5

Forms

Almost every time you want to collect information from a visitor to your site, you need to use a *form*. Some forms are quite complex, such as those that allow you to book plane tickets or purchase insurance online. Others are quite simple, such as the search box on the homepage of Google.

Many of the forms you will fill out online bear a strong resemblance to paper forms you have to fill out. On paper, forms are made up of areas to enter text, boxes to check (or tick), options to choose from, and so on. Similarly, on the Web you can create a form by combining what are known as *form controls*, such as textboxes (to enter text into), checkboxes (to place a cross in), select boxes and radio buttons (to choose from different options), and so on. In this chapter, you learn how each of these different types of controls can be combined into a form.

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

- ❑ How to create a form using the `<form>` element
- ❑ The different types of form controls you can use to make a form — such as text input boxes, radio buttons, select boxes, and submit buttons
- ❑ What happens to the data a user enters
- ❑ How to make your forms accessible
- ❑ How to structure the content of your forms

By the end of the chapter, you will be able to create all kinds of forms to collect information from visitors to your site.

XHTML is used only to present the form to the user; it does not allow you to say what happens with that data once it has been collected. To get a better idea of what happens to the data once it has been collected from a form, you will need to look at a book on a server-side language, such as ASP.NET or PHP (you can see a range of books on these topics at `www.wrox.com`*).*

Introducing Forms

Let's start by looking at a couple of examples of forms. First, Figure 5-1 shows you the Google homepage. This contains two kinds of form controls:

❑ A **text input**, which is where you enter your search term.

❑ **Submit buttons**, which are used to send the form to the server. There are two on this form: you can see the words "Google Search" written on the first one and "I'm Feeling Lucky" on the second.

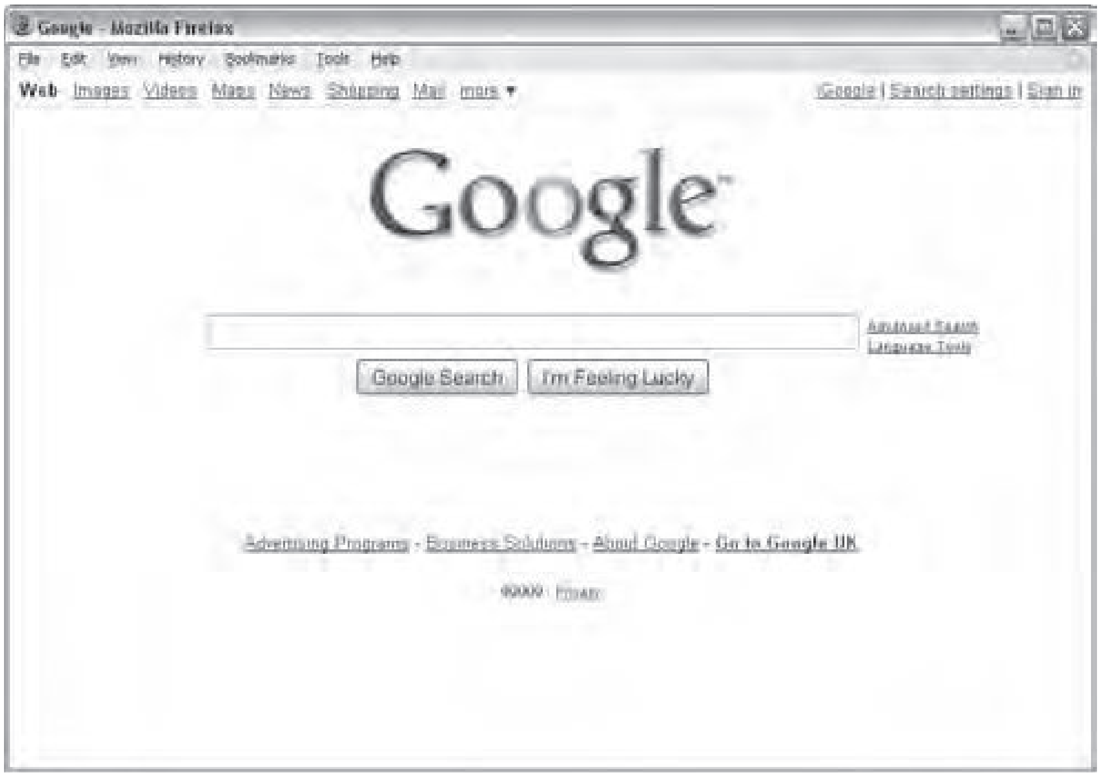

Google - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Web Images Videos Maps News Shopping Mail more ▾
iGoogle | Search settings | Sign in
Google
Advanced Search
Language Tools
Google Search
I'm Feeling Lucky
Advertising Programs - Business Solutions - About Google - Go to Google UK
©2009 - Privacy

