combination with `.TabbedPanelsTab`. Overrides top, left, and right borders, float, and margin. All other rules, such as background color and font, are preserved from the `.TabbedPanelsTab` class.
`.VTabbedPanels .TabbedPanelsTabSelected`	Dynamic	Overrides the background and bottom border colors of the selected tab. With horizontal tabs, the bottom border is set to the same color as the panel to create the illusion that the tab is part of the panel, but with vertical tabs, a solid bottom border is needed.
`.VTabbedPanels .TabbedPanelsTabGroup`	Explicit	Sets the height and width of the panels but inherits the background color and borders from the `.TabbedPanelsTabGroup` class.

These descendant selectors work in conjunction with the classes listed in Table 7-2, which control the basic colors. So, you need to perform steps 2–5 of the previous exercise to set the colors for vertical tabbed panels. The main problem with vertical tabs is the need to set a height, which must be sufficient to accommodate the content of the biggest panel. It should be specified in ems so that the panels can expand if the user increases the size of text in the browser. It is possible to omit the height to create a flexible layout, but the result doesn't look as good, as you'll see shortly.

Switching the orientation of tabbed panels

Let's convert the tabbed panels widget in `stroll_tabbed.html` to use vertical tabs. This time, I think it's easier to use the CSS Styles panel in All mode to change the style rules (using All mode was described in Chapter 4). Continue working with the same files as in the previous exercise.

1. Click anywhere in the tabbed panels widget in Design view, and select <div.TabbedPanels#TabbedPanels1> in the Tag selector at the bottom of the Document window. This is the main `<div>` that wraps around the tabbed panels widget. Right-click, and choose Set Class ➤ VTabbedPanels from the context menu, as shown in Figure 7-13.

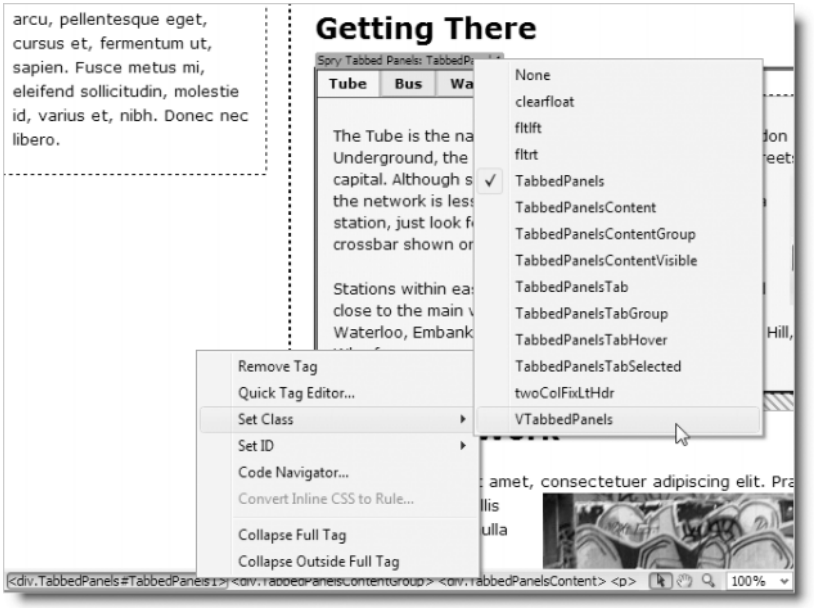
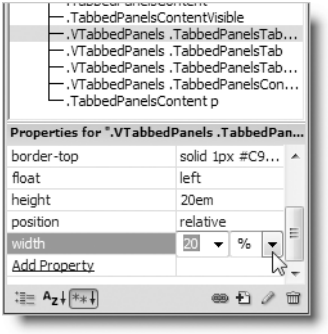


Figure 7-13. The first step in converting to vertical tabs is to change the class of the surrounding .

This changes the class of the `<div>` from `TabbedPanels` to `VTabbedPanels`, and the widget immediately inherits the default rules for vertical tabs. Because the default widths (`10em` + `30em`) are too great, the design falls apart completely in Design view.

2. Open the CSS Styles panel in All mode, and highlight the first vertical tab selector (`.VTabbedPanels .TabbedPanelsTabGroup`). Change the `width` property from 10em to 20%, as shown here:



3. Next highlight the final selector that controls vertical tabs (`.VTabbedPanels .TabbedPanelsContentGroup`), and change the `width` property from 30em to 78%. The widget springs back into shape. Choosing figures that add up to less than 100 percent avoids rounding errors. To display a web page, the browser needs to convert percentages to whole pixels. If it rounds up, floated content no longer fits and is pushed down the page, breaking your design.

4. Activate Live view, and test the tabbed panels. You'll probably notice two things: the fixed height makes the first panel (Tube) look rather bare, and there's hardly any gap between the bottom of the panel and the following headline. Click the fourth tab (Oyster Card), and you'll see that the contents of the panel spill out, as shown in Figure 7-14.

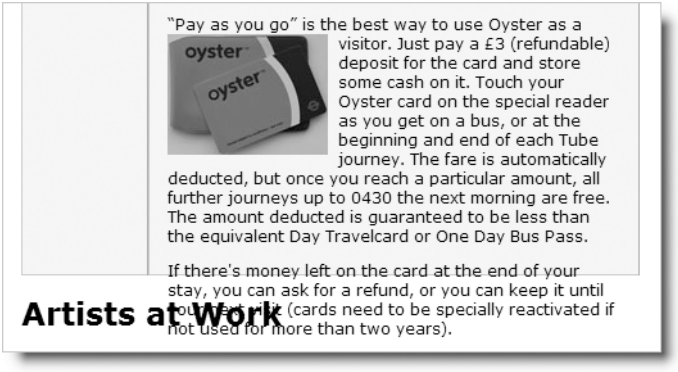


Figure 7-14. The danger with a fixed height is that text might spill out of the panel.

5. Fixing the gap between the tabbed panel widget and the next headline is easy. Add the `margin-bottom` property to the `.VTabbedPanels .TabbedPanelsContentGroup` selector, and set its value to 15px.

6. Dealing with the text overspill problem is not so easy. One solution is to change the `height` property of the `.VTabbedPanels .TabbedPanelsTabGroup` and `.VTabbedPanels .TabbedPanelsContentGroup` selectors to 23.5em. The problem with this is that the panels with less content begin to look decidedly empty.

