9 BUILDING ONLINE FORMS AND VALIDATING INPUT

Online forms are the gateway to the server and lie at the very heart of working with PHP, the focus of most of the remaining chapters. You use forms for logging into restricted pages, registering new users, placing orders with online stores, entering and updating information in a database, and sending feedback by email. But gateways need protection.

You need to filter out incomplete or wrong information: a form isn't much use if users forget to fill in required fields or enter an impossible phone number. It's also important to make sure that user input doesn't corrupt your database or turn your website into a spam relay. That's what **input validation** is all about—checking that user input is safe and meets your requirements. This is different from validating your HTML or CSS against W3C standards, and it's much more important because it protects your data.

Validating user input is a theme that will run through much of the rest of this book. In this chapter, we'll look at client-side validation with the assistance of Spry. Then, in Chapter 11, I'll show you how to process the form and validate its content on the server with PHP. Server-side validation is more important, because it's possible for users to evade client-side filters. Even so, client-side validation is useful for catching errors before a form is submitted, improving user experience.

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

- Creating a PHP page
- Creating forms to gather user input
- Understanding the difference between GET and POST
- Passing information through a hidden form field
- Making online forms accessible
- Using Spry widgets to validate input
- Displaying and controlling the number of characters in a text area

Building a simple feedback form

All the components for building forms are on the Forms tab of the Insert bar. They're also on the Form submenu of the Insert menu, but for the sake of brevity, I'll refer only to the Insert bar in this chapter.

Most form elements use the `<input>` tag, with their function and look controlled by the type attribute. The exceptions are the multiline text area, which uses the `<textarea>` tag, and drop-down menu and scrollable lists, which use the `<select>` tag. Dreamweaver handles all the coding for you, but you need to dive into Code view frequently when working with forms and PHP, so if your knowledge of the tags and attributes is a bit rusty, brush it up with a good primer, such as *HTML and CSS Web Standards Solutions: A Web Standardista's Approach* by Christopher Murphy and Nicklas Persson (friends of ED, ISBN: 978-1-43021-606-3).

Choosing the right page type

HTML contains all the necessary tags to construct a form, but it doesn't provide any means to process the form when submitted. For that, you need a server-side solution, such as

PHP. In the past, you may have used FormMail or a similar script to send the contents of a form by email. Such scripts normally reside in a directory called `cgi-bin` and work with `.html` pages. The `action` attribute in the opening `<form>` tag tells the form where to send the contents for processing. It usually looks something like this:

```
<form id="sendcomments" method="post" action="/cgi-bin/formmail.cgi">
```

You can do the same with PHP: build the form in an `.html` page, and send the contents to an external PHP script for processing. However, it's far more efficient to put the form in a page with a `.php` file name extension and use the same page to process the form. This makes it a lot easier to redisplay the contents with error messages if any problems are found. So, from now on, we'll start using PHP pages. Before going any further, you should have specified a PHP testing server, as described in Chapter 2.

Creating a PHP page

You can create a PHP page in Dreamweaver in several ways, namely:

- Select Create New ➤ PHP in the Dreamweaver welcome screen.
- Select File ➤ New to open the New Document dialog box, and select Blank Page and PHP as the Page Type. As Figure 9-1 shows, this offers the same choice of CSS layouts as an HTML page. Click Create when you have made your selection.
- Right-click in the Files panel, and select New File. If you have defined a PHP testing server, Dreamweaver creates a default blank page with a `.php` file name extension.
- Change the file name extension of an existing page to `.php` in the Files panel or Save As dialog box.

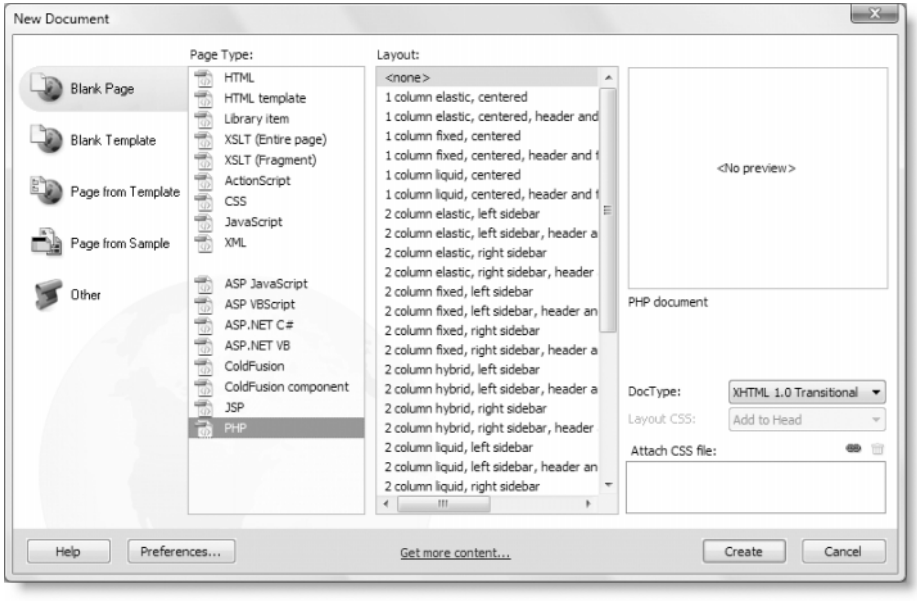


Figure 9-1. You have access to the same wide range of CSS layouts for a PHP page as for an HTML one.

The file name extension is the only difference between a blank PHP page and an HTML one. If you switch to Code view, you'll see the same DOCTYPE declaration and HTML tags. The .php extension tells the server to send the page to the PHP engine for processing before sending it to the browser.

Mixing .php and .html pages in a site

It's perfectly acceptable to mix .html and .php files in the same site. However, when building a new site, it's a good idea to create all pages with a .php extension, even if they don't contain dynamic code. That way, you can always add dynamic content to a page without needing to redirect visitors from an .html page. If you are converting an old site, you can leave the main home page as a static page and use it to link to your PHP pages.

A lot of people ask whether you can treat .html (or any other file name extension) as PHP. The answer is yes, but it's not recommended, because it places an unnecessary burden on the server and makes the site less portable. Also, reconfiguring Dreamweaver to treat .html files as PHP is messy and inconvenient.

Inserting a form in a page

It's time to get to work and build a feedback form. To concentrate on how the form is validated and processed, let's work in a blank page and keep the styling to a minimum.

Building the basic form

The final code for this page is in feedback.php in examples/ch09.

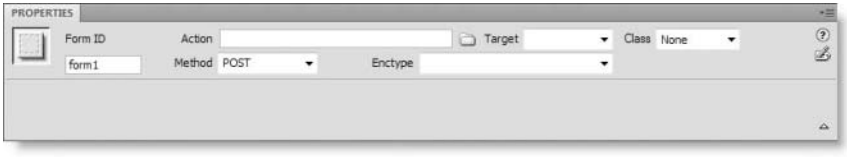
1. Create a new PHP page as described in the previous section, and save it in workfiles/ch09 as feedback.php. If you use the New Document dialog box, set Layout to <none>, and make sure no style sheets are listed under Attach CSS file.
2. Add a heading, followed by a short paragraph. Make sure you're in Design view or, if Split view is open, that the focus is in Design view. Inserting a form is completely different when the focus is in Code view, as explained in "Inserting a form in Code view" later. With the insertion point at the end of the paragraph, click the Form button in the Forms tab of the Insert bar. It's the first item, as shown here:

3. This inserts the opening and closing <form> tags in the underlying code. In Design view, the form is surrounded by a red dashed line, as shown in the next screenshot:

All form elements must be inserted inside the red line, so don't click anywhere else in Design view. Otherwise, you might end up outside the form. Of course, once you start inserting form elements, the boundary expands to accommodate the content.

> *If you try to insert a form element outside the dashed red line, Dreamweaver asks you whether you want to insert a form tag. Unless you want to create two separate forms, this is normally an indication that your insertion point is in the wrong place. Although you can have as many forms as you like on a page, each one is treated separately. When a user clicks a form's submit button, only information in the same form is processed; all other forms are ignored.*

4. The Property inspector displays the form's settings, as shown here:



Dreamweaver gives forms a generic name followed by a number. This is applied to both the `name` and `id` attributes in the underlying code. If you change the value in the Form ID field of the Property inspector, Dreamweaver updates both attributes.

The Action field is where you enter the path of the script that processes the form. Since this will be a self-processing form, leave the field empty.

The Method menu has three options: Default, GET, and POST. This determines how the form sends data to the processing script. Leave the setting on POST. I'll explain the difference between GET and POST shortly. If you select the Default option, Dreamweaver omits the `method` attribute from the `<form>` tag. This results in the form behaving the same way as if you had selected GET. I recommend against using it, because you're less likely to make mistakes by selecting GET or POST explicitly.

You can ignore the Target and Enctype options. Target should normally be used only with frames, and Dreamweaver automatically selects the correct value for Enctype if required. The only time it needs a value is for uploading files. Dreamweaver server behaviors don't handle file uploads. See my book *PHP Solutions: Dynamic Web Design Made Easy* (friends of ED, ISBN: 978-1-59059-731-6) for details of how to do it by hand-coding.

Inserting a form in Code view

If you insert a form in Code view or in Split view with the focus in Code view, Dreamweaver displays the Tag Editor (see Figure 9-2). This offers the same options as the Property inspector, but you need to fill in all the details yourself. Inserting a form in Design view is much more user-friendly.

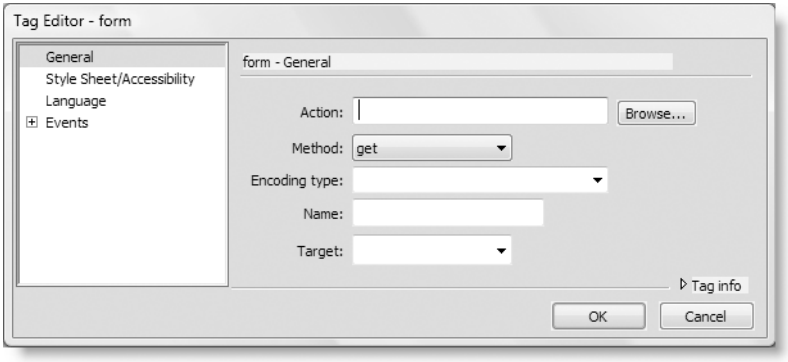


Figure 9-2. The Tag Editor is a less user-friendly way to insert a form.

The Tag Editor selects get as the default value for Method. (GET and POST are case-insensitive in the HTML `method` attribute.) If you enter a value in the Name field, Dreamweaver inserts the `name` attribute, even if you're using a strict `DOCTYPE` declaration, and doesn't assign the same value to the `id` attribute. To insert an ID, you need to select Style Sheet/Accessibility in the left column and enter the value manually.

Adding text input elements

Most online forms have fields for users to enter text, either in a single line, such as for a name, password, or telephone number, or a larger area, where the text spreads over many lines. Let's insert a couple of single-line text fields and a text area for comments.

Opinions vary on the best way to lay out a form. A simple way to get everything to line up is to use a table, but this creates problems for adding accessibility features, such as `<label>` tags. The method that I'm going to use is to put each element in a paragraph and use CSS to tidy up the layout.

Inserting text fields and a text area

Continue working with the form from the preceding exercise.

1. With your insertion point inside the red outline of the form, press Enter/Return. This inserts two empty paragraphs inside the form. Press your up arrow key once to return to the first paragraph, and click the Text Field button in the Forms tab of the Insert bar, as shown here:

2. By default, this launches the Input Tag Accessibility Attributes dialog box (see Figure 9-3).

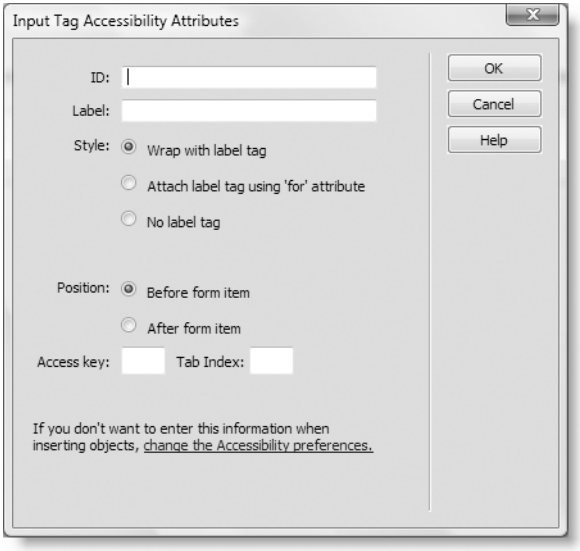


Figure 9-3. Dreamweaver makes it easy to build forms that follow accessibility guidelines.

The ID field uses the same value for the `<input>` tag's `id` and `name` attributes.

The Label field is for the label you want to appear next to the form element, including any punctuation, such as a colon, that you want to appear onscreen.

The Style option lets you choose how to associate the `<label>` tag with the form element. If you choose the first radio button, Wrap with label tag, it creates code like this:

```
<label>Name:
<input type="text" name="name" id="name" />
</label>
```

The second radio button, Attach label tag using 'for' attribute, creates code like this:

```
<label for="name">Name:</label>
<input type="text" name="name" id="name" />
```

From an accessibility point of view, either method is fine. However, using the `for` attribute often makes the page easier to style with CSS because the `<label>` and `<input>` tags are independent of each other.

The final radio button, No label tag, inserts no label at all. You normally use this with form buttons, which don't need a label because their purpose is displayed as text directly on the button.

This Style option is sticky, so Dreamweaver remembers whichever radio button you chose the last time.

The Position option, on the other hand, automatically chooses the recommended position for a form label. In the case of a text field, this is in front of the item, but with radio buttons and checkboxes, it's after the item. You can, however, override the default choice if you want to.