Figure 5-1

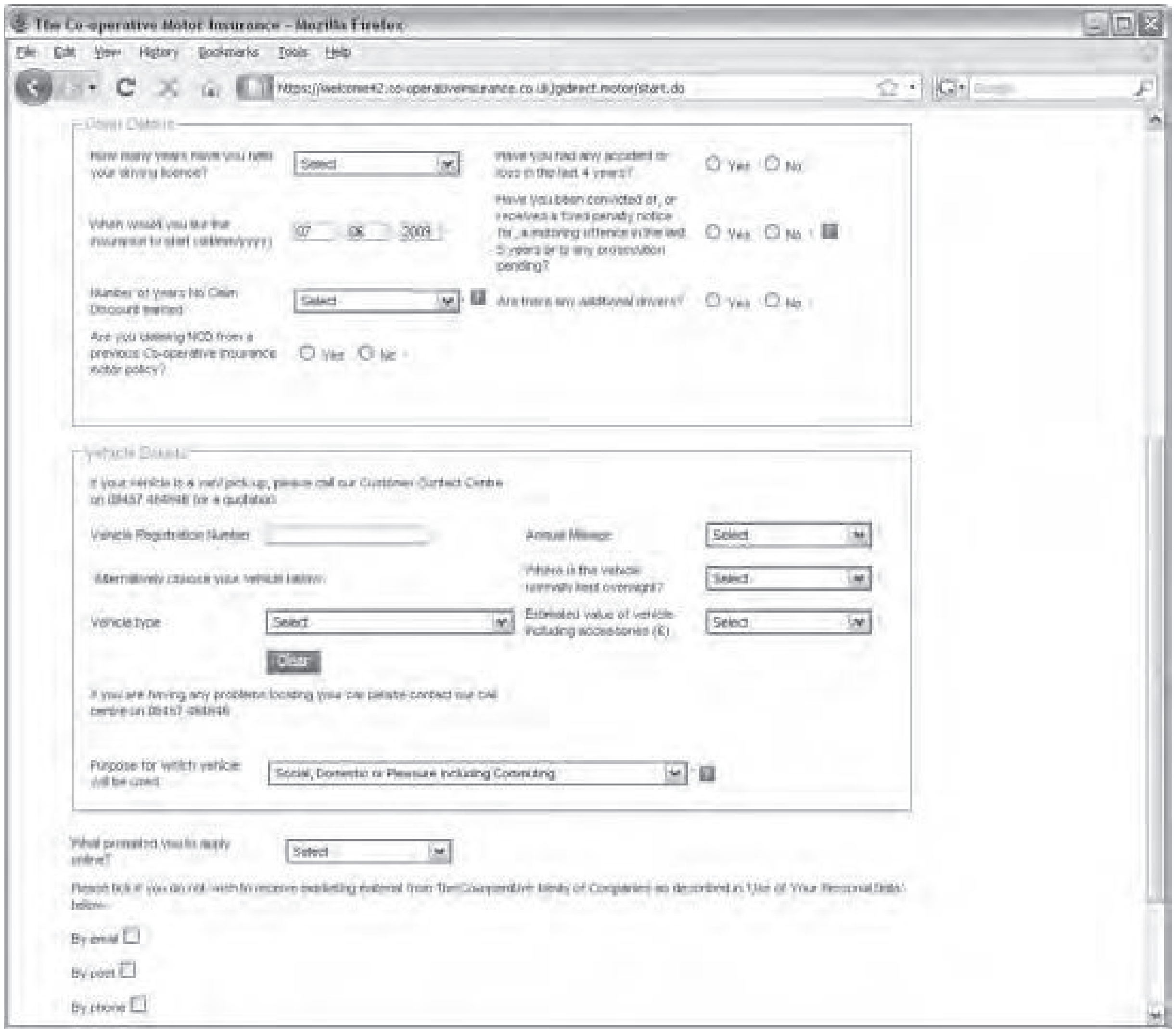
Now let's look at a more complicated example. Figure 5-2 shows part of an insurance form, which actually spreads over several pages. It shows many more types of form controls:

❑ **Select boxes**, sometimes referred to as drop-down lists, such as the one in the top left of Figure 5-2 to say how long you have been driving.

❑ **Radio buttons**, such as the ones in the top-right corner with Yes or No options. When you have a group of radio buttons, you can only pick one response.

❑ **Checkboxes**, such as the ones at the bottom of the screenshot indicating how you can be contacted (by e-mail, post, or phone). When you have a group of checkboxes, you can pick more than one response.

❑ **Text inputs**, to enter a date of birth and registration number.



The Co-operative Motor Insurance - Mozilla Firefox

File Edit View History Bookmarks Tools Help

https://welcome42.co-operativeinsurance.co.uk/gidirect.motor/start.do

Driver Details

How many years have you held your driving licence? [Select]

Have you had any accident or loss in the last 4 years? ○ Yes ○ No

When would you like the insurance to start (dd/mm/yyyy)? 07 06 2009

Have you been convicted of, or received a fixed penalty notice for, a motoring offence in the last 5 years or is any prosecution pending? ○ Yes ○ No

Number of years No Claim Discount earned [Select]

Are there any additional drivers? ○ Yes ○ No

Are you claiming NCD from a previous Co-operative Insurance motor policy? ○ Yes ○ No

Vehicle Details

If your vehicle is a van/pick up, please call our Customer Contact Centre on 08457 464646 for a quotation

Vehicle Registration Number

Annual Mileage [Select]

Alternatively choose your vehicle below:

Where is the vehicle normally kept overnight? [Select]

Vehicle type [Select]

Estimated value of vehicle including accessories (£) [Select]

Clear

If you are having any problems locating your car please contact our call centre on 08457 464646

Purpose for which vehicle will be used [Social, Domestic or Pleasure including Commuting]

What prompted you to apply online? [Select]

Please tick if you do not wish to receive marketing material from The Co-operative family of Companies as described in 'Use of Your Personal Data' below

By email ☐

By post ☐

By phone ☐

Figure 5-2

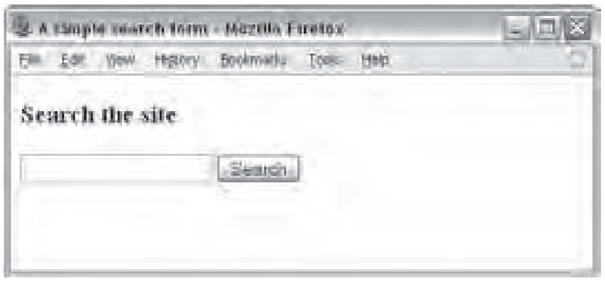
There are two additional types of form controls not shown in these examples: a text area (which is a multi-line text input), and file select boxes (which allow you to upload files), both of which you will meet later in the chapter.

Any form that you create will live inside an element called `<form>`, and the form controls (the text input boxes, drop-down boxes, checkboxes, a submit button, and so on) live between the opening `<form>` and closing `</form>` tags. A `<form>` element can also contain other XHTML markup as you would find in the rest of a page.

Once users have entered information into a form, they usually have to click what is known as a *submit button* (although the actual text on the button may say something different such as Search, Send, or Proceed — and often pressing the Return key on the keyboard has the same effect as clicking this button). This indicates that the user has filled out the form, and this usually sends the form data to a web server.

Once form data arrives at the server, a script or other program processes the data and sends a new web page back to you. The returned page will respond to a request you have made or acknowledge an action you have taken.