7. The alternative is to remove the `height` property from both selectors. This causes each panel to expand or contract according to its contents. However, the background color of the column of tabs stretches down only as far as the last tab, as shown in Figure 7-15. You can't give a background color to the surrounding `<div>`, because both the tabs and panels are floated inside, so the `<div>` itself has no height.

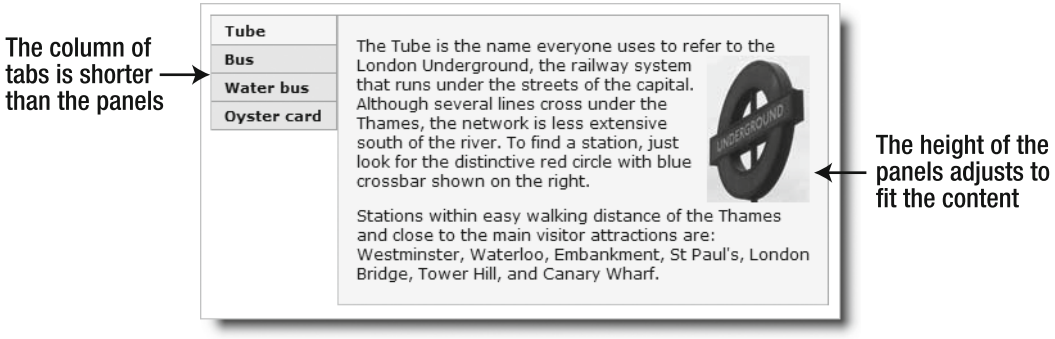


Figure 7-15. Varying amounts of content result in an uneasy compromise.

8. To revert to horizontal tabs, repeat step 1, changing the class back to `TabbedPanels`. Compare your style sheet with `SpryTabbedPanels_stroll_both.css` in the `SpryAssets` folder, if you need to check your own code. It contains the styles for both horizontal and vertical tabs.

Avoiding design problems with tabbed panels

As the previous exercise demonstrates, content overspill creates problems with the panels. You also need to take care with the tabs, because on a horizontal layout, they are floated left. If you make the labels too long, you might end up with the effect shown in Figure 7-16.

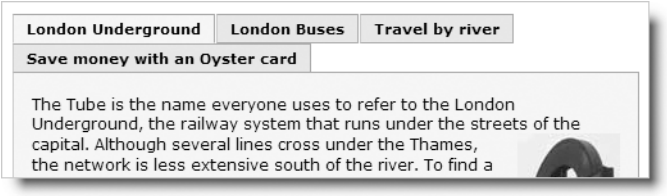


Figure 7-16. Too much content in the tabs breaks the design.

The result can look even more disastrous if you attempt to constrain the width of the tabs by setting a `width` property in the `.TabbedPanelsTab` class, as Figure 7-17 shows.

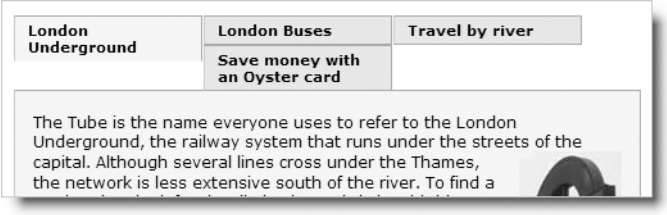


Figure 7-17. Setting a fixed width on the tabs leads to even more unpredictable results.

When using Spry tabbed panels, always keep the tab labels short. Don't try to get them to fit exactly across the top of the panels, because some visitors are likely to increase the text size, forcing one or more tabs to drop down in the same way as too much content does in Figure 7-16. In this sense, Spry tabbed panels aren't 100-percent bulletproof, but the original short labels (Tube, Bus, Water bus, and Oyster Card) don't cause any problem even when the largest font size is chosen in Internet Explorer. In Firefox, you need to increase the text size four times before the last tab slips down. Somebody who needs to make the text so large is unlikely to be concerned about design aesthetics. Still, if you are worried about overflow, you might consider adding the following properties to the `.TabbedPanelsTab` class:

```
max-width: /* less than total width divided by number of tabs */
white-space: nowrap;
overflow: hidden;
```

This keeps all the tabs on one line, regardless of how much the text is enlarged. The disadvantage is that the end of the label may be hidden if it's too long. Web pages cannot be controlled as rigidly as print, so you need to take into account the need for flexible design. Alternatively, avoid using design elements such as tabbed panels if you need to maintain pixel-perfect accuracy in your layout.

Using the accordion widget

The Spry accordion is another convenient way of storing a lot of information in a compact space. Figure 7-18 shows the same set of travel information as in the tabbed panels displayed in a Spry accordion. Instead of a tab, each panel has an individual title bar. When the user clicks the title bar of a closed panel, it glides open and simultaneously closes the panel that was previously open. By default, the panels are a fixed height and automatically display scrollbars if the content is too big. However, it's quite simple to change this so that the panels expand and contract in line with the content.

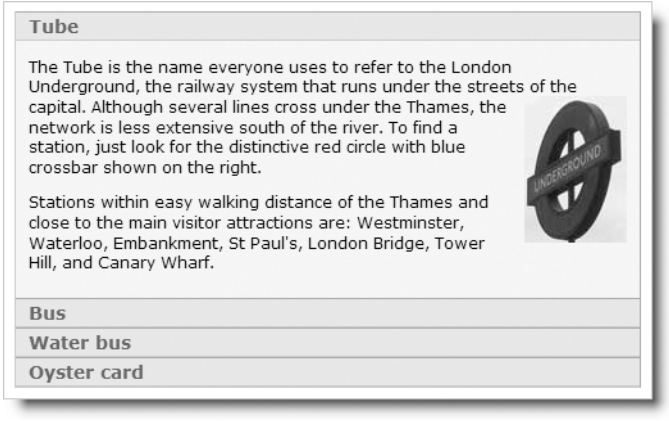


Figure 7-18. The accordion widget displays a series of interlinked panels one at a time.

Examining the structure of an accordion

To insert an accordion widget, click the Spry Accordion button on the Spry or Layout tab of the Insert bar. Alternatively, use the main menu: Insert ➤ Spry ➤ Spry Accordion.