The final two options let you specify an access key and a tab index. Finally, if you don't want to use these accessibility features, there's a link that takes you to the relevant section of the Preferences panel to prevent this dialog box from appearing again. However, since accessibility is such an important issue in modern web design, I recommend you use these attributes as a matter of course.

Use the following settings, and click OK to insert a text field and label in the form:

- ID: name
- Label: Name:
- Style: Attach label tag using 'for' attribute
- Position: Before form item
- Access key/Tab index: Leave blank

3. Move your insertion point into the empty paragraph below, and insert another text field. Enter email in the ID field, and enter Email: in the Label field. Leave the other settings the same as in the previous step, and click OK.
4. Position your cursor after the new text field, and press Enter/Return twice to insert two more blank paragraphs in the form.
5. Put your cursor in the first blank paragraph, and click the Text Area button in the Forms tab of the Insert bar, as shown in the following screenshot:

In the Input Tag Accessibility Attributes dialog box, set ID to comments and Label to Comments:, leave the other settings as before, and click OK.

6. Move into the final blank paragraph, and select Button in the Forms tab of the Insert bar, as shown here:

In the Input Tag Accessibility Attributes dialog box, set ID to send, leave the Label field empty, select No label tag as Style, and click OK. This inserts a submit button.

7. In the Property inspector, change Value from Submit to Send comments. This changes the label on the button (press Enter/Return or move the focus out of the Value field for the change to take effect). Leave Action on the default Submit form. The form should now look like this in Design view:

If you select Reset form *in the Property inspector, this creates a reset button that clears all user input from the form. However, in Chapter 11, you'll learn how to preserve user input when a form is submitted with errors. This technique relies on setting the* value *attribute of each form element, which prevents* Reset form *from working after the form has been submitted.*

If you switch to Code view, the underlying HTML for the form should look like this:

```
<form action="" method="post" name="form1" id="form1">
  <p>
    <label for="name">Name:</label>
    <input type="text" name="name" id="name" />
  </p>
  <p>
    <label for="email">Email:</label>
    <input type="text" name="email" id="email" />
  </p>
  <p>
    <label for="comments">Comments:</label>
    <textarea name="comments" id="comments" cols="45" rows="5"> ➥
</textarea>
  </p>
  <p>
    <input type="submit" name="send" id="send" value="Send comments" />
  </p>
</form>
```

The XHTML 1.0 specification (http://www.w3.org/TR/xhtml1) lists a number of elements, including <form>, for which the name attribute has been deprecated. If you select a strict

DOCTYPE declaration, Dreamweaver omits the name attribute from the <form> tag. However, it's important to realize that this applies *only* to the opening <form> tag and not to elements within a form. The name attribute doesn't play a significant role in the <form> tag, which is why it has been deprecated, but its role on input elements in the form is crucial.

> *The* name *attribute not only remains valid for* <input>, <select>, *and* <textarea>*; PHP and other scripting languages cannot process data without it. Although the* id *attribute is optional, you must use the* name *attribute for each element you want to be processed. The* name *attribute should consist only of alphanumeric characters and the underscore and should contain no spaces.*

Setting properties for text fields and text areas

In the preceding exercise, you inserted two text fields and a text area. A text field permits user input only on a single line, whereas a text area allows multiple lines of input. The Property inspector offers almost identical options for both types of text input and even lets you convert from one to the other. Figure 9-4 shows the Property inspector for the Name text field. Notice that Type is set to Single line. This is Dreamweaver trying to be user-friendly by adopting descriptive terms, rather than using the official attribute names.

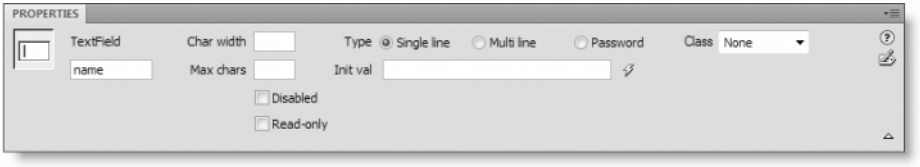


Figure 9-4. The Property inspector for a text field lets you convert it into a text area, and vice versa.

Unfortunately, if you're familiar with the correct HTML terminology, the labels in the Property inspector can be more confusing than enlightening. Let's run through the various options and their meanings:

- Type: The radio buttons determine the type of text input, as follows:
  - Single line: This creates an <input> tag and sets its type attribute to text. In other words, it creates a single-line text input field.
  - Multi line: This is what Dreamweaver uses to indicate a text area. If you select this radio button after inserting a single-line input field, Dreamweaver converts the <input> tag to a pair of <textarea> tags, as described in the next section.
  - Password: Select this option to change the type attribute of a single-line input field from text to password. This makes browsers obscure anything typed into the field by displaying a series of stars or bullets. It doesn't encrypt the input but prevents anyone from seeing it in plain text.

- Char width: This specifies the width of the input field, measured in characters. For a text field, this inserts the `size` attribute in the `<input>` tag. I normally use CSS to style the width of input fields, so you can leave this blank. This setting has no impact on the number of characters that can be typed into the field.
- Max chars: This sets the maximum number of characters that a field accepts by setting the `maxlength` attribute of a text field. If left blank, no limit is imposed.
- Init val: This lets you specify a default value for the field. It sets the `value` attribute, which is optional and normally left blank.
- Disabled: This is a new addition in Dreamweaver CS4. It adds the `disabled` attribute to the opening tag. This grays out the field when the form is displayed in a browser, preventing users from entering anything in the field.
- Read-only: This is also new to Dreamweaver CS4. Selecting this checkbox adds the `readonly` attribute to the opening tag. There's no change to the look of the field, but it prevents the user from deleting or changing the existing value.

> *The* Disabled *and* Read-only *checkboxes are visible only if you have the Property inspector expanded to its full height. Hiding the bottom half of the Property inspector saves a small amount of screen real estate but is normally a false economy.*

Figure 9-5 shows the Property inspector for the Comments text area. As you can see, it looks almost identical to Figure 9-4, although Type is set to Multi line. This time, Type has no direct equivalent in the underlying HTML. Selecting Multi line changes the tag from `<input>` to `<textarea>`.

The other important differences are that Max chars has changed to Num lines and default values have been set for Char width and Num lines. These determine the width and height of the text area by inserting the `rows` and `cols` attributes in the opening `<textarea>` tag. Both attributes are required for valid HTML and should be left in, even if you plan to use CSS to set the dimensions of the text area.

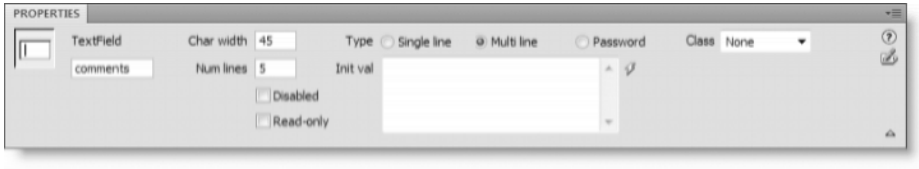


Figure 9-5. When you insert a text area, Dreamweaver gives it a default width and height.

An important change from previous versions of Dreamweaver is the removal of the Wrap option. This used to insert the invalid `wrap` attribute, which was ignored by most browsers. All modern browsers automatically wrap user input in a text area, so its removal is no loss. In its place are the Disabled and Read-only checkboxes, which work the same way as for a text input field.

Converting a text field to a text area, and vice versa

Although text fields and text areas use completely different tags, Dreamweaver lets you convert from one type to the other by changing the Type option in the Property inspector. If you change Type from Single line to Multi line, the `<input>` tag is replaced by a pair of `<textarea>` tags, and vice versa. Dreamweaver makes the process seamless by changing or removing attributes. For example, if you convert a text area to a text field, the `cols` attribute changes to `size`, and the `rows` attribute is deleted.

This is convenient if you change your mind about the design of a form, because it saves deleting one type of text input field and restarting from scratch. However, you need to remember to set both Char width and Num lines if converting a single-line field to a text area; Dreamweaver sets the defaults only when inserting a text area from the Insert bar or menu.

The Password option works only with single-line input. It cannot be used with a text area.

Styling the basic feedback form

The form looks a bit unruly, so let's give it some basic styling.

Styling the form

With the exception of a single class, all the style rules use type selectors (in other words, they redefine the style for individual HTML tags). Rather than using the New CSS Style dialog box to create them, it's quicker and easier to type them directly into a new style sheet in Code view.

1. Create a new style sheet by selecting File ➤ New. In the New Document dialog box, select Blank Page and CSS for Page Type. Insert the following rules, and save the page as `contact.css` in the `workfiles/ch09` folder. (If you don't want to type everything yourself, there's a copy in the `examples/ch09` folder. The version in the download files contains some extra rules that will be added later.)

```
body {
  background-color:#FFF;
  color:#252525;
  font-family:Arial, Helvetica, sans-serif;
  font-size:100%;
}
h1 {
  font-family:Verdana, Arial, Helvetica, sans-serif;
  font-size:150%;
}
p {
  font-size:85%;
  margin:0 0 5px 25px;
  max-width:650px;
}
form {
```

```
  width:600px;
  margin:15px auto 10px 20px;
}
label {
  display:block;
  font-weight:bold;
}
textarea {
  width:400px;
  height:150px;
}
.textInput {
  width:250px;
}
```

The style rules are very straightforward, mainly setting fonts and controlling the size and margins of elements. By setting the `display` property for `label` to `block`, each `<label>` tag is forced onto a line of its own above the element it refers to.

2. Switch to `feedback.php` in the Document window, and attach `contact.css` as its style sheet. There are several ways of doing this. One is to open the Class drop-down menu in the HTML view of the Property inspector and select Attach Style Sheet. Alternatively, you can use the menu option, Format ➤ CSS Styles ➤ Attach Style Sheet, or click the Attach Style Sheet icon at the bottom right of the CSS Styles panel.

   Browse to `contact.css`, and attach it to `feedback.php`. The form should now look a lot neater.

3. Select the Name text field, and set its class to textInput to set its width to 250 pixels. Do the same with the Email text field.

4. Save `feedback.php`, and press F12/Opt+F12 to preview it in a browser. The form should look like Figure 9-6.

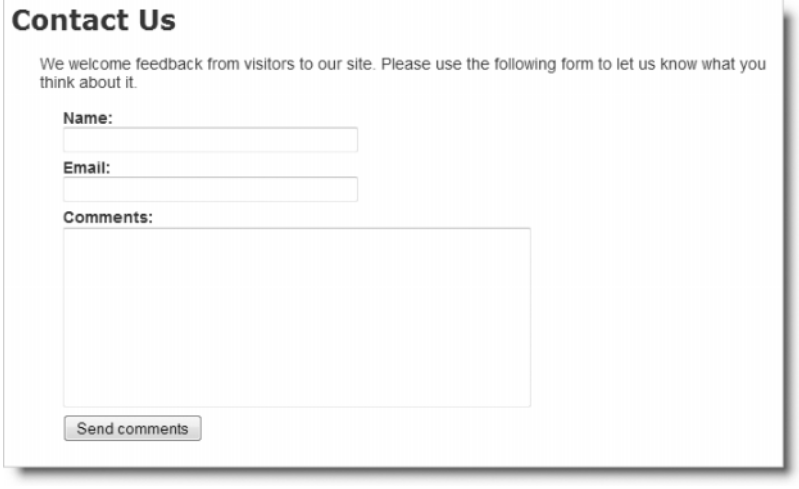


Figure 9-6.
The basic feedback form is ready for business.

Understanding the difference between GET and POST

Now that you have a form to work with, this is a good time to see how information is passed from the form and demonstrate the difference between choosing GET and POST as the method attribute. With feedback.php displayed in a browser, type anything into the form, and click the Send comments button. Whatever you typed into the text fields should disappear. It hasn't been processed because there's no script to handle it, but the content of the text fields hasn't entirely disappeared. Click the browser's reload button, and you should see a warning that the data will be resent if you reload the page.

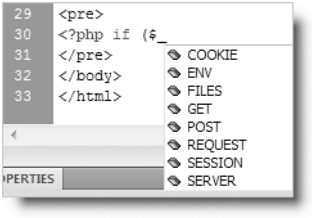
If the action attribute is empty, the default behavior is to submit the data in the form to the same page. As the warning indicates, the data has been passed to the page, but since there's no script to process it, nothing happens. Processing the data is the subject of Chapter 11, but let's take a sneak preview to see the different ways POST and GET submit the data.

Examining the data submitted by a form

In this exercise, you'll add a simple PHP conditional statement to display the data transmitted by the POST method. You'll also see what happens when the form is submitted using the GET method. Use feedback.php from the preceding exercise. If you just want to test the code, use feedback_post.php in examples/ch09.

1. Save a copy of feedback.php as feedback_post.php in workfiles/ch09. Open it in Code view, and scroll to the bottom of the page.
2. Add the following code shown in bold between the closing </form> and </body> tags:

```
</form>
<pre>
<?php if ($_POST) {print_r($_POST);} ?>
</pre>
</body>
```



As soon as you type the underscore after the dollar sign, Dreamweaver pops up a PHP code hint, as shown in the screenshot alongside. Type p (uppercase or lowercase—it doesn't matter), and press Enter/Return. Dreamweaver completes $_POST and automatically places an opening square bracket after it. Delete the square bracket. $_POST is a PHP superglobal array, which is created automatically. As the name suggests, it contains data sent by the POST method. (The role of superglobal arrays is explained in Chapter 11.)

Don't worry about the meaning of the PHP code. Just accept it for the moment, and concentrate on what it does.

3. Save the page, and load it into a browser. Enter some text in the form, and click Send comments. This time, you should see the value of each field identified by its name attribute displayed at the bottom of the page as in Figure 9-7.