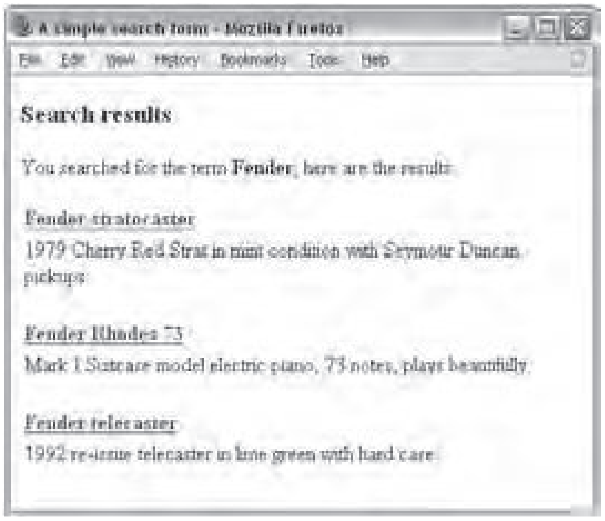
For example, you might want to add the search form shown in Figure 5-3 to your page (`ch05_eg01.html`).


A simple search form - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Search

Search the site

Figure 5-3

You can see that this form contains a textbox for the user to enter the keywords of what he or she is searching for, and a submit button, which has been set to have the word "Search" on it. When the user clicks the Search button, the information is sent to the server. The server then processes the data and generates a new page for that user telling what pages meet the search criteria (Figure 5-4).


A simple search form - Mozilla Firefox
File Edit View History Bookmarks Tools Help

Search results

You searched for the term **Fender**, here are the results

Fender stratocaster

1979 Cherry Red Strat in mint condition with Seymour Duncan pickups

Fender Rhodes 73

Mark I Suitcase model electric piano, 73 notes, plays beautifully

Fender telecaster

1992 re-issue telecaster in lime green with hard case

Figure 5-4

When a browser sends data to the server, it is transmitted in *name/value* pairs. The *name* corresponds to the name of the form control, and the *value* is what the user has entered (if the user can type an answer) or the value of the option selected (if there is a list of options to choose from).

Each item needs both a name and a value because, if you have five textboxes on a form, you need to know which data corresponds to which textbox. The processing application can then process the information from each form control individually.

Here is the code for the simple search form shown in Figure 5-3:

```
<form action="http://www.example.org/search.aspx" method="get">
   <h3>Search the site</h3>
   <input type="text" name="txtSearchItem" />
   <input type="submit" value="Search" />
</form>
```

The `<form>` element carries an attribute called `action` whose value is the URL of the page on the web server that handles search requests. Meanwhile, the `method` attribute indicates which of two HTTP methods will be used in getting the form data to the server. (You learn the difference between the `get` and `post` methods later in the chapter.)

In order to start creating forms, you first need to look at the `<form>` element in a little more detail, and then go through each of the different types of form controls and see how they sit inside the `<form>` element.

Creating a F orm with the <form> Element

As you have already seen, forms live inside an element called `<form>`. The `<form>` element can also contain other markup, such as paragraphs, headings, and so on, although it may not contain another `<form>` element.

Providing you keep your `<form>` elements separate from each other (and no `<form>` element contains another `<form>` element), your page may contain as many forms as you like. For example, you might have a login form, a search form, and a form to subscribe to a newsletter, all on the same page. If you do have more than one form on a page, users will be able to send the data from only one form at a time to the server.

Every `<form>` element should carry at least two attributes:

```
action method
```

A `<form>` element may also carry all of the universal attributes, the UI event attributes, and the following attributes:

```
enctype accept accept-charset onsubmit onreset
```

The action Attribute

The `action` attribute indicates what happens to the data when the form is submitted. Usually, the value of the `action` attribute is a page or program on a web server that will receive the information.

For example, if you had a login form consisting of a username and password, the details the user enters may get passed to a page written in ASP.NET on the web server called `login.aspx`, in which case the `action` attribute could read as follows:

```
<form action="http://www.example.org/membership/login.aspx">
```

The method Attribute

Form data can be sent to the server in two ways, each corresponding to an *HTTP method*:

- The `get` method, which sends data as part of the URL
- The `post` method, which hides data in something known as the HTTP headers

You learn more about these two methods later in the chapter, where you will learn what they mean and when you should use each one.

The id Attribute

The `id` attribute allows you to uniquely identify the `<form>` element within a page, just as you can use it to uniquely identify any element on a page.

It is good practice to give every `<form>` element an `id` attribute, because many forms make use of style sheets and scripts, which may require the use of the `id` attribute to identify the form.

The value of the `id` attribute should be unique within the document, and it should also follow the other rules for values of the `id` attribute mentioned in Chapter 1. Some people start the value of `id` and `name` attributes for forms with the characters `frm` and then use the rest of the value to describe the kind of data the form collects — for example, `frmLogin` or `frmSearch`.

The name Attribute (Deprecated)

As you have already seen through its use on other elements, the `name` attribute is the predecessor to the `id` attribute, and as with the `id` attribute, the value should be unique to the document.

Generally, you will not need to use this attribute, but when you do use it you can give it the same value as the `id` attribute. You will often see the value of this attribute begin with the characters `frm` followed by the purpose of the form (such as `frmLogin` or `frmSearch`).

The onsubmit Attribute

At some point, you have probably filled in a form on a web site, and then, as soon as you have clicked the button to send the form data (even before the page is sent to the server), been shown a message telling you that you have missed entering some data, or entered the wrong data. When this happens, the chances are you have come across a form that uses the `onsubmit` attribute to run a script in the browser that checks the data you entered before the form is sent to the server.

When a user clicks a submit button, something called an *event* fires. It is rather like the browser raising its hand and saying, “Hey, I am sending this form data to the server.” The idea behind these events is that a script (usually written in JavaScript) can be run before the data is sent to the server to check that users have filled in the necessary parts of the form in a format the server expects. The value of the `onsubmit` attribute should be a script function that would be used when this event fires.

So, an `onsubmit` attribute on the `<form>` element might look like this:

```
onsubmit="validateFormDetails();"
```

We will look at scripts in Chapters 11 and 12, but for the moment all you need to know is that, in the line of code above, the `onsubmit` attribute tells the browser that when the user presses the submit button, the browser should run the script called `validateFormDetails()`, and that this script is probably in the `<head>` element.

There are two key advantages to making some checks on the form before it is sent to the server:

- ❑ If users have missed information, they do not have to wait the extra time it would take for the page to be sent to the server and then returned with details of their errors. It is far quicker if it is checked in the browser first.
- ❑ The server does not have to receive as many forms with errors, because the browser will have already made some checks before it receives the data (therefore, saving the load on the server).

The onreset Attribute

Some forms contain a `reset` button that empties the form of all details, although the button might say something like `clear form` instead; when this button is pressed, an `onreset` event fires and a script can be run.