Dreamweaver inserts a default two-panel accordion. The layout in Design view is very similar to the tabbed panels widget, and you access closed panels for editing in exactly the same way, by moving your mouse pointer over the right edge of the panel's title bar and clicking the eye icon.

The underlying HTML looks like this:

```
<div id="Accordion1" class="Accordion" tabindex="0">
  <div class="AccordionPanel">
    <div class="AccordionPanelTab">Label 1</div>
    <div class="AccordionPanelContent">Content 1</div>
  </div>
```

```
    <div class="AccordionPanel">
      <div class="AccordionPanelTab">Label 2</div>
      <div class="AccordionPanelContent">Content 2</div>
    </div>
  </div>
```

It's a simple structure consisting of a wrapper `<div>`, inside which each panel is a `<div>` with two more nested inside: one each for the title bar and the content panel. Like the tabbed panels widget, the use of `tabindex` makes the code technically invalid. Remove it from the opening `<div>` tag if W3C validation is a requirement, but doing so will disable keyboard navigation of the accordion.

All the styles are controlled by classes and descendant selectors, which are described in Table 7-5. As with Spry tabbed panels, some classes are declared explicitly in the HTML; others are generated dynamically by JavaScript.

Table 7-5. Style rules for the accordion widget

Selector	Type	Notes
`.Accordion`	Explicit	Sets all borders for the accordion, except for the top border, which is taken from the first title bar. Also sets `overflow` to `hidden` to prevent the content of hidden panels from being displayed. Add the `background-color` property to this rule if you want the panels to be shaded. By default, accordion widgets expand horizontally to fill all available space. Add the `width` property to this selector to constrain the space it occupies.
`.AccordionPanel`	Explicit	Eliminates padding and margin for each panel so the accordion displays as a single unit.
`.AccordionPanelTab`	Explicit	Sets the default background color and border of the title bar of each panel. The top border of the first title bar becomes the top border of the whole widget. Change this rule to style the text in the title bar. The nonstandard properties `-moz-user-select` and `-khtml-user-select` prevent users from selecting the title bar label in Mozilla, Firefox, and Konqueror browsers.

Selector	Type	Notes
.AccordionPanelContent	Explicit	Sets the height and overflow properties of the open panel. Change these properties if you want a different or flexible height. Do *not* change or delete the padding property, which is set to 0. Always add padding or margins to elements inside the accordion panel, rather than to the itself.
.AccordionPanelOpen .AccordionPanelTab	Dynamic	Sets the background color of the title bar for the currently open tab. However, this is overridden by later dynamic rules if the accordion has focus.
.AccordionPanelTabHover	Dynamic	Sets the background color of the title bar in rollover state.
.AccordionPanelOpen .AccordionPanelTabHover	Dynamic	Sets the background color of the title bar of the currently opened panel when the mouse rolls over the title bar.
.AccordionFocused .AccordionPanelTab	Dynamic	Sets the background color of the title bar of all panels when the accordion has focus.
.AccordionFocused .AccordionPanelOpen .AccordionPanelTab	Dynamic	Sets the background color of the title bar of the currently open panel when the accordion has focus.

Editing and styling a Spry accordion

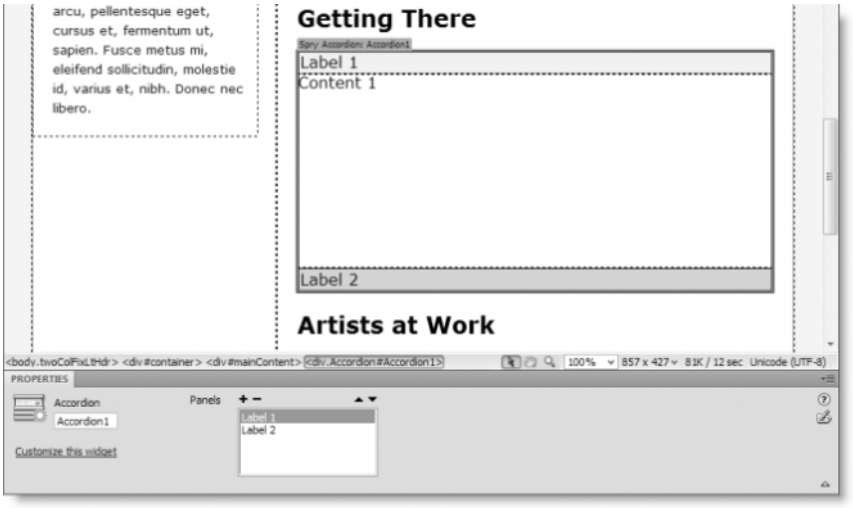
Although the structure of the accordion makes it relatively easy to style, the proliferation of dynamic classes and selectors can be confusing. It's easier to understand how they work through hands-on experimentation. So let's get to work.

Inserting the accordion and adding content

The following exercise is based on stroll.html, which you should have copied to your workfiles/ch07 folder for the Spry tabbed panels exercises earlier in the chapter. If you don't have the file, copy stroll.html and stroll.css from examples/ch07 to workfiles/ch07. Update links if prompted by Dreamweaver.

1. Open stroll.html in the Document window, and save it as stroll_accordion.html.
2. Create the new level 2 heading Getting There just above the Artists at Work heading.

3. With your cursor at the end of the new heading, click the Spry Accordion button on the Spry or Layout tab of the Insert bar. The page should look like this:



The Property inspector for a Spry accordion has very few options (hover your mouse pointer over the accordion in Design view, and click the Spry Accordion tab at the top left, if the Property inspector is showing something else). Dreamweaver automatically assigns `Accordion1` as the ID of the first accordion in a page and numbers subsequent instances `Accordion2` and so on. The Property inspector displays the ID in the field on the left, where you can change it if you want. The only other options are to add, remove, and reorder panels using the plus, minus, and arrow buttons. Clicking Customize this widget opens Dreamweaver Help at the page with details of the style rules that control an accordion.