Figure 9-7.
The PHP $_POST superglobal array contains the data submitted from the form.

The values gathered by the $_POST array contain not only the information entered into the text fields but also the value attribute of the submit button.

4. Change the value of method in the opening <form> tag from post to get like this:

```
<form action="" method="get" name="form1" id="form1">
```

5. Save the page, and display it again in the browser by clicking inside the address bar and pressing Enter/Return. Don't use the reload button, because you don't want to resend the POST data.

6. Type anything into the form, and click Send comments. This time, nothing will be displayed below the form, but the contents of the form fields will be appended to the URL, as shown in Figure 9-8. Again, each value is identified by its name attribute.

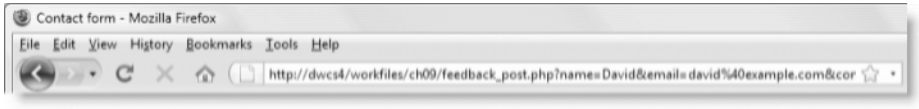


Figure 9-8. Data sent using the GET method is appended to the URL as a series of name/value pairs.

As you have just seen, the GET method sends your data in a very exposed way, making it vulnerable to alteration. Also, most browsers limit the amount of data that can be sent through a URL. The effective maximum is determined by Internet Explorer, which accepts no more than 2,083 characters, including both the URL and variables (http://support.microsoft.com/kb/208427). The POST method is more secure and can be used for much larger amounts of data. By default, PHP permits up to 8MB of POST data, although hosting companies may set a smaller limit.

Because of these advantages, you should normally use the POST method with forms. The GET method is used mainly in conjunction with database searches and has the advantage that you can bookmark a search result because all the data is in the URL.

Although the POST *method is more secure than* GET*, you shouldn't assume that it's 100-percent safe. For secure transmission, you need to use encryption or the Secure Sockets Layer (SSL).*

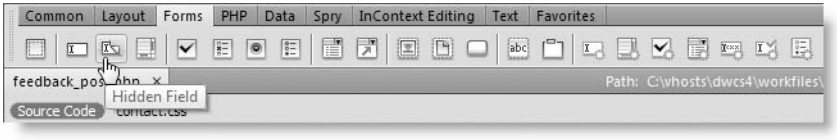
Passing information through a hidden field

Frequently, you need to pass information to a script without displaying it in the browser. For example, a form used to update a database record needs to pass the record's ID to the update script. You can store the information in what's called a hidden field.

Adding a hidden field

Although you don't need a hidden field in this feedback form, let's put one in to see how it works. Hidden fields play an important role in later chapters. Continue working with `feedback_post.php` from the preceding exercise. The finished code is in `feedback_hidden.php`.

1. Set the value of `method` back to `post`. Do this in Code view or by selecting the form in Design view and setting Method to POST in the Property inspector.
2. A hidden field isn't displayed, so it doesn't matter where you locate it, as long as it's inside the form. However, it's normal practice to put hidden fields at the bottom of a form. Switch back to Design view, and click to the right of the Send comments button.
3. Click the Hidden Field button in the Forms tab of the Insert bar, as shown here:



4. Dreamweaver inserts a hidden field icon alongside the Send comments button. Type a name for the hidden field in the left text field in the Property inspector and the value you want it to contain in the Value field, as shown in Figure 9-9.

Figure 9-9. Select a hidden field's icon in Design view to edit its name and value in the Property inspector.

Note that the PHP script at the bottom of the page is indicated by a gold PHP icon. If you can't see the hidden field or PHP icons in Design view, select View ➤ Visual Aids ➤ Invisible Elements.

The option on the View *menu controls the display of invisible elements only on the current page. To change the default, open the* Preferences *panel from the* Edit *menu (*Dreamweaver *menu in a Mac), and select the* Invisible Elements *category. Make sure there's a check mark alongside* Hidden form fields *and* Visual Server Markup Tags*, and then click* OK*. The* Visual Aids *submenu is useful for turning off the display of various tools when they get in the way of the design of a page. You can toggle currently selected visual aids on and off with the keyboard shortcut Ctrl+Shift+I/Shift+Cmd+I.*

5. Switch to Code view. You'll see that Dreamweaver has inserted the following code at the end of the form:

```
<input name="secret" type="hidden" id="secret" value="Guess what?" />
```

6. Save `feedback_post.php`, and press F12/Opt+F12 to load the page in a browser (or use `feedback_hidden.php` in `examples/ch09`). The hidden field should be, well . . . hidden. Right-click to view the page's source code. The hidden field and its value are clearly visible. Test the form by entering some text and clicking Send comments. The value of `secret` should be displayed with the rest of the form input.

Just because a hidden field isn't displayed in a form doesn't mean that it really is hidden. Frequently, the value of a hidden field is set dynamically, and the field is simply a device for passing information from one page to another. Never use a hidden field for information that you genuinely want to keep secret.

Using multiple-choice form elements

Useful though text input is, you have no control over what's entered in the form. People spell things wrong or enter inappropriate answers. There's no point in a customer ordering a yellow T-shirt when the only colors available are white and black. Multiple-choice form elements leave the user in no doubt what the options are, and you get answers in the format you want.

Web forms have four multiple-choice elements, as follows:

- **Checkboxes**: These let the user select several options or none at all. They're useful for indicating the user's interests, ordering optional accessories, and so on.
- **Radio buttons**: These are often used in an either/or situation, such as male or female and yes or no, but there's no limit to the number of radio buttons that can be used in a group. However, only one option can be chosen.
- **Drop-down menus**: Like radio buttons, these allow only one choice but are more compact and user-friendly when more than three or four options are available.
- **Multiple-choice lists**: Like checkboxes, these permit several options to be chosen, but present them as a scrolling list. Often, the need to scroll back and forth to see all the options makes this the least user-friendly way of presenting a multiple choice.

Let's add them to the basic feedback form to see how they work.

Offering a range of choices with checkboxes

There are two ways to use checkboxes. One is to give each checkbox a different name; the other is to give the same name to all checkboxes in the same group. Which you choose depends on the circumstances. Use the first when the options represented by checkboxes aren't related to each other; create a checkbox group when there's a logical relationship between them. Normally, Dreamweaver uses the same values for the `id` and `name` attributes of form elements. Since an ID must always be unique, treating checkboxes as a group in previous versions of Dreamweaver involved adjusting the `name` attribute of each checkbox in Code view.

That's no longer a problem in Dreamweaver CS4, because you have the choice of creating individual checkboxes or a checkbox group. Individual checkboxes have the same value for both `name` and `id` attributes. Members of a checkbox group have separate `id` attributes but share a common `name`. You create a checkbox group through the simple dialog box shown in Figure 9-10. Access the dialog box through the Checkbox Group button in the Forms tab of the Insert bar or by selecting Insert ➤ Form ➤ Checkbox Group.

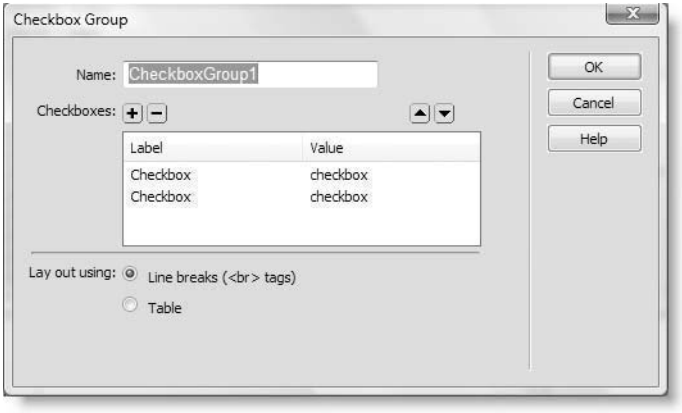


Figure 9-10. The new Checkbox Group dialog box speeds up the creation of related checkboxes.

The Checkbox Group dialog box has the following options:

- Name: This field is where you enter the `name` attribute that you want to assign to all checkboxes within the group. If you're feeling lazy, you can just accept the default. Dreamweaver automatically increments the number at the end of the default name if there is more than one checkbox group on a page.
- Checkboxes: This field is prefilled with two dummy checkboxes. To change the placeholder text, click inside the first Label field to open it for editing. You can move to the other prefilled fields with the Tab key or by clicking directly inside them. Add or remove checkboxes with the plus and minus buttons at the top left of the Checkboxes field. Change their order with the up and down arrows at the top right.

- Label: This is for the text you want to appear alongside the checkbox.
- Value: This is the value you want the checkbox to represent, if selected, when the form is submitted for processing.
- Lay out using: Choose whether to lay out the checkbox group using line breaks (`<br />` tags are used with an XHTML `DOCTYPE` declaration) or a single-column table.

However, you can't reopen the dialog box to add new checkboxes after the group has been created. Any extra checkboxes need to be added individually. The next exercise shows you how to do both.

Inserting a group of checkboxes

Continue working with `feedback_post.php` from the preceding exercise. Alternatively, copy `feedback_multi_start.php` from `examples/ch09` to `workfiles/ch09`. The finished code for this exercise is in `feedback_checkbox.php`.

1. Save the page as `feedback_checkbox.php` in `workfiles/ch09`.
2. With the page open in Design view, click immediately to the right of the Comments text area. Press Enter/Return to insert a new paragraph.
3. Each checkbox has its own label, so you need a heading for the checkbox group that uses the same font size and weight as the `<label>` tags.

   Make sure that the HTML view of the Property inspector is selected, and click the Bold button (the large B just to the right of the CSS button). Although the tooltip says Bold, this inserts the `<strong>` tag in accordance with current standards, rather than the presentational `<b>` tag.

   > *When clicking the Bold button, it's vital that you're in the* HTML *view of the Property inspector. If you're in the* CSS *view, clicking the* Bold *button adds* `font-weight: bold;` *to the current selection in* Targeted Rule. *Since you're inside a paragraph, this makes the text in* all *paragraphs bold, not just the current one.*

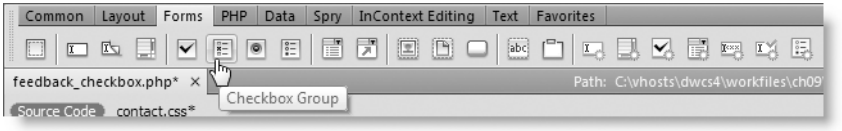
4. Type a heading for the checkbox group. I used What aspects of London most interest you? Click the Bold tag again to move the cursor outside the closing `</strong>` tag in the underlying code.
5. Checkboxes usually have short labels, so it's often a good idea to display them in columns. The Checkbox Group dialog box has two layout options. You can display the checkboxes in a single-column table or use line breaks (`<br />` tags).

   If you choose line breaks, Dreamweaver automatically wraps the checkbox group in a pair of `<p>` tags unless the insertion point is already in a paragraph, in which case it uses the existing tags.

   Rather than use a table, I'm going to use a couple of floated paragraphs. You need to create a style rule for them later, but let's start by creating the checkbox group.

   With your insertion point at the end of the paragraph you entered in step 4, press Enter/Return to create a new paragraph.

6. Click the Checkbox Group button in the Forms tab of the Insert bar, as shown here:



> *Make sure you select the correct button. The* Checkbox Group *button uses the same icon as the* Radio Group *button (two buttons farther right). I find them easy to tell apart because each group button is immediately to the right of the button that inserts a single checkbox or radio button. However, if you find the plethora of icons in Dreamweaver confusing, either use the menu alternatives or display the* Insert *bar as a panel with labels (see Chapter 1).*

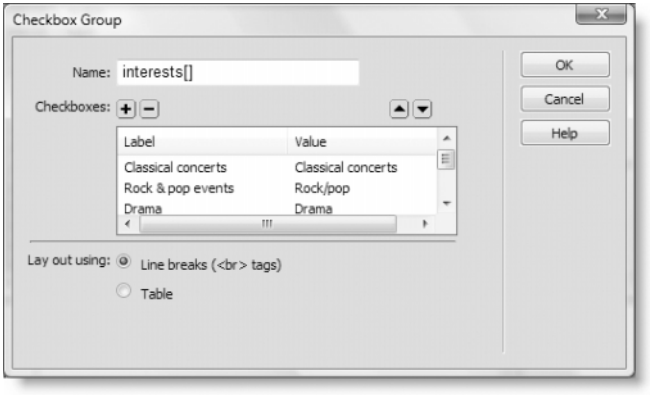
This opens the Checkbox Group dialog box shown in Figure 9-10.

7. In Chapter 11, you will build a PHP script to process this form. If more than one form item has the same `name` attribute, PHP expects the values to be submitted as an array (PHP arrays are described in the next chapter). To get PHP to treat all values in the checkbox group as an array, you need to add an empty pair of square brackets after the `name` attribute. I want to use `interests` as the `name` for this checkbox group, so enter interests[] in the Name field (there should be no space between interests and the square brackets).

8. Click the plus button alongside Checkboxes twice to add two checkboxes, and edit the Label and Value fields using the following values:

■ Classical concerts	Classical concerts
■ Rock & pop events	Rock/pop
■ Drama	Drama
■ Guided walks	Walks

9. Select Line breaks (
 tags) for the layout. The Checkbox Group dialog box should now look like this:



10. Click OK to insert the checkbox group. The new code should look like this in Split view:

As the preceding screenshot shows, Dreamweaver wraps the `<label>` tags around the checkbox `<input>` tags, rather than using the `for` attribute. This doesn't affect the way the checkboxes are displayed in this form, so I'm going to leave them as they are. You can also see that each checkbox has the same name attribute, but the id attributes are all unique. Dreamweaver has numbered them incrementally as `interests_0`, `interests_1`, and so on.