When the `onreset` attribute is used, its value is a script (as with the `onsubmit` attribute) that is executed when the user clicks the button that calls it.

The `onreset` *event and attribute are used a lot less than* `onsubmit`*. If you offer a Clear Form button, however, it is good to confirm with users that they did intend to clear the form before performing the action (in case they have pressed it by accident).*

The enctype Attribute

If you use the HTTP `post` method to send data to the server, you can use the `enctype` attribute to specify how the browser encodes the data before it sends it to the server. Browsers tend to support two types of encoding:

- ❑ `application/x-www-form-urlencoded`, which is the standard method most forms use. Browsers use this because some characters, such as spaces, the plus sign, and some other non-alphanumeric characters cannot be sent to the web server. Instead, they are replaced by other characters which are used to represent them.
- ❑ `multipart/form-data`, which allows the data to be sent in parts, where each consecutive part corresponds to a form control, in the order it appears in the form. It is commonly used when visitors have to upload files (such as photos) to a server. Each part can have an optional content-type header of its own indicating the type of data for that form control.

If this attribute is not used, browsers use the first value. As a result, you are likely to use this attribute only if your form allows users to upload a file (such as an image) to the server, or if they are going to use non-ASCII characters, in which case the `enctype` attribute should be given the second value:

```
enctype="multipart/form-data"
```

The accept-charset Attribute

Different languages are written in different *character sets* or groups of characters. However, when creating web sites, developers do not always build them to understand all different languages. The idea behind the `accept-charset` attribute is that it specifies a list of character encodings that a user may enter and that the server can then process. Values should be a space-separated or comma-delimited list of character sets (as shown in Appendix E).

For example, the following indicates that a server accepts UTF-8 encodings:

```
accept-charset="utf-8"
```

The accept Attribute

The `accept` attribute is similar to the `accept-charset` attribute except it takes a comma-separated list of content types (or file types) that the server processing the form can handle. Unfortunately, none of the main browsers supports this feature.

The idea is that a user would not be able to upload a file of a different content type other than those listed. Here, you can see that the only types intended to be uploaded are images that are GIFs or JPEGs:

```
accept="image/gif, image/jpg"
```

Since the main browsers currently ignore this attribute, if you were to use it visitors would still be able to upload any file. A list of MIME types appears in Appendix H.

The target Attribute

The `target` attribute is usually used with the `<a>` element to indicate which frame or browser window the link should be loaded into. It can also be used with a form to indicate which frame or window the form results open in when the user has submitted a form (frames are covered in the next chapter).

White Space and the <form> Element

You should also be aware that when a browser comes across a `<form>` element it often creates extra white space around that element. This can particularly affect your design if you want a form to fit in a small area, such as putting a search form in a menu bar. If CSS will not cure this problem in the browsers you are targeting, the only way to avoid the problem is through careful placement of the `<form>` element.

To avoid the extra space created, you can try either placing the `<form>` element near the start or end of the document, or, if you are using tables for layout purposes in a Transitional XHTML 1.0 document, between the `<table>` and `<tr>` elements. (You should be aware that this latter approach is a cheat, and therefore it might cause an error if you tried to validate the page. However, most browsers will still display the table and form as you intended.)

Form Controls

You've met the `<form>` element, so this section goes on to cover the different types of form controls that live inside the `<form>` element to collect data from a visitor to your site. You will see:

- Text input controls
- Buttons
- Checkboxes and radio buttons
- Select boxes (sometimes referred to as drop-down menus and list boxes)
- File select boxes
- Hidden controls

Text Inputs

Text input boxes are used on many web pages. Possibly the most famous text input box is the one right in the middle of the Google homepage that allows you to enter what you are searching for.

On a printed form, the equivalent of a text input is a box or line in or on which you write a response.

There are actually three types of text input used on forms:

- **Single-line text input controls:** Used for items that require only one line of user input, such as search boxes or e-mail addresses. They are created using the `<input>` element and sometimes referred to simply as "textboxes."
- **Password input controls:** These are just like the single-line text input, except they mask the characters a user enters so that the characters cannot be seen on the screen. They tend to either show an asterisk or a dot instead of each character the user types, so that someone cannot simply look at the screen to see what a user types in. Password input controls are mainly used for entering passwords on login forms or sensitive details such as credit card numbers. They are also created using the `<input>` element.
- **Multi-line text input controls:** Used when the user is required to give details that may be longer than a single sentence. Multi-line input controls are created with the `<textarea>` element.

Let's take a look at each of these types of text input in turn.

Single-Line Text Input Controls

Single-line text input controls are created using an `<input>` element whose `type` attribute has a value of `text`. Here is a basic example of a single-line text input used for a search box (`ch05_eg02.html`):

```
<form action="http://www.example.com/search.aspx" method="get"
  name="frmSearch">
  Search:
  <input type="text" name="txtSearch" value="Search for" size="20"
         maxlength="64" />
  <input type="submit" value="Submit" />
</form>
```

Figure 5-5 shows what this form looks like in a browser.

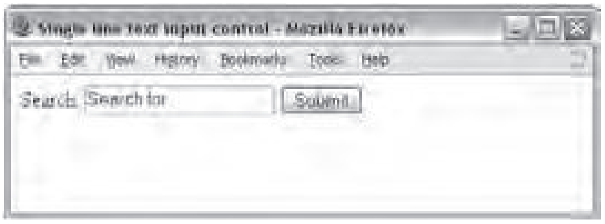

Single line text input control - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Search:
Search for
Submit

Figure 5-5

Just as some people try to start form names with the characters `frm`*, it is also common to start text input names with the characters* `txt` *to indicate that the form control is a textbox. This can be particularly handy when working with the data on the server, to remind you what sort of form control sent that data. However, some programmers prefer not to use this notation, so if you are working with someone else on a project, it is worth discussing that person's preference at the start of the work.*

The table that follows lists the attributes the `<input>` element can carry when creating a text input control. Note how the purpose of the `name` attribute is quite specific on this element, and different from its use on other elements you have met already.

Attribute	Purpose
`type`	This attribute is required, and indicates the type of input control you want to create. The value for this attribute should be `text` when you want to create a single-line text input control. The attribute is required because the `<input>` element is also used to create other form controls such as radio buttons and checkboxes.
`name`	This attribute is also required, and is used to give the name part of the name/value pair that is sent to the server (remember, each control on a form is represented as a name/value pair where the name identifies the form control and the value is what the user entered).