4. Save `stroll_accordion.html`, and click OK to copy the dependent files.
5. Rename `SpryAccordion.css` in the Spry assets folder as `SpryAccordion_stroll.css`, and update the links when prompted. Since the web page contains style rules embedded in conditional comments, move the link to `SpryAccordion_stroll.css` from just before the closing `</head>` tag to above the conditional comments.
6. You edit an accordion in the same way as a tabbed panels widget. The only difference is that instead of Tab 1, and so on, the accordion uses Label 1, and so on. Follow steps 6 through 11 of "Inserting and editing a tabbed panels widget" to populate the accordion with four panels labeled Tube, Bus, Water bus, and Oyster Card. Because the title bar of each panel stretches the full width of the accordion, the eye icon that opens closed panels is much further to the right than in the tabbed panels.

7. When you paste the text into some panels, the end appears to be cut off. This is because the default styles set a height of 200 pixels and hide the overflow. To display the accordion content for editing when this happens, double-click inside one of the panels that have an overflow (sometimes you need to double-click twice). Alternatively, right-click, and select Element View ➤ Full from the context menu. This expands the whole accordion in Design view.

   With the accordion fully expanded, insert `underground.jpg` in the first panel and `oystercard.jpg` in the fourth panel, and apply the `fltrt` and `fltlft` classes to them, respectively (see steps 12 through 14 of "Inserting and editing a tabbed panels widget").

8. To collapse the accordion after editing, press F5, or right-click and select Element View ➤ Hidden from the context menu.

9. Activate Live view, and test the accordion panels. You'll notice that the panels are all the same height, and a vertical scrollbar appears inside each one. The colors are the same neutral grays as in the tabbed panels widget.

10. Save `stroll_accordion.html`, and press F12/Opt+F12 to preview the page in a browser. Use the Tab key to shift focus to the accordion. As soon as it has focus, the color of the title bars changes from neutral grays to rather ghastly shades of blue. This is the effect of the last two selectors listed in Table 7-5.

    We'll sort out the colors next, but first press the down arrow on your keyboard. As long as you haven't removed the `tabindex`, the next panel should glide open, closing the previous one behind it. While the accordion has focus, you can navigate through the panels in sequence with the up and down keyboard arrows. Alternatively, you can click any title bar to open a particular panel. Click anywhere outside the accordion and the colors revert to gray.

> *The different colors should serve as an important reminder that you should always test your pages in a browser—and preferably all the main browsers—before deploying a site on the Internet. Live view speeds up the process of development, but it's no substitute for the real thing when it comes to judging what your site will really look like.*

Changing the default colors of an accordion

The following instructions show you how to change the colors of `stroll_accordion.html` from the preceding exercise, but they apply equally to any accordion. Just use your own colors in place of those suggested here. The color scheme I have used is essentially the same as for the tabbed panels. Table 7-6 summarizes the default background colors and my replacements. The default style sheet uses keywords rather than hexadecimal notation for some colors.

Table 7-6. Background colors used in the accordion widget

Default color	Replacement	Applies to
Gray (gray)	Light brown (#DFBD9F)	Lighter borders
Black (black)	Russet (#C99466)	Darker borders
Medium gray (#CCCCCC)	Dark pink (#F2E1D2)	Closed title bar
Pale gray (#EEEEEE)	Dark pink (#F2E1D2)	Open title bar
Royal blue (#3399FF)	None	Closed title bar with focus
Sky blue (#33CCFF)	None	Open title bar with focus

1. Select SpryAccordion_stroll.css in the Related Files toolbar, and launch the Find and Replace dialog box (Edit ➤ Find and Replace or Ctrl+F/Cmd+F). Replace the first four colors listed in Table 7-6 in the same way as you did with the tabbed panels widget earlier in this chapter.
2. Close the Results panel, scroll down to the bottom of SpryAccordion_stroll.css, and locate the following section:

```
108 .AccordionFocused .AccordionPanelTab {
109     background-color: #3399FF;
110 }
111
112 /* This is an example of how to change the appearance of
    the panel tab that is
113  * currently open when the Accordion has focus.
114  */
115 .AccordionFocused .AccordionPanelOpen .AccordionPanelTab
    {
116     background-color: #33CCFF;
117 }
```

3. Delete the background-color properties and values shown on lines 109 and 116 of the preceding screenshot. This leaves both style rules empty. I have left them like this in case you decide you want to add different colors to indicate when the accordion has focus. Of course, if you don't want a visual indication that the accordion has focus, you can delete these two rules in their entirety.
4. Save the style sheet, and load stroll_accordion.html into a browser. When you test the accordion, the colors no longer clash with the rest of the page, but the styles could still do with some improvement.

   Currently, the panels have no background color, and the text in the title bars needs to look a bit more substantial.
5. I'll leave it up to you whether to make the remaining changes directly in the style sheet or in the CSS Styles panel in All mode. The important thing here is to understand which rules you're changing and why.

To give the panels a background color, add the background-color property to the .Accordion selector, and set it to #FAF3ED (light pink).

6. The .AccordionPanelTab selector styles the tab or title bar of each panel, so this is where you can make changes to the text in the title bars. Add the following properties and values:

```
font-family: Verdana, Geneva, sans-serif;
font-size: 90%;
font-weight: bold;
color: #555;
```

7. The text could also do with a bit of horizontal space, so change the value of the padding property in the .AccordionPanelTab selector from 2px to 2px **10px**. This gives 2 pixels of padding top and bottom and 10 pixels on either side.

8. The .AccordionPanelTabHover selector controls the rollover state of the title bars, but only when the accordion doesn't have focus. Change the color property to a slightly darker gray (#333). Also add the background-color property, and set it to #ECD3BD (dusky pink). This keeps the rollover color in harmony with the rest of the accordion when the focus is elsewhere in the page.

9. Give the next selector (.AccordionPanelOpen .AccordionPanelTabHover) the same values as in step 8. This makes the rollover colors the same, regardless of whether the accordion has focus.

10. One final change: because you cannot add padding to the AccordionPanelTab class, it's a good idea to create a new rule for .AccordionPanelContent p. By this stage, I expect you should have sufficient experience of creating new style rules. Define it in SpryAccordion_stroll.css using the following properties and values:

```
font-size: 75%;
padding-left: 10px;
padding-right: 10px;
```

This makes the text slightly smaller than in the rest of the page and gives 10 pixels breathing space on either side of the paragraphs inside the accordion. You can check your code against SpryAccordion_stroll_done.css in the SpryAssets folder.