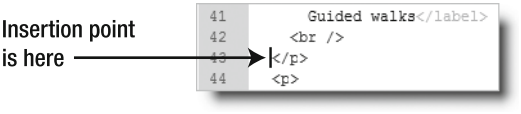
11. As explained earlier, you cannot reopen the Checkbox Group dialog box to add more checkboxes. The simple way to add another checkbox is to open Code view and copy and paste an existing checkbox. For example, to add a checkbox for Art, you could copy and paste the code shown on lines 39–41 in the preceding screenshot and edit them like this (new code is shown in bold):

```
<label>
  <input type="checkbox" name="interests[]" value="Art" ➥
id="interests_4" />
Art</label>
```

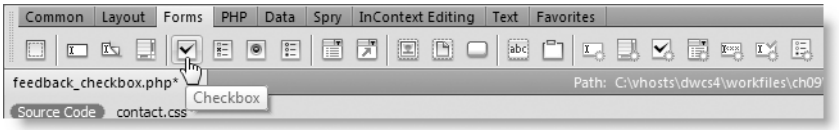
Alternatively, you need to add a single checkbox using the Checkbox button immediately to the left of the Checkbox Group button in the Forms tab of the Insert bar (or Insert ➤ Form ➤ Checkbox). The next few steps show you how to do that.

12. One of the trickiest aspects of adding a checkbox to an existing group is getting the insertion point in the right place. As with the `<form>` tag, you get a far less user-friendly dialog box if you position your cursor in Code view. To get to the right position in Design view, open Split view, and keep an eye on the position of the insertion point in Code view.

Click to the right of the label of the last checkbox (Guided walks) in Design view, and press the down arrow key twice. This should move the insertion point to just inside the closing `</p>` tag of the checkbox group, as shown.



13. Click the Checkbox button in Forms tab of the Insert bar, as shown in the following screenshot:



14. In the Input Tag Accessibility Attributes dialog box, enter the following values, and click OK.
    - ID: interests_4
    - Label: Art
    - Style: Wrap with label tag
    - Position: After form item (Dreamweaver selects this automatically)

15. Dreamweaver inserts the following code in Code view:

```
<label>
  <input type="checkbox" name="interests_4" id="interests_4" />
Art</label>
```

    You need to change the value of the `name` attribute to match the other checkboxes like this:

```
<label>
  <input type="checkbox" name="interests[]" id="interests_4" />
Art</label>
```

    You must do this in Code view. If you use the Property inspector, Dreamweaver uses the same value for both `name` and `id` attributes. Square brackets are permitted in the `name` attribute, but not in an ID.

16. Save the page, and load it into a browser. Select some of the checkboxes, and click the Send comments button. The checked values should appear at the bottom of the page. Try it with no boxes checked. This time, `interests` isn't listed.

    Check your code, if necessary, with `feedback_checkbox.php` in `examples/ch09`. Keep the file open, because you'll continue working with it in the next exercise.

As you can see, adding an extra checkbox to an existing checkbox group is rather fiddly, because you can't set the `name` and `id` attributes separately at the time of creation. However, if you are creating a stand-alone checkbox, for example one that asks users to confirm they agree to the terms and conditions of the site, it doesn't matter if the `name` and `id` are the same.

Displaying the checkboxes in columns

Currently, the checkboxes are stacked one on top of the other. Moving them into two columns is simply a matter of splitting them into two paragraphs and floating them left. Continue working with the same file as in the previous exercise.

1. To split the checkboxes into separate paragraphs, you need to go into Code view and replace the `<br />` tag between the third and fourth checkboxes with a closing `</p>` tag and an opening `<p>` one like this:

```
  Drama</label>
</p>
<p>
  <label>
    <input type="checkbox" name="interests[]" value="Walks" ➥
id="interests_3" />
```

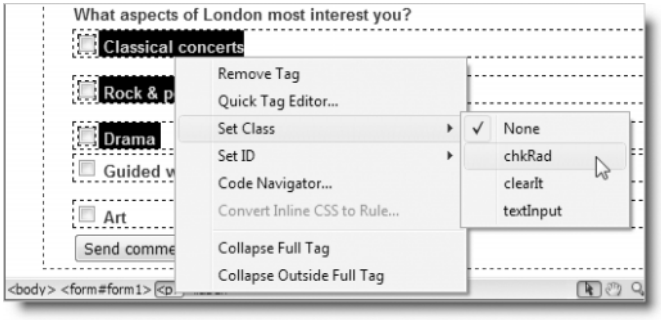
2. You now need to create a couple of style rules to float the paragraphs. Select `contact.css` in the Related Files toolbar, and add the following rules at the bottom of the page:

```
.chkRad {
  float: left;
  margin-bottom: 15px;
  margin-left: 50px;
}
.clearIt {
  clear: both;
}
```

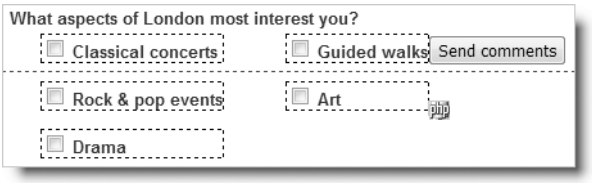
   The first rule creates the chkRad class, which will be applied to both checkboxes and radio buttons, floating them left and adding margins on the bottom and left.

   The `.clearIt` selector uses the `clear` property, which prevents other elements from moving up into empty space alongside a floated element. This will be applied to the paragraph containing the submit button.

3. Click inside any of the first three checkboxes, and right-click the `<p>` tag in the Tag selector at the bottom of the Document window. Select Set Class ➤ chkRad from the context menu, as shown here:

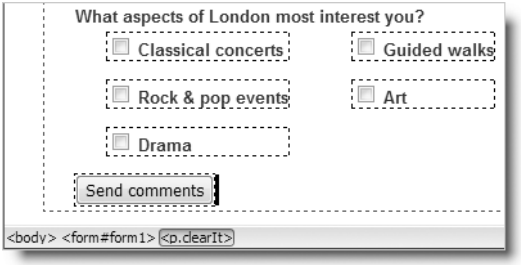


4. Do the same for the second paragraph containing checkboxes. This results in the Send comments button floating up alongside the second column of checkboxes, as shown here:



5. Fix this by selecting the Send comments button in Design view and then right-clicking the <p> tag in the Tag selector at the bottom of the Document window. Select Set Class ➤ clearIt from the context menu.

   The Send comments button should move down to its original position, as shown in the following screenshot:



   If the button remains floating, make sure you applied the class to the surrounding paragraph, not to the button itself.

   Check your code, if necessary, with `feedback_checkbox_cols.php` in `examples/ch09`.

Using radio buttons to offer a single choice

The term **radio buttons** is borrowed from the preset buttons common on radios: you push a button to select a station and the currently selected one pops out; only one can be selected at any given time. Like a radio, there shouldn't be too many buttons to choose from. Otherwise, the user gets confused.

As with checkboxes, Dreamweaver offers you the choice of inserting radio buttons one at a time or as a group in a single operation. Similarly, there's no way of relaunching the Radio Group dialog box to edit the radio buttons or add a new one to the group. The options in the Radio Group dialog box are identical to the Checkbox Group dialog box (see Figure 9-10), so refer to the previous section for details.

The following exercise shows how to insert individual radio buttons.

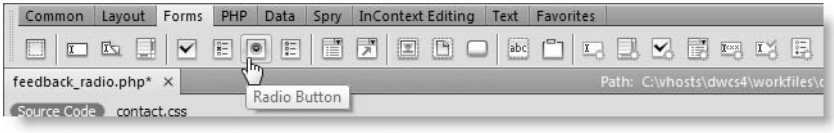
Creating a radio button group with individual buttons

Continue working with the form from the preceding exercise, or copy `feedback_checkbox_cols.php` from `examples/ch09` to `workfiles/ch09`. The finished code is in `feedback_radio.php`.

1. Save the file as `feedback_radio.php`.
2. Like checkboxes, each radio button has its own label, so you need to create a heading to indicate the question being asked. To add a new paragraph below the checkbox group, click in the last checkbox label in the right column (Art) in Design view, and press Enter/Return. Instead of the cursor moving below the checkbox group, it lines up alongside the Guided walks label. This is because Dreamweaver automatically applies the same style as the preceding paragraph when you press Enter/Return.
3. You need to remove the chkRad class from the new paragraph and replace it with the `clearIt` class. Right-click <p.chkRad> in the Tag selector at the bottom of the Document window, and select Set Class ➤ clearIt from the context menu. This is the same technique as in step 3 of the previous exercise, only this time you are changing the class rather than applying a new one.

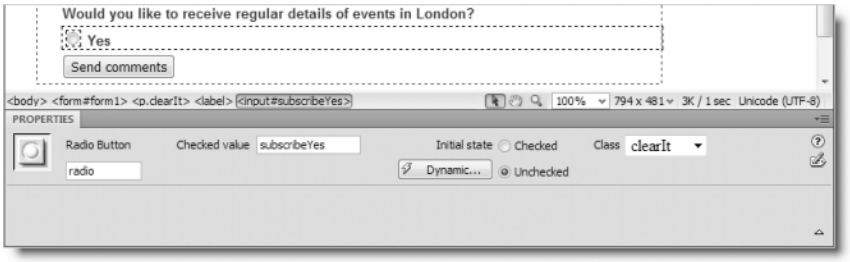
> *You cannot use the Tag selector or Property inspector to apply multiple classes to the same element. The only ways to do so in Dreamweaver are through the* Tag Inspector *panel or in Code view.*

4. Click the Bold button in the HTML view of the Property inspector, and type a question. I used Would you like to receive regular details of events in London?
5. At the end of the line, click the Bold button again to move the insertion point outside the closing `</strong>` tag, and press Enter/Return to create a new paragraph.
6. Click Radio Button in the Forms tab of the Insert bar, as shown here:



7. Enter the following settings in the Input Tag Accessibility Attributes dialog box:
   - ID: subscribeYes
   - Label: Yes
   - Style: Wrap with label tag
   - Position: After form item (Dreamweaver selects this automatically)

8. When you click OK, Dreamweaver inserts the radio button and its associated label. Select the radio button element in Design view to display its details in the Property inspector, which should look like this:



   The field on the left immediately below Radio Button sets the `name` attribute for the radio button. Change it to subscribe. Unlike other form elements, the `name` and `id` attributes of radio buttons aren't automatically linked in the Property inspector because Dreamweaver is smart enough to know that all buttons in a radio group share the same name, but they must have unique IDs. However, since only one value is submitted from a radio group, unlike a checkbox group, you don't need to add square brackets after the name.

   Dreamweaver automatically enters the same value as the ID in Checked value. While this is OK, you can change the value here without affecting the ID. Just type the letter y in the Checked value field.

   Leave the other values unchanged. Although the Class field displays clearIt, this is inherited from the surrounding paragraph. You need to change the paragraph's class, but it's better to do it after you have finished inserting the other radio button because it's easier to position the insertion point in Design view in nonfloated elements.

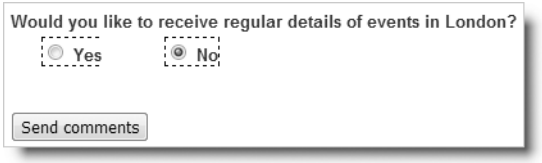
9. Click to the right of the Yes label in Design view, and press Enter/Return to insert a new paragraph. Repeat steps 6 and 7 to insert a second radio button, setting ID to subscribeNo and Label to No.

10. Select the second radio button element in Design view to display its details in the Property inspector. Change its name from radio to subscribe, and shorten Checked value to the letter n. It's always a good idea to set a default value for a radio button group, so set Initial state to Checked.

11. All that remains to do is change the class of the paragraphs surrounding the two radio buttons and float them alongside each other. However, to make it easy to insert the next form element in the following exercise, click to the right of the No label, and press Enter/Return to insert a new paragraph. This inherits the `clearIt` class, so it won't float alongside the radio buttons.

12. Click to the right of the Yes label in Design view, right-click <p.clearIt> in the Tag selector at the bottom of the Document window, and select Set Class ➤ chkRad

from the context menu. Do the same with the paragraph surrounding the No radio button. The radio buttons should float alongside each other like this:



13. Save the page, and load it in a browser. Test it to make sure that the value of `subscribe` is `y` or `n` depending on the radio button selected.

    Check your code, if necessary, against `feedback_radio.php` in `examples/ch09`.

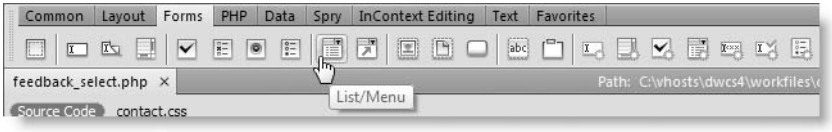
Offering a single choice from a drop-down menu

Drop-down menus and multiple-choice lists both use the HMTL `<select>` tag, with each individual item in an `<option>` tag. Apart from two attributes in the opening `<select>` tag, their underlying structure is identical, so Dreamweaver uses the same tools to insert and configure them. First, let's take a look at a single-choice menu. The following instructions show you how to add one to the feedback form.

Inserting and configuring a drop-down menu

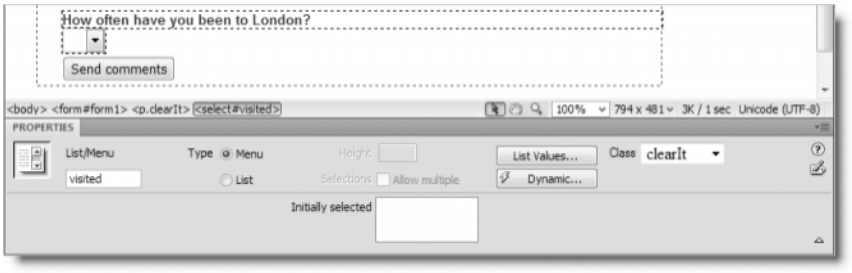
Continue working with the form from the preceding exercise, or copy `feedback_radio.php` from `examples/ch09` to `workfiles/ch09`. The finished code is in `feedback_select.php`.