Attribute	Purpose
`value`	Provides an initial value for the text input control that the user will see when the form loads. You would only use this attribute if you want something to be written in the text input when the page loads (such as a cue to tell the user what he or she should be entering).
`size`	Allows you to specify the width of the text input control in terms of characters; the search box in the earlier example is 20 characters wide. The `size` property does not affect how many characters users can enter (in this case they could enter 40 characters even when the `size` property has a value of `20`); it just indicates how many characters wide the input will be. If users enter more characters than the size of the input, they can scroll right and left to see what they have entered using the arrow keys.
`maxlength`	Allows you to specify the maximum number of characters a user can enter into the text box. Usually after the maximum number of characters has been entered, even if the user keeps pressing more keys, no new characters will be added.

When an `<input>` element's `type` attribute has a value of `text`, it can also carry the following attributes:

- ❑ All of the universal attributes
- ❑ `disabled`, `readonly`, `tabindex`, and `accesskey`, which are covered later in the chapter

Password Input Controls

If you want to collect sensitive data such as passwords and credit card information, you can use the password input. The password input masks the characters the user types on the screen by replacing them with either a dot or asterisk, so that they would not be visible to someone looking over the user's shoulder.

Password input controls are created almost identically to the single-line text input controls, except that the `type` attribute on the `<input>` element is given a value of `password`.

Here you can see an example of a login form that combines a single-line text input control and a password input control (`ch05_eg03.html`):

```
<form action="http://www.example.com/login.aspx" method="post">
  Username:
  <input type="text" name="txtUsername" value="" size="20" maxlength="20" />
  <br />
  Password:
  <input type="password" name="pwdPassword" value="" size="20"
  maxlength="20" />
  <input type="submit" value="Submit" />
</form>
```

As you can see, it is common to start the name of any password with the characters `pwd` *so that when you come to deal with the data on the server, you know the associated value came from a password input box.*

Figure 5-6 shows you how this login form might look in a browser when the user starts entering details.

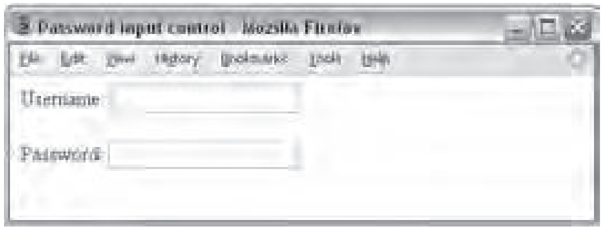

Password input control - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Username
Password

Figure 5-6

While passwords are hidden on the screen, they are still sent across the Internet as clear text, which is not considered very secure. In order to make them secure you should use an SSL connection between the client and server and encrypt any sensitive data (such as passwords and credit card details). SSL connections and encryption should be covered in a book about server-side languages such as ASP.NET and PHP.

Multiple-Line Text Input Controls

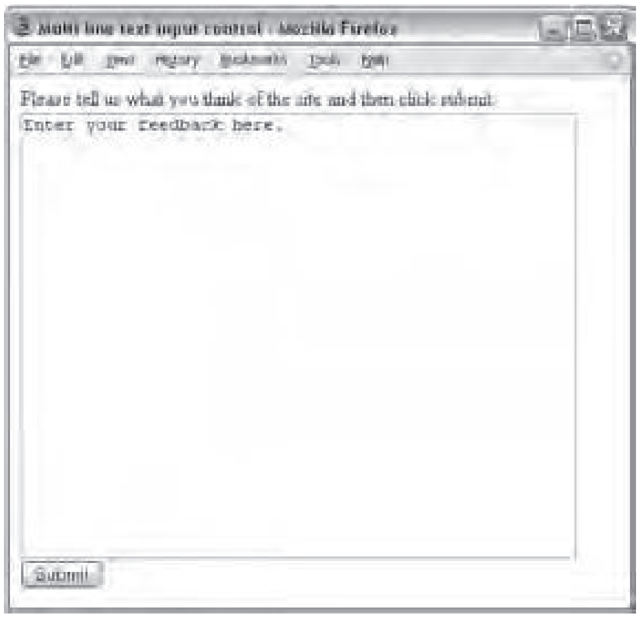
If you want to allow a visitor to your site to enter more than one line of text, you should create a multiple-line text input control using the `<textarea>` element.

Here is an example of a multiple-line text input used to collect feedback from visitors to a site (`ch05_eg04.html`):

```
<form action="http://www.example.org/feedback.aspx" method="post">
 Please tell us what you think of the site and then click submit:<br />
  <textarea name="txtFeedback" rows="20" cols="50">
Enter your feedback here.
  </textarea>
 <br />
 <input type="submit" value="Submit" />
</form>
```

Note that the text inside the `<textarea>` element is not indented (in the same way that other code in this book is indented). Anything written between the opening and closing `<textarea>` tags is treated as if it were written inside a `<pre>` element, and formatting of the source document is preserved. If the words “Enter your feedback here” were indented in the code, they would also be indented in the resulting multi-line text input on the browser.

Figure 5-7 shows what this form might look like.


Multi line text input control - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Submit

Please tell us what you think of the site and then click submit:

```
Enter your feedback here.
```

Figure 5-7

In the figure, you can see the writing between the opening `<textarea>` and closing `</textarea>` tags, which is shown in the text area when the page loads. Users can delete this text before adding their own text, and if they do not delete the text from the textbox it will be sent to the server when the form is submitted. Users often just type after any text written in a `<textarea>` element, so you may choose to avoid adding anything in between the elements, but you should still use both opening and closing `<textarea>` tags; otherwise, older browsers may not render the element correctly.

The `<textarea>` element can take the attributes shown in the table that follows.

Attribute	Purpose
`name`	The name of the control. This is used in the name/value pair that is sent to the server.
`rows`	Used to specify the size of a `<textarea>`; it indicates the number of rows of text a `<textarea>` element should have and therefore corresponds to the height of the text area.
`cols`	Used to specify the size of a `<textarea>`; it specifies the number of columns of text and therefore corresponds to the width of the box. One column is the average width of a character.