The smaller font size created by the final change to the default styles removes the vertical scrollbar from all except the last panel. In the next chapter, I'll show you how to tweak the settings of an accordion so that the panels expand and contract to fit the content in the same way as the tabbed panels. You can't do it with CSS alone; you need to get your hands dirty (not very) with the Spry JavaScript code.

Using collapsible panels

Collapsible panels are very similar to the accordion. In fact, they look identical to an accordion if you use several of them in succession. The difference is that each panel is separately controlled, so they can be all open, all closed, or any combination in between.

Examining the structure of a collapsible panel

To insert a collapsible panel, click the Spry Collapsible Panel button in the Spry or Layout tab of the Insert bar. Alternatively, use the menu option: Insert ➤ Spry ➤ Spry Collapsible Panel. This inserts a default collapsible panel (see Figure 7-19) at the current insertion point of the page.

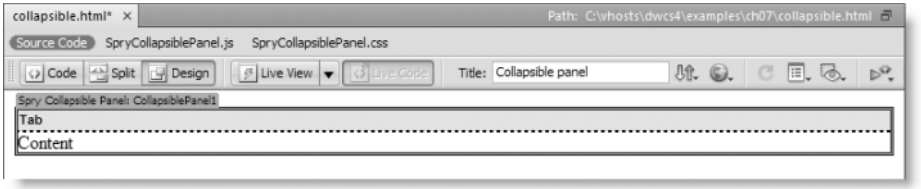


Figure 7-19. A collapsible panel consists of a single tab and content area.

The underlying HTML is extremely simple: a `<div>` for the tab and another for the content, both nested in a wrapper `<div>` like this:

```
<div id="CollapsiblePanel1" class="CollapsiblePanel">
  <div class="CollapsiblePanelTab" tabindex="0">Tab</div>
  <div class="CollapsiblePanelContent">Content</div>
</div>
```

This simple structure makes for equally simple CSS styling. Table 7-7 lists the default selectors. As with the tabbed panels and accordion widgets, the use of `tabindex` is technically invalid but is a compromise to make the panels accessible through keyboard navigation.

Table 7-7. Style rules for the collapsible panel widget

Selector	Type	Notes
`.CollapsiblePanel`	Explicit	This zeros margin and padding on the widget and sets a light-colored border on the left and bottom and a darker-colored on the right and top. By default, collapsible panels expand horizontally to fill the available space, so set a width here if required. Set a background color for the panel here.
`.CollapsiblePanelTab`	Explicit	This styles the tab. Only the bottom border is set, as the top, left, and right border styles come from the preceding selector. Change this rule to style the text in the title bar. The nonstandard properties `-moz-user-select` and `-khtml-user-select` prevent users from selecting the title bar label in Mozilla, Firefox, and Konqueror browsers.
`.CollapsiblePanelContent`	Explicit	This zeros padding and margins. Do *not* change or delete the `padding` property. Always add padding or margins to elements inside the panel, rather than to the `<div>` itself.

Selector	Type	Notes
`.CollapsiblePanelTab a`	Explicit	This doesn't actively affect the widget in its default state. If you put a dummy link around the text in a tab, this style rule limits the focus lines around the text, rather than around the entire tab.
`.CollapsiblePanelOpen .CollapsiblePanelTab`	Dynamic	Sets the background color of the tab when the panel is open.
`.CollapsiblePanelTabHover, .CollapsiblePanelOpen, .CollapsiblePanelTabHover`	Dynamic	Sets the background color of the tab in rollover state.
`.CollapsiblePanelFocused .CollapsiblePanelTab`	Dynamic	Sets the background color of the tab when the panel has focus.

Editing and styling collapsible panels

When you insert a collapsible panel widget, it's open by default, ready for editing. However, since you can have collapsible panels open and closed in any combination, the options in the Property inspector need a little explanation. As you can see in Figure 7-20, there are two drop-down menus that are set to Open by default. The first one—labeled Display—controls whether the content of the collapsible panel is visible in Design view. The second—labeled Default state—controls whether the panel is open or closed when the web page first loads.

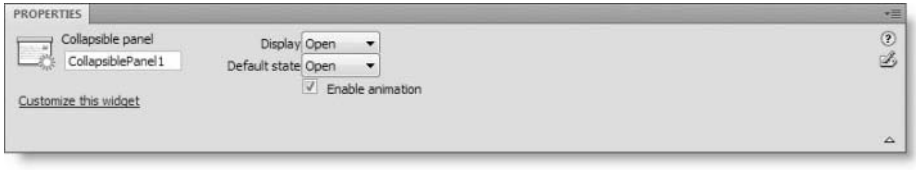


Figure 7-20. Two settings control the state of a collapsible panel—one for Design view, the other for the web page.

The Display setting is purely for your convenience when editing the page in Dreamweaver. If you set Default state to Closed, the panel is closed when the page first loads into a browser window.

The Enable animation option at the bottom of the Property inspector is checked by default. If you deselect it, the collapsible panel snaps open and closed, rather than gliding.

If you have more than one collapsible panel on a page, Dreamweaver initializes each one independently, so you need to set the options individually for each panel. There is no way of setting global options for all panels on a page.

Since collapsible panels are so similar to the Spry accordion, I won't give step-by-step instructions for inserting and editing them. Table 7-8 lists the default colors used in SpryCollapsiblePanel.css together with the substitutes I used to fit the color scheme in the exercise file that we have been using throughout this chapter.

Table 7-8. Background colors used in collapsible panels

Default color	Replacement	Applies to
Pale gray (#EEE)	Light pink (#FAF3ED)	Open tab
Light gray (#DDD)	Dark pink (#F2E1D2)	Tab
Medium gray (#CCC)	Light brown (#DFBD9F)	Tab on rollover and light borders
Dark gray (#999)	Russet (#C99466)	Dark borders
Royal blue (#3399FF)	Dusky pink (#ECD3BD)	Tab on focus

All the other changes follow the same pattern as for an accordion. To give the panels a background color, add a background-color property to the .CollapsiblePanel selector. I used #FAF3ED (light pink).