1. Save the file as `feedback_select.php`.
2. Insert your cursor in the empty paragraph between the radio buttons and the Send comments button, and click the List/Menu button on the Forms tab of the Insert bar, as shown here:



3. Enter the following settings in the Input Tag Accessibility Attributes dialog box:
   - ID: visited
   - Label: How often have you been to London?
   - Style: Attach label tag using 'for' attribute
   - Position: Before form item (Dreamweaver selects this automatically)

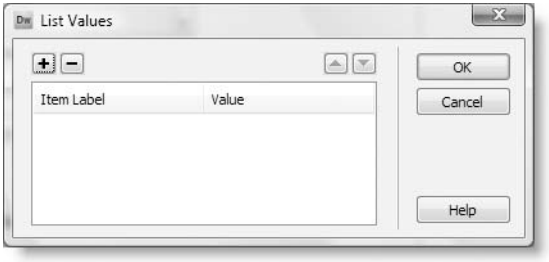
4. When you click OK, Dreamweaver inserts the label and a blank menu element in Design view. Click the menu element to select it and display its details in the Property inspector, as shown in the following screenshot:



> *If you have difficulty selecting the menu element in Design view, open Split view, and click anywhere inside the* `<select>` *tag.*

5. Type is set by default to Menu, which builds a single-choice drop-down menu. The List option creates a scrolling list. You'll see how that works in the next section.

To populate the menu, click the List Values button in the Property inspector. This opens the List Values dialog box, as shown in the following screenshot:



Item Label is what you want to be shown in the menu, and Value is the data you want to be sent if the item is selected when the form is submitted. The `value` attribute of the `<option>` tag is optional, so the Value field needs to be set only if you want the label and the data to be different.

> *This is another example of Dreamweaver using what it regards as user-friendly expressions.* Item Label *is the text element that goes between the* `<option>` *tags of a* `<select>` *menu. While this is, no doubt, helpful to some users, it can also be confusing because it bears no relation to the* `<label>` *tags that are used to improve the accessibility of online forms.*

The easiest way to fill in the dialog box is to tab between the fields. Tabbing from the Value field creates the next item. You can also click inside an existing field to edit it. Use the minus (–) button to delete a selected item and the up and down arrows to reorder the list.

I used the following values:

- -- Select one -- 0
- Never been Never
- Once or twice 1-2 times
- Less than once a year Not yearly
- I go most years Yearly
- I live there Resident

The first item simply asks users to select one of the options. I have set the Value field to 0 to indicate that nothing has been selected. Without an explicit value, the text contents of the `<option>` tag is submitted by the form.

Click OK when you are finished.

6. Dreamweaver normally displays the longest option in Design view. To specify the one you want to be displayed when the form first loads, select it in the Initially selected field in the Property inspector. This adds `selected="selected"` to the `<option>` tag.

   By default, browsers show the first item in the menu if you don't set the Initially selected field. However, it's often useful to select an item that's lower down the list. For example, you may want to display a list of countries in alphabetical order, but if most of your visitors are from the United States, it's a courtesy to display that by default rather than forcing them to scroll all the way down the list to select it.

7. Save `feedback_select.php`, and load it in a browser. Select a menu item, and click Send comments. The value should be displayed as `visited` at the bottom of the page.

   Check your code, if necessary, against `feedback_select.php` in `examples/ch09`.

Creating a multiple-choice scrollable list

The way you build a multiple-choice list is almost identical to a drop-down menu. It involves only a couple more steps to set the `size` and `multiple` attributes in the opening `<select>` tag. Strictly speaking, the `multiple` attribute is optional. If it's omitted, the user can select only a single item.

You could convert the menu from the preceding section by changing Type from Menu to List in the Property inspector. However, the way you process data from a multiple-choice list is different, so let's add a separate list to the same form.

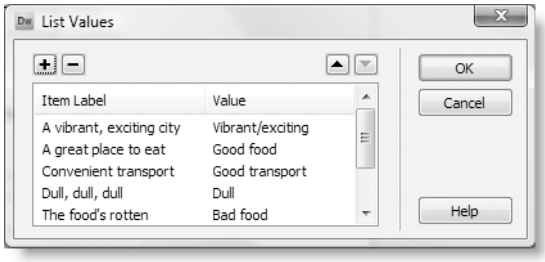
Inserting and configuring a scrollable list

Continue working with the form from the preceding exercise, or copy `feedback_select.php` from `examples/ch09` to `workfiles/ch09`. The finished code is in `feedback_multiselect.php`.

1. Save the file as `feedback_multiselect.php`.
2. In Design view, click immediately to the right of the drop-down menu you inserted in the previous exercise, and press Enter/Return to insert a new paragraph. Because the `clearIt` class was applied to the preceding paragraph, Dreamweaver applies the same class to the new paragraph. Leaving it does no harm, but you don't really need it either, so reset Class to None in the HTML view of the Property inspector.
3. Click the List/Menu button on the Forms tab of the Insert bar.
4. Enter the following settings in the Input Tag Accessibility Attributes dialog box:
   - ID: views
   - Label: What image do you have of London?
   - Style: Attach label tag using 'for' attribute
   - Position: Before form item (Dreamweaver selects this automatically)
5. When you click OK, Dreamweaver inserts a blank drop-down menu into the page in the same way as in step 4 of the preceding exercise. Select the menu element in Design view to display its details in the Property inspector.

   Change Type to List. This activates the Height and Selections options. These are more examples of Dreamweaver's attempt at user-friendly names instead of using the HTML attributes. Height sets the `size` attribute, which determines the number of items visible in the list; the browser automatically adds a vertical scrollbar. Change the value to 6, and put a check mark in the Selections checkbox to permit multiple choices. This adds `multiple="multiple"` in the `<select>` tag. The menu is converted into a tall, narrow rectangle, as shown here:

6. Click the List Values button to enter the labels and data values the same as for a drop-down menu. Leave Value blank if you want the data sent by the form to be the same as the label. The following screenshot shows the first five values I used:



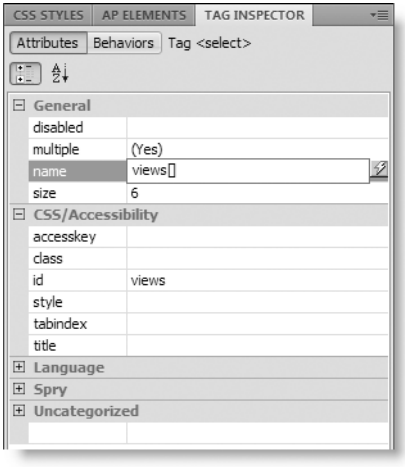
I set the sixth Item Label to A transport nightmare, and its Value to Transport nightmare.

7. Save the page, and load it into a browser. Select several items in the list (holding down the Shift or Ctrl/Cmd key while clicking), and click the Send comments button.

Uh, oh . . . something is wrong. Only the last selected item appears at the bottom of the page. To get all items, you need to use an array in the same way as with the checkbox group by appending a pair of square brackets to the end of the `name` attribute. Fortunately, there's only one `name` attribute to change.

The problem with the Property inspector is that it uses the same field for the `name` and `id` attributes. If you add the square brackets to views in the Property inspector, it affects both `name` and `id`. You could dive into Code view to fix the problem, but let me show you another way—using the Tag Inspector panel.

8. Make sure the list is selected in Design view, and open the Tag Inspector panel (F9/Shift+Opt+F9 or Window ➤ Tag Inspector). If the Behaviors button is selected, click the Attributes button at the top left of the Tag Inspector panel. This gives you direct access to the attributes of the element currently selected in the Document window. It has two views: listing attributes by category or in alphabetical order.



Expand the General and CSS/Accessibility categories in category view to reveal the `name` and `id` attributes. Click inside the name field to add a pair of square brackets after views, as shown in the screenshot alongside. (Depending on your monitor's resolution, they might appear to merge into an upright rectangle. This doesn't matter.)

9. Press Enter/Return to save the change. Save the page, and test it again in a browser. This time, all selected items from the multiple-choice list should be displayed as an array at the bottom of the page.
10. Click Send comments without selecting anything in the list. This time, `views` won't be among the items displayed at the bottom of the page. This is the same as with a checkbox group, and it has important implications for how you process the output of a form, as you'll see in Chapter 11.

    Compare your code, if necessary, with `feedback_multiselect.php` in `examples/ch09`.

Organizing form elements in logical groups

An important element in designing a usable form is making sure that everything is laid out logically so that users can see at a glance what sort of information is required. It can also help to divide the form into a number of clearly labeled sections. HTML provides two tags for this purpose: `<fieldset>` and `<legend>`, which most browsers automatically style with a border (see Figure 9-11).

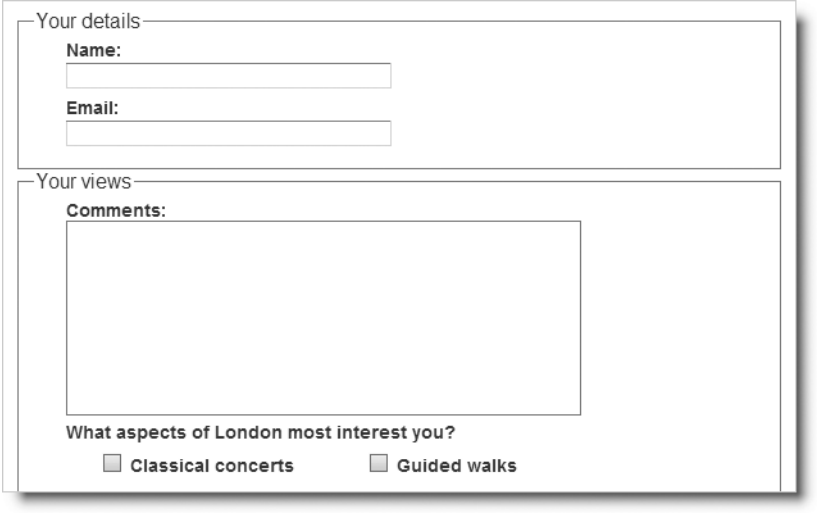
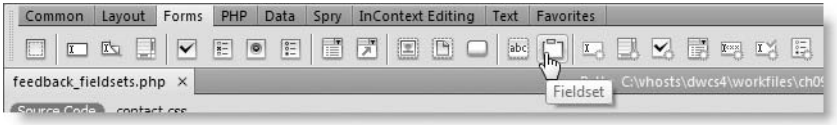


Figure 9-11. Fieldsets give forms a visual and logical structure that help make them more accessible to all users.

Inserting a fieldset

You can add fieldsets to your form before inserting the individual form elements or after you have finished. To insert a fieldset, click the Fieldset button on the Forms tab of the Insert bar, as shown here:



This opens the Fieldset dialog box. It has just one field: Legend, which is the title you want to give to the group of form elements within the fieldset.

When you click OK, Dreamweaver inserts the following code in your form:

```
<fieldset>
<legend>Your details</legend>
</fieldset>
```

If you create the fieldset before inserting the individual form elements, press your right keyboard arrow after clicking OK in the Fieldset dialog box. This positions the insertion point between the closing `</legend>` and `</fieldset>` tags ready for adding the form elements that belong to the fieldset.

To add a fieldset to existing form elements, select the elements you want to include by dragging your mouse across them in Design view. If you have Split view open, you will see that Dreamweaver doesn't select the opening and closing tags of your selection. However, when you insert the fieldset, it's smart enough to put the `<fieldset>` and `<legend>` tags in the correct place. If the fieldset border and legend appear in the wrong place, it probably means that you failed to select the form elements correctly. Press Ctrl+Z/Cmd+Z or Edit ➤ Undo, and try again. Alternatively, go into Code view, and make sure the target form elements are between the closing `</legend>` and `</fieldset>` tags.

To see the effect of adding fieldsets to the form you have been using throughout this chapter and to study the code, take a look at `feedback_fieldsets.php` in `examples/ch09`. You can alter the look of fieldsets with CSS by adding `fieldset` and `legend` type selectors to your style sheet.

Now that I've covered all the main form input and layout elements, let's turn our attention to checking user input before submitting the form to the server for processing.

Validating user input before submission

Validation of user input plays a very important role in the design and processing of online forms. Let's say you're building a form that offers to send customers more information. There's no point processing the form if it doesn't contain certain details, such as email or postal address. Similarly, a form that asks for the user's age needs to make sure the information supplied falls within an acceptable range. For example, it must be a number. Is there a minimum age, such as 16? Setting a maximum age is more difficult, but obviously a figure such as 402 should be rejected. Validation can't stop people from entering false

information; its role is to ensure that you get the type of information you expect—an email address in an email field and something that looks like a phone number in a phone number field. Well-designed sites usually perform validation twice—on the client side before the data is submitted to the server and once again on the server side.

The problem with client-side validation is that it relies on JavaScript. A visitor simply needs to turn off JavaScript in the browser and press the submit button; all your client-side filters are rendered useless. Consequently, some developers argue that client-side validation is a waste of time. Nevertheless, most visitors to your sites aren't deliberately trying to abuse your forms and are likely to have JavaScript enabled. So, it's generally a good idea to detect errors before a form is submitted. JavaScript validation is conducted locally and is usually instantaneous. It's done as a courtesy to the user, who doesn't need to wait for a response from the server if there's a mistake in the information submitted. It also helps reduce the burden on the server, because forms aren't submitted with incomplete information.

Nevertheless, the fact that client-side validation can be so easily evaded raises the question of how thorough it should be. Since the real checks need to be done on the server, there's a strong argument for keeping client-side checks to the absolute minimum or eliminating them altogether. Client-side validation is optional; server-side validation should never be omitted. We'll look at server-side validation in Chapter 11. The rest of this chapter is devoted to client-side validation with Spry validation widgets.