The `<textarea>` element can also take the following attributes:

- ❑ All of the universal attributes
- ❑ `disabled`, `readonly`, `tabindex`, and `accesskey`, which are covered later in the chapter
- ❑ The UI event attributes

By default, when a user runs out of columns in a `<textarea>`, the text is wrapped onto the next line (which means it just flows onto the next line as text in a word processor does), but the server will receive it as if it were all on one line. Because some users expect the sentences to break where they see them break on the screen, the major browsers also support an extra attribute called `wrap` that allows you to indicate how the text should be wrapped. Possible values are as follows:

- ❑ `off` (the default), which means scrollbars are added to the box if the user's words take up more space than the allowed width, and users have to scroll to see what they have entered
- ❑ `virtual`, which means that wherever the text wraps, users see it on the new line but it is transmitted to the server as if it were all on the same line unless the user has pressed the Enter key, in which case it is treated as a line break
- ❑ `physical`, which means that wherever the user sees the text start on a new line, so will the server

The `wrap` attribute is not, however, part of the XHTML specification.

Buttons

Buttons are most commonly used to submit a form, although they are sometimes used to clear or reset a form and even to trigger client-side scripts. (For example, on a basic loan calculator form within the page, a button might be used to trigger the script that calculates repayments without sending the data to the server.) You can create a button in three ways:

- ❑ Using an `<input>` element with a `type` attribute whose value is `submit`, `reset`, or `button`
- ❑ Using an `<input>` element with a `type` attribute whose value is `image`
- ❑ Using a `<button>` element

With each different method, the button will appear slightly different.

Creating Buttons Using the <input> Element

When you use the `<input>` element to create a button, the type of button you create is specified using the `type` attribute. The `type` attribute can take the following values to create a button:

- ❑ `submit`, which creates a button that submits a form when pressed
- ❑ `reset`, which creates a button that automatically resets form controls to their initial values as they were when the page loaded
- ❑ `button`, which creates a button that is used to trigger a client-side script when the user clicks that button

Here you can see examples of all three types of button (`ch05_eg05.html`)

```
<input type="submit" name="btnVoteRed" value="Vote for reds" />
<input type="submit" name="btnVoteBlue" value="Vote for blues" />
<br /><br />
<input type="reset" value="Clear form" /> <br /><br />
<input type="button" value="calculate" onclick="calculate()" />
```

Figure 5-8 shows what these buttons might look like in Firefox on a PC (a Mac displays them in the standard Mac style for buttons).

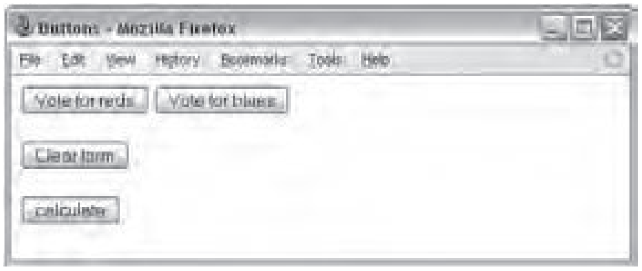

Buttons - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Vote for reds
Vote for blues
Clear form
calculate

Figure 5-8

The table that follows shows the attributes used by the buttons.

Attribute	Purpose
`type`	Specifies the type of button you want and takes one of the following values: `submit`, `reset`, or `button`.
`name`	Provides a name for the button. You need to add only a `name` attribute to a button if there is more than one button on the same form (in which case it helps to indicate which of the buttons was clicked). It is considered good practice, however, to give the button a name anyway to provide an indication of what the button does.
`value`	Enables you to specify what the text on the button should read. If a `name` attribute is given, the value of the `value` attribute is sent to the server as part of the name/value pair for this form control. If no value is given, then no name/value pair is sent for this button.
`onclick`	Used to trigger a script when the user clicks the button; the value of this attribute is the script that should be run.

In the same way that you can trigger a script when the user clicks a button, you can also trigger a script when the button gains or loses focus with the `onfocus` and `onblur` event attributes.

When an `<input>` element has a `type` attribute whose value is `submit`, `reset`, or `button`, it can also take the following attributes:

- ❑ All the universal attributes
- ❑ `disabled`, `readonly`, `tabindex`, and `accesskey`, which are discussed later in the chapter
- ❑ The UI event attributes

If you do not use the `value` attribute on the submit button, you may find that a browser displays text that is inappropriate to the purpose of the form — for example, IE displays the text `Send Query`, which is not ideal for a login button form.

Using Images for Buttons

You can use an image for a button rather than using the standard button that a browser renders for you. Creating an image button is very similar to creating any other button, but the `type` attribute has a value of `image`:

```
<input type="image" src="submit.jpg" alt="Submit" name="btnImage" />
```

Note how you can start the value of a `name` *attribute for a button with the characters* `btn`*, in keeping with the naming convention that I mentioned earlier. (When you refer to the name of the form control in other code, the use of this prefix will help remind you what type of form control the information came from.)*

Because you are creating a button that has an image, you need to have two additional attributes, which are listed in the table that follows.

Attribute	Purpose
`src`	Specifies the source of the image file.
`alt`	Provides alternative text for the image. This will be displayed when the image cannot be found and also read to people using voice browsers. (It was first supported only in IE5 and Netscape 6.)

If the image button has a `name` attribute, when you click it, the browser sends a name/value pair to the server. The name will be what you provide for the `name` attribute and the value will be a pair of *x* and *y* coordinates for where on the button the user clicked (just as you saw when dealing with server-side image maps in Chapter 3).

In Figure 5-9, you can see a graphical submit button. Both Firefox and IE change the cursor when the user hovers over one of these buttons to help users know that they can click on it.

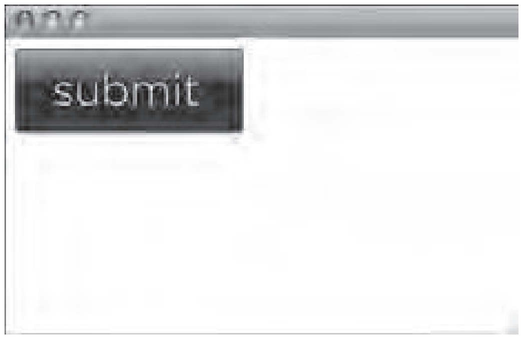


Figure 5-9

Creating Buttons Using the <button> Element

The `<button>` element is a more recent introduction that allows you to specify what appears on a button between an opening `<button>` tag and a closing `</button>` tag. So you can include textual markup or image elements between these tags.