To make the text in the tabs look more substantial, I added the same rules to .CollapsiblePanelTab as I did with the accordion tabs, namely:

```
color: #555;
font-family: Verdana, Geneva, sans-serif;
font-size: 90%;
font-weight: bold;
```

To ensure the text in the tabs stands out on rollover, add a color property to .CollapsiblePanelTabHover, .CollapsiblePanelOpen .CollapsiblePanelTabHover. I used #333.

Finally. because you cannot add padding to the CollapsiblePanelTab class, it's a good idea to create a new rule for .CollapsiblePanelContent p in the same way as for the accordion. I used the following properties and values:

```
font-size: 75%;
padding: 5px 10px;
margin: 0;
```

You can see the finished result in stroll_collapsible.html. The amended style sheet is in SpryCollapsiblePanel_stroll.css in the SpryAssets folder.

Creating tooltips with Spry

The Spry Tooltip widget is new to Dreamweaver CS4. It makes it very easy to add an extended tooltip to a page element. Figure 7-21 shows a Spry Tooltip attached to one of the images in `stroll.html`, but any page element can be used as a trigger.

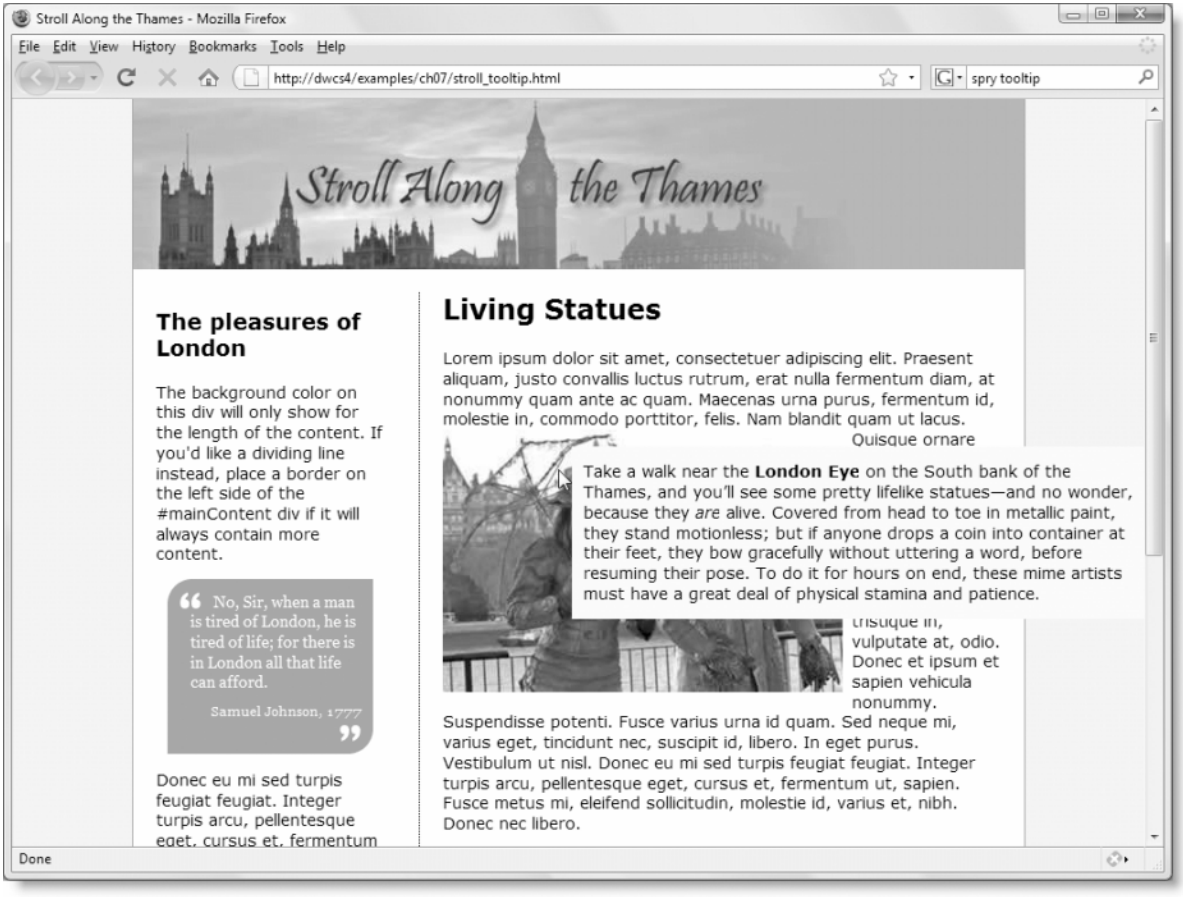


Figure 7-21. Tooltips are a good way of adding supplementary information to a page.

Examining the structure of a Spry tooltip

To insert a Spry Tooltip, click the Spry Tooltip icon on the Spry tab of the Insert bar or use the menu option: Insert ➤ Spry ➤ Spry Tooltip.

A Spry Tooltip consists of two parts: the trigger element and the tooltip. The tooltip is always a `<div>`, which Dreamweaver places at the end of the document, just before the

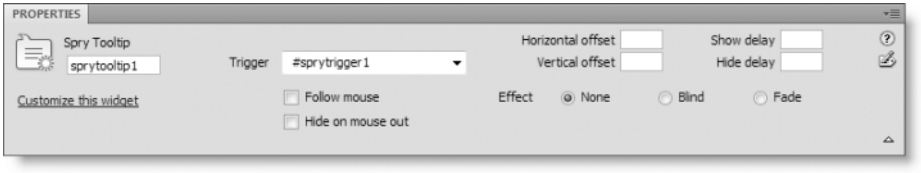
closing </body> tag (although you can move this later). The trigger depends on what, if anything, is selected in the page, as follows:

- If nothing is selected, the trigger consists of placeholder text wrapped in a and inserted at the current insertion point.
- If an HTML element, such as an image, paragraph, or , is selected, Dreamweaver adds an id attribute to the element's tag to associate it with the tooltip.
- If the element already has an id attribute, the existing ID is preserved and used to associate the element with the tooltip.
- If only part of a text element is selected, the selected portion is wrapped in a , and an id attribute is added to the tag.