In addition to Spry validation widgets, the Validate Form *behavior can be accessed through the* Behaviors *panel. However, the checks it performs are so rudimentary as to be virtually worthless.*

Using Spry validation widgets

The Spry validation widgets, which were first introduced in Dreamweaver CS3, are anything but rudimentary. They're capable of performing a wide range of checks and use a combination of JavaScript and CSS to display customized alerts alongside the affected field. Three new validation widgets were added in Dreamweaver CS4, bringing the types of form input they can handle to seven, as follows:

- Text fields
- Text areas
- Checkboxes
- Radio button groups
- Menus and lists
- Password fields
- Password confirmation fields

The text field widget is particularly impressive, because it lets you test for a wide range of formats, including numbers, currency, IP addresses, Social Security numbers, and credit card numbers. You can even set up your own custom patterns without the need to master the complex subject of regular expressions. The text area validation widget also provides one of the most frequently requested features—the ability to display how many characters the user has entered or still has left before reaching a predetermined limit. The validation widgets also display warning messages that you can easily edit and style with CSS.

You can access the validation widgets on both the Forms and Spry tabs of the Insert bar. As Figure 9-12 shows, the icons are very similar to those of their related form elements. However, if you prefer menus to icons, the same options are also available on the Spry submenu of the Insert menu.

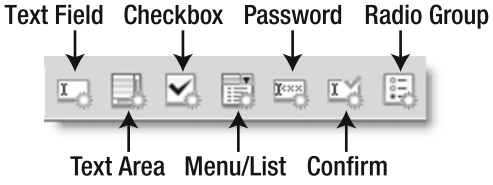


Figure 9-12.
Spry validation widgets have an orange sunburst on the same icons as their related form elements.

Spry validation widgets are certainly powerful, but they greatly increase page size. If you add all seven widgets to a form, the external JavaScript files and style sheets weigh in at more than 200KB. The text field widget is responsible for roughly one third that amount because of its extensive pattern-matching features. It's overkill for a very basic form, but could be extremely useful in validating user input on a form for a job application or an insurance policy quote.

The external files are cached by the user's browser, so are downloaded only once. However, if you're concerned about file size, you can use optimized versions of the Spry JavaScript files by downloading the Spry framework from `http://labs.adobe.com/technologies/spry/`. *The versions of the files in the* `includes_packed` *folder weigh in at just 85KB. The files have the same names as those inserted by Dreamweaver, so just swap them over.*

If you insert a widget into a blank part of a form, Dreamweaver inserts both the validation code and the form element. Alternatively, you can apply a widget to an existing form element. Whichever approach you use, the method of configuration is exactly the same. In the remaining pages of this chapter, I'm going to show you how to apply validation widgets to an existing form.

I suggest you study carefully the first section of "Validating a text field with Spry," because it contains most of the knowledge you need to work with all validation widgets, particularly with regard to editing and controlling the display of alert messages.

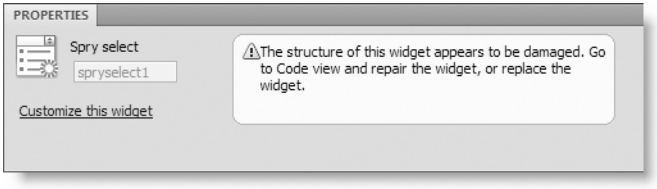
Inserting a Spry validation widget

As with all Spry widgets, the page must have been saved at least once before you can apply a validation widget. Save the page again immediately afterward to attach the external JavaScript code and style sheet, and copy them to the Spry assets folder if necessary.

> *Unless you're a JavaScript expert, don't try to use other JavaScript validation functions, such as the* Validate Form *behavior, on the same form with Spry validation widgets, because they're likely to conflict and cease functioning.*

Removing a validation widget

Removing a widget immediately after you have applied it is easy. Unfortunately, the standard method of removing a Spry widget (selecting its turquoise tab and pressing Delete) removes the form element with it. The simple way to get around this problem is to select the form element (and label, if necessary) in Design view and cut it to your clipboard (Ctrl+X/Cmd+X). Then select the turquoise tab to delete the widget. If you see the following warning that the widget has been damaged, you can safely ignore it:



Once you have removed the widget, paste (Ctrl+V/Cmd+V) the form element back into the page.

> *Dreamweaver is context-sensitive. If you cut from Design view, always paste back into Design view; the same with Code view. If you don't, Dreamweaver is likely to mess up your page.*

Validating a text field with Spry

To validate a text field, either select an existing text field or position your cursor inside a form where you want to insert a new text field, and click the Spry Validation Text Field button on the Insert bar. If you are inserting a new text field, fill in the ID and Label fields in the Input Tag Accessibility Attributes dialog box as described earlier in the chapter.

Figure 9-13 shows what happens when you apply a validation widget to the first text field in the form that you have been working with throughout the chapter. The screenshot was taken with the Document window open in Split view, so you can see the underlying code (the section highlighted on lines 19–21).

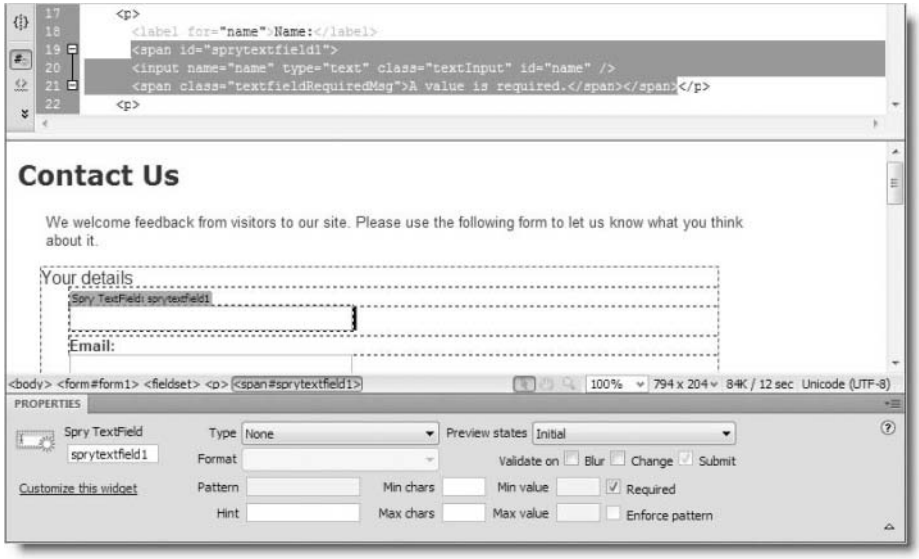


Figure 9-13. Validation widgets insert tags and control their display with JavaScript.

The <input> tag has been surrounded by a tag with the ID set to sprytextfield1. Immediately after the <input> tag is another , which contains the text: A value is required. As you can see in Figure 9-13, that text isn't displayed in Design view. This is because the display of all validation messages in Spry widgets is controlled by JavaScript.

As you can see in Figure 9-13, the text field validation widget has a lot of options in the Property inspector. Let's run through them quickly before a practical exercise to show them in action:

- Type: This is where the real power of the text field validation widget lies. It lets you check user input against a wide range of formats, summarized in Table 9-1. All options, except None, insert an Invalid format in the underlying code. Use the Preview states menu to display this in Design view for editing and/or styling.
- Format: This displays a drop-down menu of available formats, depending on the value of Type (see Table 9-1). It is disabled if the validation type is not associated with any formats.
- Pattern: Some validation types accept a custom pattern, which should be entered in this field. See "Building your own custom pattern" later in this chapter.
- Hint: This displays default text that disappears as soon as the text field has focus or anything is entered into it. It's useful for indicating the type of input or format expected. The value is displayed dynamically, so it won't be submitted as part of the form data if the user enters nothing in the field.
- Preview states: This controls the display of validation messages in Design view, allowing you to see what they look like and edit them and their associated styles.

- Validate on: This determines when the field is validated, namely:
  - Blur: This validates the input when focus moves from the field to another part of the page, for example when the user moves to the next input field.
  - Change: This validates the input each time the field changes. You should rarely use this on a text field, because it performs the validation each time the user types or deletes a character.
  - Submit: Validation is always performed when the form is submitted, so this checkbox is read-only.
- Min/Max chars: These fields let you specify the minimum or maximum number of characters required for validation. They add an alert message in a , and the Preview states menu is updated to include an option to display and edit the alert.
- Min/Max value: These let you set a minimum or maximum value for validation.
- Required: This makes the field required. It is selected by default.
- Enforce pattern: This blocks invalid characters. For example, if Type is set to Integer, nothing is entered in the field if the user attempts to type a letter.

Table 9-1. Formats that the text field validation widget can recognize

Type	Available formats	Notes
None		Use this when no other suitable format is available.
Integer		This validates whole numbers only. Negative numbers are accepted but not decimal fractions or thousands separators. Use Real Number/Scientific Notation for decimals or Currency for whole numbers with thousands separators.
Email address		This performs only a rudimentary check for an email address, making sure that it contains a single @ mark followed by at least one period.
Date	mm/dd/yy mm/dd/yyyy dd/mm/yyyy dd/mm/yy yy/mm/dd yyyy/mm/dd mm-dd-yy dd-mm-yy yyyy-mm-dd mm.dd.yyyy dd.mm.yyyy	This checks not only the format but also the validity of the date, rejecting impossible dates, such as September 31. Leap years are recognized. A bug in Dreamweaver CS3 that incorrectly rejected February 29, 2000, has been fixed.
Time	HH:mm HH:mm:ss hh:mm tt hh:mm:ss tt hh:mm t hh:mm:ss t	HH represents the 24-hour clock, hh the 12-hour clock. Hours before 10 *must* have a leading zero. When using the 12-hour clock, tt stands for AM or PM; t stands for A or P. Lowercase is not accepted.

Type	Available formats	Notes
Credit Card	All Visa MasterCard American Express Discover Diner's Club	Matches basic patterns for major credit cards but should not be relied upon to check for a valid card number. Numbers must be entered without hyphens or spaces.
Zip Code	US-5 US-9 UK Canada Custom Pattern	This tests only that the right combination of numbers and/or letters is used. It doesn't check whether the code exists or matches other parts of an address. See "Building your own custom pattern" for details of how to use the Custom Pattern format.
Phone Number	US/Canada Custom Pattern	US/Canada must be in the same format as (212) 555-0197. For Custom Pattern, see "Building your own custom pattern."
Social Security Number	US/Canada Custom Pattern	This has been updated since Dreamweaver CS3 to accept a custom pattern.
Currency	`1,000,000.00` `1.000.000,00`	In both formats, the thousands separator is optional, as is the decimal fraction. This makes it possible to validate currencies, such as yen, which aren't normally quoted with a smaller unit.
Real Number/ Scientific Notation		Used for numbers with a decimal fraction, which can optionally be expressed in scientific (exponential) notation, for example, 3.14159, 1.56234E+29, or 1.56234e29. The letter E can be uppercase or lowercase, but it must not be preceded by a space.
IP Address	IPv4 only IPv6 only IPv6 and IPv4	Covers all formats of IP address.
URL		This converts the URL to punycode (`http://en.wikipedia.org/wiki/Punycode`) before validation, so it should also accept international URLs that contain non-Latin characters.
Custom		This allows you to define your own format as described in "Building your own custom pattern."

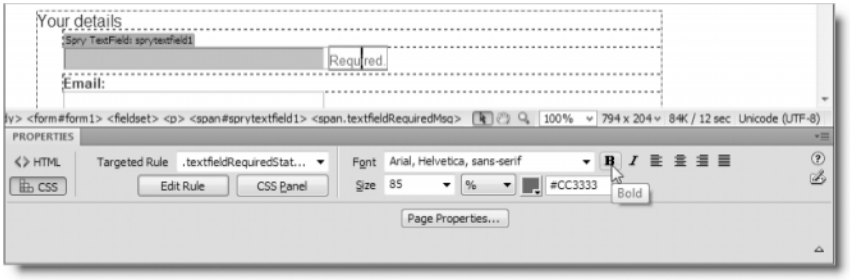
Editing and controlling the display of validation alerts

The following exercise shows you how to control the display of validation alerts in a form. It uses the same form as has been used throughout this chapter. Continue using the form you built earlier. Alternatively, copy `feedback_spry_start.php` from `examples/ch09`, and save it in `workfiles/ch09` as `feedback_spry.php`.

1. Select the Name text input field in Design view, and click the Spry Validation Text Field button in the Forms or Spry tab of the Insert bar (or use Insert ➤ Spry ➤ Spry Validation Text Field). Save the page to copy the external JavaScript file and style sheet to your site.
2. Make sure there's a check mark in the Required checkbox in the Property inspector (it should be selected by default), and choose Required from the Preview states drop-down menu. The text field should now look like this in Design view:

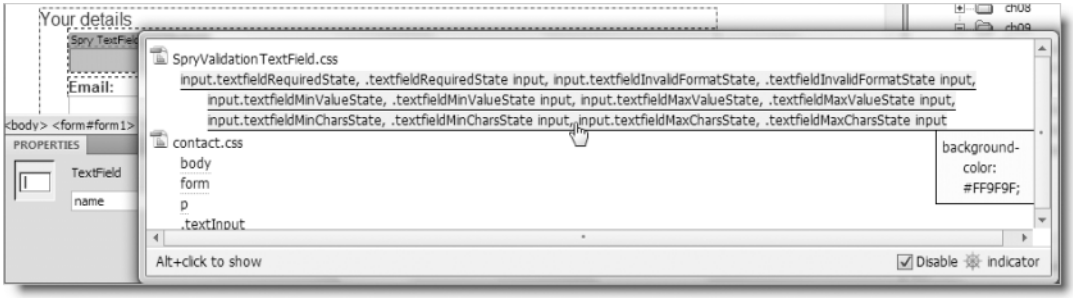
Not only is the text displayed, the background color of the text field has turned an alarming shade of pink.