This element was first supported in IE4 and Netscape 6, but the browsers that do support this element also offer a relief (or 3D) effect on the button, which resembles an up or down motion when the button is clicked.

Here are some examples of using the `<button>` element (`ch06_eg06.html`):

```
<button type="submit">Submit</button>
<br /><br />
<button type="reset"><b>Clear this form,</b> I want to start again</button>
<br /><br />
<button type="button"><img src="submit.gif" alt="submit" /></button>
```

As you can see, the first submit button just contains text, the second reset button contains text and other markup (in the form of the `<b>` element), and the third submit button contains an `<img>` element.

Figure 5-10 shows what these buttons would look like.

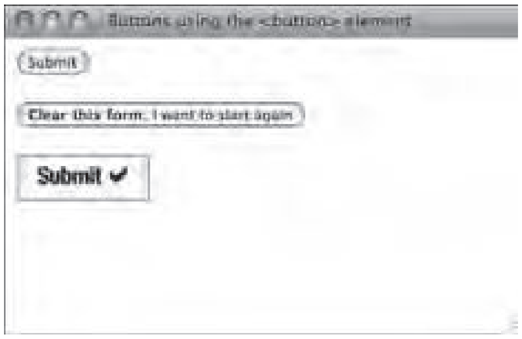

Buttons using the <button> element
Submit
Clear this form, I want to start again
Submit ✔

Figure 5-10

Checkboxes

Checkboxes are just like the little boxes on paper forms in which you can place a cross or tick. As with light switches, they can be either on or off. When they are checked they are on — the user can simply toggle between on and off positions by clicking the checkbox.

Checkboxes can appear individually, with each having its own name, or they can appear as a group of checkboxes that share a control name and allow users to select several values for the same property.

Checkboxes are ideal form controls when you need to allow a user to:

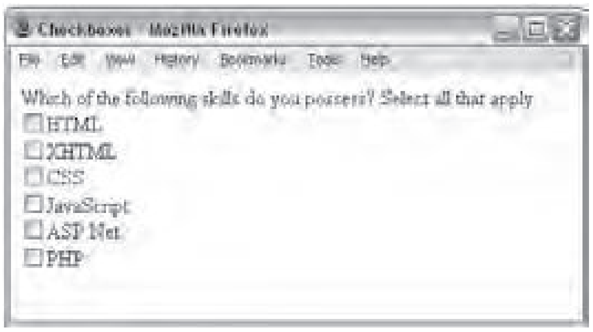
- ❑ Provide a simple yes or no response with one control (such as accepting terms and conditions)
- ❑ Select several items from a list of possible options (such as when you want a user to indicate all the skills they have from a given list)

A checkbox is created using the `<input>` element whose `type` attribute has a value of `checkbox`. Following is an example of some checkboxes that use the same control name (`ch05_eg07.html`):

```
<form action="http://www.example.com/cv.aspx" method="get" name="frmCV">
Which of the following skills do you possess? Select all that apply.
   <input type="checkbox" name="chkSkills" value="xhtml" />XHTML <br />
   <input type="checkbox" name="chkSkills" value="CSS" />CSS<br />
   <input type="checkbox" name="chkSkills" value="JavaScript" />JavaScript<br />
   <input type="checkbox" name="chkSkills" value="aspnet" />ASP.Net<br />
   <input type="checkbox" name="chkSkills" value="php" />PHP
</form>
```

For consistency with the naming convention we have used for form elements throughout the chapter, you can start the name of checkboxes with the letters `chk`.

Figure 5-11 shows how this form might look in a browser. Note how there is a line break after each checkbox, so that it clearly appears on each line (if you have checkboxes side by side, users are likely to get confused about which label applies to which checkbox).


Checkboxes - Mozilla Firefox
File Edit View History Bookmarks Tools Help

Which of the following skills do you possess? Select all that apply.
☐ HTML
☐ XHTML
☐ CSS
☐ JavaScript
☐ ASP.Net
☐ PHP

Figure 5-11

Because all the selected skills will be sent to the processing application in the form of name/value pairs, if someone selects more than one skill there will be several name/value pairs sent to the server that all share the same name.

In contrast, here is a single checkbox, acting like a simple yes or no option:

```
<form action="http://www.example.org/accept.aspx" name="frmTandC"
method="get">
  <input type="checkbox" name="chkAcceptTerms" checked="checked" />
   I accept the <a href="terms.htm">terms and conditions</a>.<br />
  <input type="submit" />
</form>
```

Note how the `<input>` element that creates this checkbox does not carry a `value` attribute. In the absence of a `value` attribute, the value is `on`. In this example, you can also see an attribute called `checked`, with a value of `checked`, which indicates that when the page loads the checkbox is selected.

The table that follows shows the attributes that an `<input>` element whose `type` attribute has a value of `checkbox` can carry.

Attribute	Purpose
`type`	Indicates that you want to create a checkbox.
`name`	Gives the name of the control. Several checkboxes may share the same name, but this should happen only if you want users to have the option of selecting several items from the same list — in which case, they should be placed next to each other on the form.
`value`	The value that will be sent to the server if the checkbox is selected.
`checked`	Indicates that when the page loads the checkbox should be selected.

Checkboxes can also carry the following attributes:

- ❑ All universal attributes
- ❑ `disabled`, `readonly`, `tabindex`, and `accesskey` which are discussed later in the chapter
- ❑ UI event attributes

Radio Buttons

Radio buttons are similar to checkboxes in that they can be either on or off, but there are two key differences:

- ❑ When you have a group of radio buttons that share the same name, only one of them can be selected. Once one radio button has been selected, if the user clicks another option, the new option is selected and the old one deselected.
- ❑ You should not use radio buttons for a single form control where the control indicates on or off, because once a lone radio button has been selected it cannot be deselected again (without writing a script to do that).