Like all Spry widgets, the Spry Tooltip relies on an external JavaScript file (SpryTooltip.js) and a style sheet (SpryTooltip.css). In contrast with the other UI components, the style sheet is extremely simple. It contains just two selectors: .iframeTooltip and .tooltipContent. The first selector is a hack that overcomes a problem with Internet Explorer and shouldn't be altered. The .tooltipContent selector contains just a single property: the background color of the tooltip, which is set to light yellow (#FFFFCC).

Inserting and styling tooltips

The following screenshot shows the Property inspector for a Spry Tooltip:



The field on the left of the Property inspector displays the ID of the selected tooltip . Dreamweaver automatically assigns this value, incrementing the number for each tooltip added to the page. You can change this value, but you must ensure that the same ID is not used elsewhere on the same page.

The other options are as follows:

- Trigger: This displays the ID of the trigger element. You can change the trigger element by clicking the arrow on the right of the field to reveal a list of all IDs on the current page, except that of the tooltip .
- Follow mouse: Selecting this checkbox makes the tooltip follow the mouse within the bounds of the trigger element.
- Hide on mouse out: By default, tooltips are hidden when the mouse moves off the trigger element. Selecting this option keeps the tooltip open as long as the mouse remains over the tooltip, even if it's no longer over the trigger element. See the following exercise for a practical example.

- Horizontal/Vertical offset: By default, the top-left corner of the tooltip is displayed 20 pixels to the right and below the cursor. To set a different offset, enter the desired values (use only numbers without px) in these fields. Negative numbers move the tooltip above and to the left of the cursor. Positive numbers move it below and to the right. The new offset is calculated from the current position of the cursor, not from the default top left of the tooltip.
- Show/Hide delay: These fields let you delay the appearance or disappearance of the tooltip by a specified number of milliseconds (1000 = 1 second).
- Effect: This determines how the tooltip is displayed. There are three options:
  - None: The whole tooltip appears or is hidden immediately.
  - Blind: The tooltip is revealed from the top. When being hidden, it disappears from the bottom. This effect sometimes results in the background being drawn too big or too small. Test it thoroughly in your target browsers.
  - Fade: This fades the tooltip in or out.

Applying tooltips to text and images

This exercise shows you how to add tooltips to various elements in the page that has been used throughout this chapter. It also examines how the tooltips are dynamically generated.

1. Save `stroll.html` as `stroll_tooltip.html` in `exercises/ch07`.
2. Click anywhere in the sidebar. Then select <div#sidebar1> in the Tag selector at the bottom of the Document window. This selects the whole `<div>`.
3. Insert a Spry Tooltip using either the Insert bar or the Insert menu. Although it looks as though nothing happened, scroll down to the bottom of the page, and you should see that the tooltip `<div>` has been inserted below the footer, as shown in Figure 7-22. Notice that the value of the Trigger field is #sidebar1, indicating that Dreamweaver has used the existing `id` attribute of the selected element.

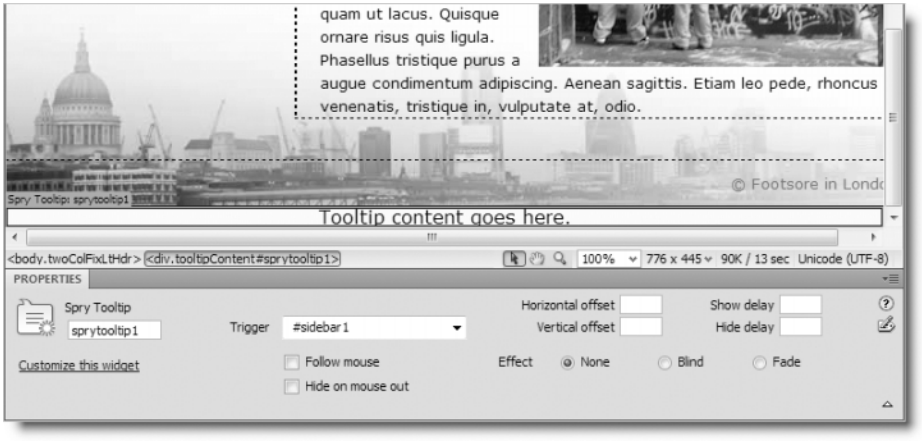
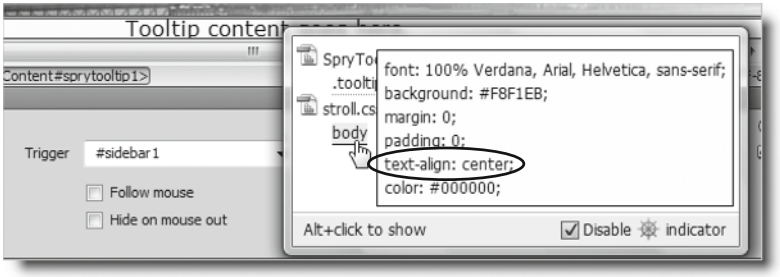


Figure 7-22. The tooltip is always placed at the foot of the page.

4. Save `stroll_tooltip.html` to copy the dependent files to the site's Spry assets folder.
5. Rename `SpryTooltip.css` to `SpryTooltip_stroll.css`, and update the link to `stroll_tooltip.html` when prompted.
6. As you can see in Figure 7-22, the placeholder text in the tooltip is centered. To understand why, hold down the Alt key (or Opt+Cmd), and click inside the tooltip `<div>`. The Code Navigator displays two CSS selectors: .tooltipContent and body. Mouse over them to inspect the style rules they define. You'll see that the `.tooltipContent` selector specifies only a background color. It's the body selector that's centering the text, as the following screenshot shows:

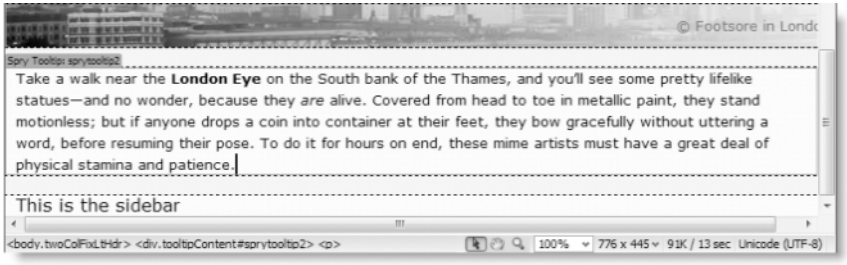


   If you read the comments in `stroll.css`, you'll see that it's to center the `container <div>` in Internet Explorer 5. The text in the `container <div>` is realigned in a separate rule. However, as I explained before, Dreamweaver inserts the tooltip `<div>` just before the closing `</body>` tag, so it's outside the `container <div>` and not affected by its style rules. Although Internet Explorer 5 is becoming increasingly rare, rather than risk problems, it's better to change the text alignment for the tooltip.
7. Select SpryTooltip_stroll.css in the Related Files toolbar. and amend the `.tooltipContent` style rule by adding the `text-align` and `padding` properties like this:

```
.tooltipContent
{
  background-color: #FFFFCC;
  text-align: left;
  padding: 0 10px;
}
```