3. Both the text field and the validation message are highlighted, so click inside the message so you can edit it. Shorten the text to Required.
4. With your cursor still inside the validation message, select the CSS view of the Property inspector, and click the Bold button, as shown here:



It's very important to use the CSS view of the Property inspector because this changes the targeted rule in the external style sheet, `SpryValidationTextField.css`, rather than in the underlying HTML. As a result, *all* validation messages will now be styled with bold text, not just the one you're currently editing.

5. Select the text field in Design view. The CSS view of the Property inspector is no longer visible, so you need to access the style sheet directly to change the background color of the text field. Hold down the Alt key (or Opt+Cmd keys on a Mac) to open the Code Navigator. As you can see in the following screenshot, the background color of the text field is controlled by a very complex selector:

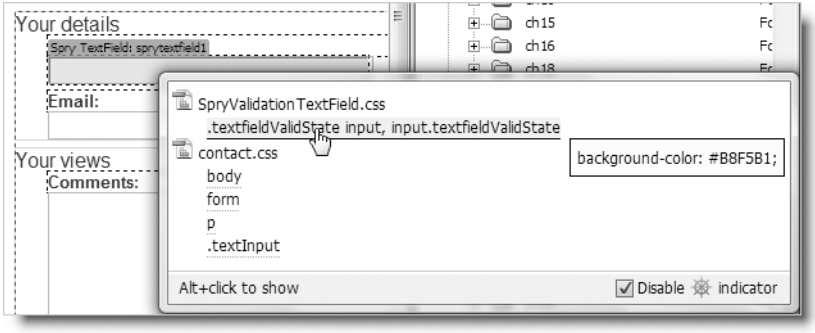


The selector is complex because it controls the look of the text field when validation fails in a wide range of circumstances. Don't worry about the selector. Just click its link in the Code Navigator to open the style sheet in Split view. Your cursor should automatically be located inside the right style rule.

6. Change the value of the `background-color` property from #FF9F9F to a less dramatic pink. I chose #FFDFDF. You can check the result by pressing F5 to refresh Design view.

7. Click the turquoise tab at the top left of the widget. In the Property inspector, change Preview states to Valid. The background color of the text field changes to green.

8. Select the text field, and then hold down the Alt/Opt+Cmd key(s) to open the Code Navigator. Select the style rule from `SpryValidationTextField.css`, as shown in the following screenshot:



It's important to select the text field first. If you open the Code Navigator by holding down Alt/Opt+Cmd and clicking without first selecting the input field, Dreamweaver cannot detect the correct style rule. This is because the classes are dynamically generated by Spry, not hard-coded into the HTML tags.

9. Change the value of the `background-color` property from #B8F5B1 to a different shade of green. I chose #E3FBE1.

10. Select File ➤ Save All to save the page and style sheet, and load `feedback_spry.php` into a browser. Click inside the Name field. Assuming you're using a modern browser and JavaScript is enabled, the field should turn yellow, indicating that it has focus.
11. Don't enter anything in the field, but move the focus to another field. The Name field reverts to its previous state.
12. Click the Send comments button. The background of the text field turns pink, and the word Required is displayed alongside in bold crimson text. Also note that nothing is displayed below the Send comments button. The file `feedback_spry.php` contains the PHP script used earlier to display the data submitted by the form, so this is confirmation that the validation widget prevented the form from being submitted.
13. Type your name in the Name field, and move the focus to another field. Although the field turns yellow while you're typing, it turns pink again when the focus moves to another field, and the Required alert isn't cleared, as the following screenshot shows:

    This is because the default behavior is to validate form elements only when the form is submitted, although you can easily change that.
14. Click the Send comments button. If your monitor is large enough for you to still see the text field, you'll see the background momentarily turn green indicating that it passed validation. You'll also see the form data displayed at the bottom of the page.
15. Back in Dreamweaver, select the turquoise tab at the top left of the validation widget to display its details in the Property inspector. Select the Blur checkbox, save the page, and repeat steps 10–14 to test it again. This time, the field turns green, and the Required message disappears in step 13.

    Check your code, if necessary, against `feedback_spry_text.php` and `SpryValidationTextField_edit.css` in `examples/ch09`.

The styles changed in the preceding exercise affect all text field validation widgets in the same page, and they apply equally to all text field validation alerts. Although the Preview states menu gives you access to most style rules, you might want to edit the following two selectors directly in `SpryValidationTextField.css`:

- `.textfieldFocusState input, input.textfieldFocusState`: This gives the text field a yellow background when it has focus. The default color is `#FFFFCC`.
- `.textfieldFlashText input, input.textfieldFlashText`: This applies only when you select Enforce pattern in the Property inspector, and it makes the text briefly flash red if an invalid character is inserted.

Styling the alert messages for all remaining validation widgets follows the same principles as for a text field. Study the style sheets in the Spry assets folder, or click the Customize this widget *link in the Property inspector to display the help file, which explains which style rules to change.*

Building your own custom pattern

Spry makes it easy to build custom patterns using special pattern characters that act as a mask for the user's input. Spry custom patterns aren't as powerful as regular expressions, but they're a lot easier to use, so it's a reasonable trade-off for most people. Table 9-2 describes the special pattern characters.

Table 9-2. Special characters used for building custom patterns in Spry

Character	Matches	Case sensitivity
0	Any number 0–9	
A	Any letter A–Z	Converted to uppercase
a	Any letter a–z	Converted to lowercase
B	Any letter A–Z	Original case preserved
b	Any letter A–Z	Original case preserved
X	Any alphanumeric character (A–Z and 0–9)	Letters converted to uppercase
x	Any alphanumeric character (A–Z and 0–9)	Letters converted to lowercase
Y	Any alphanumeric character (A–Z and 0–9)	Letters preserve original case
y	Any alphanumeric character (A–Z and 0–9)	Letters preserve original case
?	Any character	

Although there are ten special pattern characters, you need concern yourself with only eight of them, because uppercase and lowercase B are identical. So are uppercase and lowercase Y.

When using a custom pattern, you must select the Enforce pattern *checkbox at the bottom right of the Property inspector (see Figure 9-13 earlier in the chapter).*

Any other character included in a custom pattern is treated as an auto-complete character. For example, let's say you have a stock code that looks like this: BC-901/c. If all stock codes follow the same pattern of two uppercase letters followed by a hyphen, three digits, a forward slash, and a lowercase letter, you could use the following custom pattern:

```
AA-000/a
```

Immediately after the first two letters are inserted, Spry automatically inserts the hyphen. Then after the next thee digits, it inserts the forward slash ready for the user to insert the final letter.

If you want to use any of the special characters listed in Table 9-2, you must precede them with a backslash (for example, \A). To insert a backslash as part of an auto-complete sequence, use a double backslash (\\).

Validating a text area with Spry

Unlike a text field, the `<textarea>` tag doesn't have any way in HTML to control the acceptable number of characters, so the text area validation widget optionally displays a counter that tells the user how many have been entered or can still be entered. This is important when inserting text in a database, because the text is truncated if the user inputs more than the maximum accepted by the database column. With Spry, this is no longer a problem, because you can block further input once the maximum has been reached.

To validate a text area, either select an existing text area or position your cursor in a form where you want to insert a new text area, and click the Spry Validation Text Area button in the Forms or Spry tab in the Insert bar or select Insert ➤ Spry ➤ Spry Validation Text Area. If you are inserting a new text area, fill in the ID and Label fields in the Input Tag Accessibility Attributes dialog box as described earlier in the chapter. Figure 9-14 shows the options available in the Property inspector for a text area validation widget.

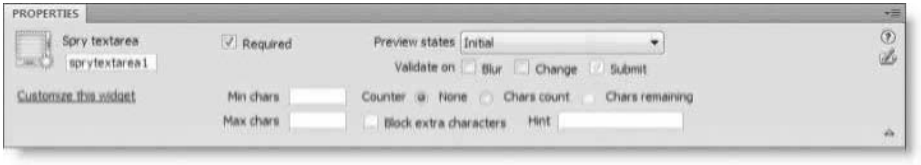


Figure 9-14. The text area validation widget has options to control and monitor the number of characters entered.

The layout of options in the Property inspector is slightly different, but Required, Preview states, Validate on, Min chars, Max chars, and Hint all work exactly the same as for a text field, so I won't explain them again (refer to "Validating a text field with Spry" if you need to refresh your memory).

Let's take a look at the two new options:

- Counter: There are three settings to choose from, as follows:
  - None: This is the default. It turns off automatic counting of characters entered.
  - Chars count: This displays the total number of characters entered. If you combine this with Validate on Change, it displays a constantly updated total (see Figure 9-15).
  - Chars remaining: This is grayed out until you enter a value in Max chars. It uses this value to calculate how many more characters can be accepted. If combined with Validate on Change, it displays a running total of characters left (see Figure 9-15).
- Block extra characters: This is self-explanatory. It prevents the user from entering more characters than the number specified in Max chars. The checkbox remains grayed out if Max chars is not specified.

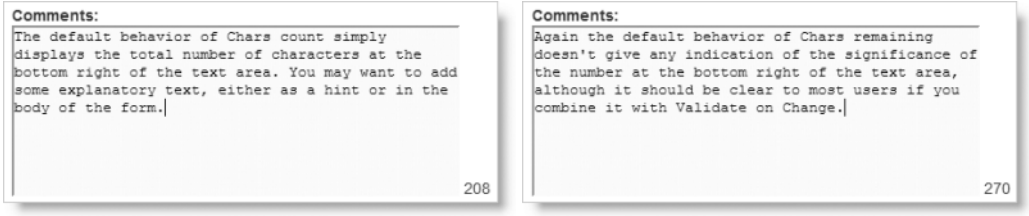
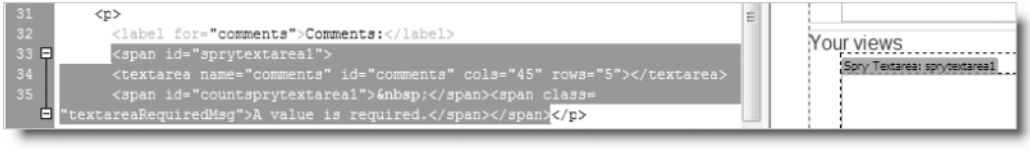


Figure 9-15. The character counter appears at the bottom right of the text area but gives no indication of its meaning.

Improving the character counter

As Figure 9-15 shows, the Spry character counter simply displays a number at the bottom right of the text area. Although most users will probably guess its meaning, it's more user-friendly to add a label to the counter. The following instructions show you how to do this. I have used `feedback_spry.php` from the previous exercise, but you can use any form with a text area.

1. In Design view, select the Comments text area, and apply a validation widget by clicking the Spry Validation Text Area button on the Insert bar (or use the Insert menu). Save the page to copy the external JavaScript file and style sheet to your site.
2. In the Property inspector, select Validate on Change, and set Counter to Chars count.
3. Open Split view to inspect the code inserted by Dreamweaver. It should look like this:

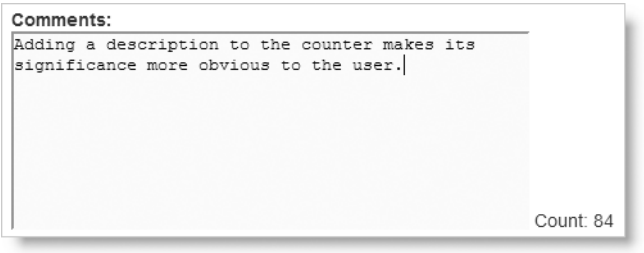


The first shown on line 35 in the preceding screenshot contains a non-breaking space ( ). Spry uses this to display the character count. Because the content of the is generated dynamically, the label needs to go outside.

4. Click in Code view, position your cursor immediately to the left of the first shown on line 35, and insert the following code shown in bold:

```
<span> Count: </span><span id="countsprytextarea1"> </span>
```

5. Save the page and test it. You should now see a more user-friendly display like this:



When using the Chars remaining option, change the text inside the new to Remaining:.

By default, Dreamweaver puts all alerts in *tags and styles them to display inline alongside the form element. This can result in the alert splitting across two lines, which makes the default border look very messy. Either shorten the text or change the style rules so that they blend in with your design. In fact, there is nothing to stop you from moving the alerts to a different position. As long as you keep the classes and IDs assigned by Dreamweaver, you can change the* *tags to other HTML elements, as demonstrated in the next exercise.*

Validating checkboxes with Spry

A common requirement on forms is a checkbox to confirm that the user agrees with certain terms and conditions. Creating this with Dreamweaver couldn't be simpler. If you already have the checkbox in your form, select it, and click the Spry Validate Checkbox button on the Insert bar. Save the page to copy the external JavaScript file and style sheet to your Spry assets folder.

If you don't have a checkbox, position your cursor where you want it to go inside the form, and click the Spry Validate Checkbox button in the Forms or Spry tab of the Insert bar. Fill in the ID and Label fields in the Input Tag Accessibility Attributes dialog box, and save the page.

That's all there is to it.

Validating a checkbox group is also easy, but the default use of tags makes it difficult to create a layout that uses valid code and looks halfway decent. However, this is also

a good opportunity to show you that you don't need to be constrained by Dreamweaver's way of doing things. The best way to explain is with a practical example based on the form you have been using throughout the chapter.

The form has a group of five checkboxes displayed in two columns, each of which is formed by a paragraph floated left. The Dreamweaver documentation tells you to add multiple checkboxes in the `<span>` created by the validation widget, but `<span>` tags cannot contain block-level elements like `<div>`, `<table>`, or `<p>`. So the best way to validate a checkbox group is to apply the widget first to a single checkbox. You can then convert the Dreamweaver code to wrap the entire group in `<div>` tags.

Applying a checkbox validation widget to a checkbox group

Continue using the page from the preceding exercises, or copy `feedback_spry_start.php` from `examples/ch09` to `workfiles/ch09` and save it as `feedback_spry.php`.

1. In Design view, select the checkbox labeled Classical concerts, and click the Spry Validation Checkbox button on the Insert bar (or use the Insert menu). Save the page to copy the external style sheet and JavaScript file to your site.
2. In the Property inspector, select the Enforce range (multiple) radio button, and type 2 in the Min # of selections field. Press Enter/Return or Tab to make sure Dreamweaver updates the validation code.
3. Open Split view to inspect the code inserted by Dreamweaver. It should look like Figure 9-16.

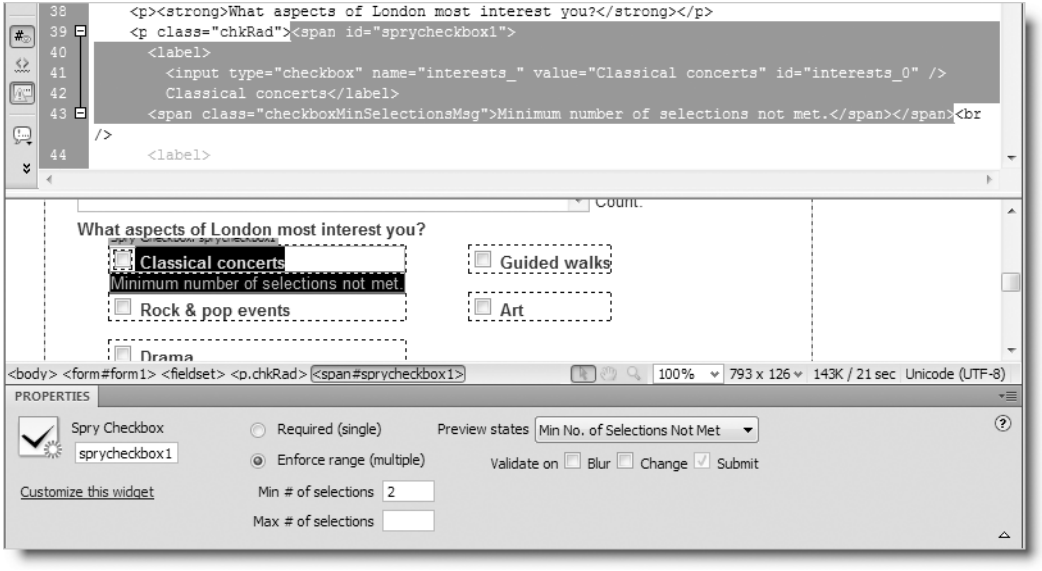


Figure 9-16. Apply a validation widget to a single checkbox, and edit the code to validate a group.

As you can see on line 39 in Figure 9-16, Dreamweaver creates an opening tag with the ID sprycheckbox1 to wrap the checkbox (the closing tag is at the end of line 43). Another at the beginning of line 43 is assigned the class checkboxMinSelectionsMsg and contains the alert message.

With Preview states set to Min No. of Selections Not Met, you can see that the alert is displayed between the checkbox and its label. It looks a mess, but not for long . . .

4. What you need to do is to convert the sprycheckbox1 into a and wrap it around the entire checkbox group.

   Switch to Code view, select the following tag shown on line 39 of Figure 9-16, and cut it to your clipboard:

   ```
   <span id="sprycheckbox1">
   ```

5. Create a new line immediately above, paste the code back into the new line, and change span to div. The resulting code should look like this (the lines above and below are included for context):

   ```
   <p><strong>What aspects of London most interest you?</strong></p>
   <div id="sprycheckbox1">
   <p class="chkRad">
   ```

6. Cut the following (it's shown on line 43 of Figure 9-16), and paste immediately below the line you moved in the previous step:

   ```
   <span class="checkboxMinSelectionsMsg">Minimum number of selections ➥
   not met.</span>
   ```

   The resulting code should look like this:

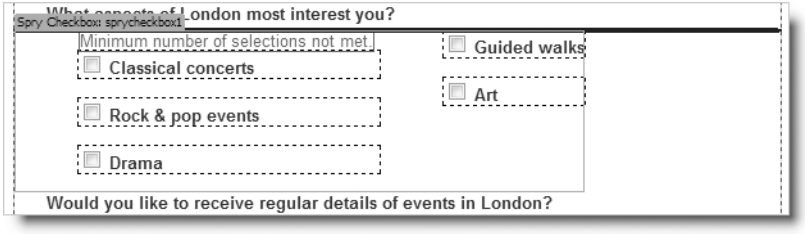
   ```
   <div id="sprycheckbox1">
   <p class="chkRad">
   <span class="checkboxMinSelectionsMsg">Minimum number of selections ➥
   not met.</span>
     <label>
     <input type="checkbox" name="interests_" value="Classical concerts" ➥
   id="interests_0" />
     Classical concerts</label>
   </span><br />
   ```

7. The closing tag highlighted in bold after the Classical concerts label is left over from the you converted into an opening tag in step 5. It needs to be converted to a closing tag and moved right to the end of the checkbox group.

   This is where a good understanding of HTML and your page structure comes in. Although it's just a case of moving a closing tag, you must get it in the correct position after the closing tag of the second chkRad class paragraph (it should now be on line 62). The following code shows the new tag in bold in its surrounding context:

```
  <label>
    <input type="checkbox" name="interests_4" id="interests_4" />
  Art</label>
</p>
</div>
<p class="clearIt"><strong>Would you like to receive regular details ➥
of events in London?</strong></p>
```

8. Switch back to Design view, and click the turquoise tab at the top left of the checkbox validation widget. The checkbox group should now look like this:



You can tell whether you have inserted the closing `</div>` tag in the right place by looking at the Spry widget's thin turquoise border. It should wrap all the checkboxes but not extend into the following line of text.

You can still display and hide the alert message using the Preview states menu in the Property inspector. The heavy blue outline around the validation widget doesn't enclose the checkboxes because they're floated. If you put the checkbox group in a nonfloated element, such as a table, the outline would enclose the whole group.

9. Select Validate on Change in the Property inspector, save the page, and test it in a browser. Select one checkbox, and the alert message should appear above the checkbox group. Select a second checkbox, and the alert disappears.

You might want to make some changes to the CSS, but this shows you how you can adapt the basic code created by Dreamweaver. This is something you will appreciate even more during the second half of this book when working with PHP. Dreamweaver provides a solid basis, but the rest is up to you.

This exercise just lifts the lid on the possibilities. I'll leave you to experiment with other variations.

Validating a radio button group with Spry

The Spry radio button group validation widget is unusual in that you cannot apply it to an existing radio button group. If anything is selected on the page, even a radio button, Dreamweaver opens the Spry Validation Radio Group dialog box and inserts a new radio button group after the selected element. If nothing is selected on the page, the new radio button group is inserted at the current insertion point.

To insert a radio button group validation widget, click the Spry Validation Radio Group button on the Forms or Spry tab of the Insert bar (or use the Insert menu). The options in the

Spry Validation Radio Group dialog box are identical to the Checkbox Group dialog box (see Figure 9-10), so refer to the description earlier in the chapter if you need help.

Figure 9-17 shows the Property inspector for a Spry radio button group validation widget. It has very few options. Most are the same as for other validation widgets, but the following two require explanation:

- Empty Value: This is used when you want to force the user to select a radio button other than the default. For example, rather than two buttons with the values yes and no, you might have a default button with the value unspecified. If you enter unspecified in this field, validation will fail until one of the other values is chosen.
- Invalid Value: This is for a button that contains an unacceptable answer. For example, if users must agree to terms and conditions before submitting a form, you could have two radio buttons with the values accept and decline. If you enter decline in this field, validation will fail, and a message will be displayed when the user selects the decline button.

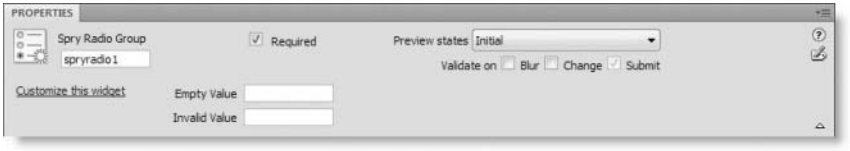


Figure 9-17. The Spry radio button group validation widget has only a small number of options.

You edit and style the validation messages in the same way as other validation widgets. To edit individual radio buttons after creating the validation widget, select the radio button element in Design view to display the normal radio button Property inspector.

Validating a drop-down menu with Spry

The Spry validation widget for `<select>` elements does not have an option to enforce multiple choices. It has options only to reject a blank or invalid value. Consequently, it's more suited to single-choice drop-down menus than multiple-choice lists.

To apply the validation widget to an existing `<select>` element, highlight the menu object in Design view, and click the Spry Validation Select button in the Forms or Spry tab of the Insert bar (or use the Insert menu). Figure 9-18 shows the available options in the Property inspector.

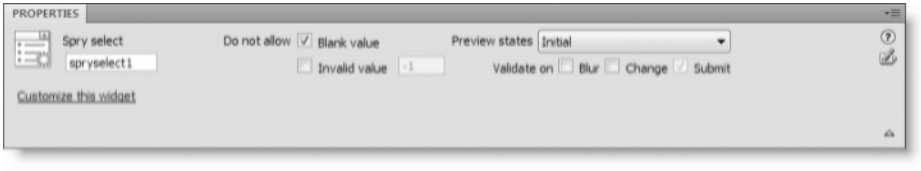


Figure 9-18. The select validation widget checks only whether a single value is blank or invalid.

Apart from those common to all validation widgets, there are just two options, namely:

- Blank value: This option is selected by default. Validation fails if the user selects a menu item that doesn't have a value. Spry considers a blank value to be an `<option>` tag either without a `value` attribute or with `value=""`.

  The `value` attribute is not required in the `<option>` tag. When it's omitted, forms use the contents of the `<option>` tag as the submitted value. If you build a menu that doesn't set the `value` attribute for each item, you *must* deselect the Blank value checkbox. Otherwise, the menu will fail validation, even when an option is selected.

- Invalid value: If you select this option, enter the value you want to be treated as invalid in the field alongside. For example, in the form built in the exercises earlier in the chapter, the `value` for -- Select one -- was set to `0`. To prevent this from being accepted, select Invalid value, and enter `0` in the field alongside.

Validating passwords with Spry

There are two validation widgets for passwords, both of which are new to Dreamweaver CS4. Since they work in combination with each other, I'll deal with them together. The first widget lets you specify criteria against which the password should be validated, while the second simply checks whether the password entered in a confirmation field matches the original password.

You can apply the password validation widget to an existing password field or use it to insert a new field. To apply it to an existing password field, select the field in Design view, and click the Spry Validation Password button on the Forms or Spry tab of the Insert bar (or use the Insert menu).

If the selected element is not a password field, Dreamweaver opens the Input Tag Accessibility Attributes dialog box for you to enter the ID and label for the password field. Even if you click OK or Cancel without filling in any of the fields, Dreamweaver inserts a new password field immediately after the currently selected element and assigns it default values.

Figure 9-19 shows the options in the Property inspector for a password validation widget. There are a lot of them, but their meaning is self-explanatory. They let you specify the strength of the password by setting a minimum and maximum number of characters, as well as upper and lower limits for letters, numbers, uppercase, and special characters. Letters are defined as unaccented letters of the Roman alphabet (A–Z, both uppercase and lowercase). Special characters are anything other than a number or unaccented letter.

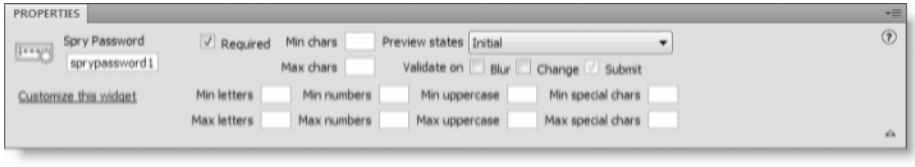


Figure 9-19. The password validation widget lets you specify the strength of the password.

You can apply the password confirmation validation widget to an existing text input or password field, or you can use it to insert a new field. However, there must already be at least one other text input or password field in the form. Otherwise, the widget will generate errors.

The password confirmation field should normally be positioned below the original password field. Dreamweaver scans the page upward to locate the first password field and uses its ID to associate the two fields with each other. As you can see in Figure 9-20, there's an option to change the field against which the password is validated.

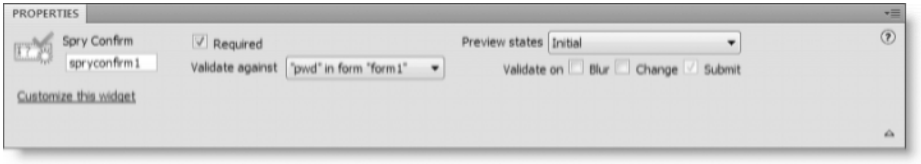


Figure 9-20. The password confirmation widget has very few options.

Chapter review

This has been a long chapter, crammed with detail, but it's an important one. You'll use forms time and again when building dynamic sites, and making sure that user input is in the right format saves endless headaches later. Spry does a lot to help with validation and is fairly easy to use, but the Dreamweaver interface could still do with some improvement. However, it's important to remember that client-side validation is only half the story. Because JavaScript can be turned off in the browser, you also need to check user input on the server side with PHP.

Moreover, forms are useless without a script capable of processing the data. The next chapter serves as a crash course in PHP basics for readers new to PHP. Then in Chapter 11, we get down to the nitty-gritty of server-side programming, using PHP to validate user input and then send it to your mail inbox.