8. Replace the placeholder text with something relatively short like This is the sidebar.
9. Select `living_statues.jpg` in Design view, and apply a Spry Tooltip. Scroll down to the bottom of the page. You'll see that the new tooltip `<div>` has been inserted below the footer but above the previous tooltip `<div>`. Check the value of Trigger in the Property inspector. The image didn't have an ID, so Dreamweaver has automatically assigned #sprytrigger1.

10. Replace the placeholder text in the new tooltip with a paragraph containing several sentences. You can use the text in tooltip.doc or tooltip.docx in examples/ch07. The bottom of the page should now look like this in Design view (the size of the text in the second tooltip is smaller because it's in a paragraph):



11. Save both stroll_tooltip.html and its style sheet. Then preview the page in a browser. As you move your mouse over the sidebar, the first tooltip should appear and disappear again when you move your mouse away.

12. Now, mouse over the image of the living statues. Move your mouse from the top left of the image down toward the bottom right. As the mouse reaches the top left corner of the tooltip, the tooltip jumps to maintain its distance and keeps on doing so until you're no longer over the image.

13. Return to Dreamweaver, and select the second tooltip (click the turquoise tab at the top left to bring up its details in the Property inspector). Select the Hide on mouse out option.

14. Save the page, and view it again in the browser. Mouse over the image from the top left again. This time, the tooltip remains in its original position until your mouse leaves both the image and the tooltip.

15. I'll leave you to experiment with other options, but finally let's examine how the browser displays the tooltip. Back in Dreamweaver, switch to Split view, and select Source Code in the Related Files toolbar. Press F4 to hide the panels if you need more room to see both the underlying HTML and part of the page in Design view. You don't need to see the whole page, but just enough to be able to mouse over the image of the living statues.

16. Activate Live view, and mouse over the image to display the tooltip. Now press F6 to freeze the JavaScript. This lets you move your mouse off the image to click the Live Code button in the Document toolbar (or select View ➤ Live Code). The Code view section of the Document window turns yellow, indicating that it's showing the dynamically generated code.

17. Scroll down the Code view section of the Document window until you find the beginning of the tooltip . It should look like this:

```
49  <div class="tooltipContent" id="sprytooltip2" style="
    position: absolute; z-index: 9999; display: block; left:
    289px; top: 127px; visibility: visible; ">
50    <p>Take a walk near the <strong>London Eye</strong> on
    the South bank of the Thames, and you'll see some pretty
```

Notice that it has an inline `style` attribute that converts the `<div>` into an absolutely positioned element. This is dynamically generated by the Spry Tooltip's external JavaScript file.

When the page first loads, the JavaScript sets the `visibility` property to `none`, hiding it from view. Passing the mouse over the image triggers a JavaScript event that manipulates the DOM. Live Code shows you the information being passed to the browser to display the tooltip.

18. Turn off Live Code by clicking the Live Code button or selecting View ➤ Live Code. The same section of code should now look like this:

```
53  <div class="tooltipContent" id="sprytooltip2">
54    <p>Take a walk near the <strong>London Eye</strong> on
the South bank of the Thames, and you'll see some pretty
```

The inline `style` attribute has gone. Also, the line numbers have changed. This reflects the fact that the DOM is no longer being manipulated by JavaScript.

Like all Spry widgets, the tooltip degrades gracefully when JavaScript is turned off. The content of each tooltip `<div>` is displayed as ordinary text. Dreamweaver locates them at the bottom of the page, but you can put them anywhere in the page as long as it's a valid place to locate a `<div>`. After all, as far as HTML is concerned, a tooltip `<div>` is no different from any other. It's the JavaScript that converts it temporarily into an absolutely positioned element.

To preserve the logical flow of the page for search engines and anyone browsing without JavaScript enabled, it's a good idea to move each tooltip `<div>` to the section of the page to which it refers. However, you should always do this in Code view.

> *Do not use Design view to move a tooltip* `<div>` *to a different part of the page. When you cut the* `<div>`*, it also removes a vital part of the JavaScript code, which is not restored when you paste the* `<div>` *in its new location.*

Removing a Spry widget

Removing a Spry widget is very easy: just click the turquoise tab at the top left of the widget, and press Delete. Dreamweaver removes both the widget and its associated JavaScript. However, if you have renamed the style sheet (as in the exercises in this chapter), the link to the style sheet *isn't* removed. Dreamweaver removes only style sheets that retain the default name.

Although this sounds simple and convenient, it comes with a big downside: *removing a widget also removes all its contents*. So, think carefully before pressing Delete. Do you need to display the contents in some other format? If so, make sure you have a copy before blasting everything to cyberoblivion.

Chapter review

In this chapter, I've given you an in-depth look at Spry effects and user interface components. The effects are easy to use: just a couple of clicks, and you're done. The user interface components are also easy to insert into a page. The difficulty comes with styling them to fit in with the rest of your design. However, once you have worked out a set of colors to replace the defaults, you can customize them fairly quickly.

As I said at the beginning of the chapter, you don't even need Dreamweaver to use Spry. You could, in fact, implement everything in this chapter by downloading the external JavaScript files of the Spry framework and hand-coding a few lines to embed the effects and widgets in your page. There's no doubt that Dreamweaver lightens the load when using these features, but it can't automate everything. For example, you can't create a link to open an accordion or tabbed panel without learning about the Spry application programming interface (API) and diving into Code view. Dreamweaver code hints for Spry, as well as improved support for the DOM and other JavaScript libraries, make this easier. So, in the next chapter, I invite you to roll up your sleeves and get closer to the code.