Therefore, a group of radio buttons are ideal if you want to provide users with a number of options from which they must pick only one. In such situations, an alternative is to use a drop-down select box that allows users to select only one option from several. Your decision between whether to use a select box or a group of radio buttons depends on three things:

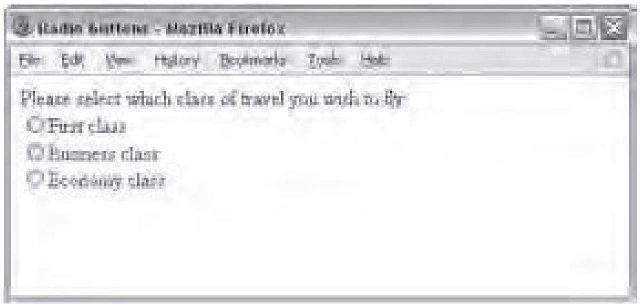
- ❑ **User expectations:** If your form models a paper form where users would be presented with several checkboxes, from which they can pick only one, then you should use a group of radio buttons.
- ❑ **Seeing all the options:** If users would benefit from having all the options in front of them before they pick one, you should use a group of radio buttons.
- ❑ **Space:** If you are concerned about space, a drop-down select box will take up far less space than a set of radio buttons.

The term "radio buttons" comes from old radios. On some old radios, you could press only one button at a time to select the radio station you wanted to listen to from the ones that had been set. You could not press two of these buttons at the same time on your radio, and pressing one would pop the other out.

The `<input>` element is again called upon to create radio buttons, and this time the `type` attribute should be given a value of `radio`. For example, here radio buttons are used to allow users to select which class of travel they want to take (`ch05_eg08.html`):

```
<form action="http://www.example.com/flights.aspx" name="frmFlightBooking"
      method="get">
   Please select which class of travel you wish to fly: <br />
  <input type="radio" name="radClass" value="First" />First class <br />
  <input type="radio" name="radClass" value="Business" />Business class <br />
  <input type="radio" name="radClass" value="Economy" />Economy class <br />
</form>
```

As you can see, the user should be allowed to select only one of the three options, so radio buttons are ideal. You can also start the name of a radio button with the letters `rad`. Figure 5-12 shows you what this might look like in a browser.


Radio buttons - Mozilla Firefox
File Edit View History Bookmarks Tools Help

Please select which class of travel you wish to fly:
First class
Business class
Economy class

Figure 5-12

The table that follows lists the attributes for an `<input>` element whose `type` attribute has a value of `radio`.

Attribute	Purpose
`type`	To indicate that you want a radio button form control.
`name`	The name of the form control.
`value`	Used to indicate the value that will be sent to the server if this option is selected.
`checked`	Indicates that this option should be selected by default when the page loads. Remember that there is no point using this with a single radio button as a user can't deselect the option. If you use this attribute, the value should also be checked in order for the attribute to be XHTML-compliant.
`size`	This attribute indicates the size of the radio button in pixels, but this attribute does not work in IE8 or Firefox 3.

Radio buttons can also take the following attributes:

- All the universal attributes
- All the UI event attributes
- `disabled`, `tabindex`, and `accesskey`, which are covered later in the chapter

When you have a group of radio buttons that share the same name, some browsers will automatically select the first option as the page loads, even though they are not required to do so in the HTML specification. Therefore, if your radio buttons represent a set of values — say for a voting application — you might want to set a medium option to be selected by default so that, should some users forget to select one of the options, the results are not overly biased by the browser's selection. To do this, you should use the checked attribute.

Select Boxes

A drop-down select box allows users to select one item from a drop-down menu. Drop-down select boxes can take up far less space than a group of radio buttons.

Drop-down select boxes can also provide an alternative to single-line text input controls where you want to limit the options that a user can enter. For example, imagine that you were asking which country someone was from. If you had a textbox, visitors from the United States could enter different options such as U.S.A., U.S., United States, America, or North America, whereas with a select box you could control the options they could enter.

A drop-down select box is contained by a `<select>` element, while each individual option within that list is contained within an `<option>` element. For example, the following form creates a drop-down select box for the user to select a color (`ch05_eg09.html`):

```
<select name="selColor">
   <option selected="selected" value="">Select color</option>
   <option value="red">Red</option>
   <option value="green">Green</option>
   <option value="blue">Blue</option>
</select>
```

As you can see here, the text between the opening `<option>` tags and the closing `</option>` tags is used to display options to the user, while the value that would be sent to the server if that option were selected is given in the `value` attribute. You can also see that the first `<option>` element does not have a value and that its content is `Select color`; this is to indicate to the user that he or she must pick one of the color choices. Finally, notice again the use of the letters `sel` at the start of the name of a select box.

Figure 5-13 shows what this would look like in a browser.

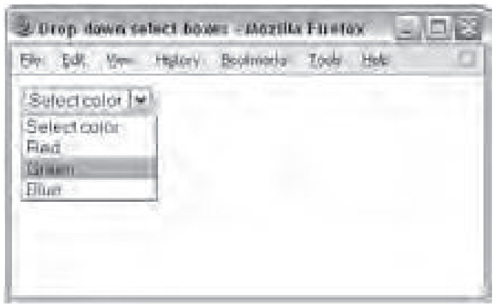

Drop down select boxes - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Select color
Select color
Red
Green
Blue

Figure 5-13

Note that the width of the select box will be the width of the longest option displayed to the user; in this case, it will be the width of the text `Select color`.

The <select> Element

The `<select>` element is the containing element for a drop-down list box; it can take the attributes shown in the table that follows:

Attribute	Purpose
`name`	The name for the control.
`size`	Can be used to present a scrolling list box, as you will see shortly. Its value would be the number of rows in the list that should be visible at the same time.
`multiple`	Allows a user to select multiple items from the menu. If the attribute is not present, the user may select only one item. In earlier versions of HTML, this attribute did not have a value. However, to be valid XHTML it should be given the value of `multiple` (i.e., `<select multiple="multiple">`). Note that the use of this attribute will change the presentation of the select box, as you will see in the section "Selecting Multiple Options with the multiple Attribute" later in this chapter.

According to the XHTML specification, a `<select>` element *must* contain at least one `<option>` element, although in practice it should contain more than one `<option>` element. After all, a drop-down list box with just one option might confuse a user.

The <option> Element

Inside any `<select>` element, you will find at least one `<option>` element. The text between the opening `<option>` and closing `</option>` tags is displayed to the user as the label for that option. The `<option>` element can take the attributes shown in the table that follows.

Attribute	Purpose
`value`	The value that is sent to the server if this option is selected.
`selected`	Specifies that this option should be the initially selected value when the page loads. This attribute may be used on several `<option>` elements even if the `<select>` element does not carry the `multiple` attribute. Although earlier versions of XHTML did not require a value for this attribute, in order to be valid XHTML you should give this attribute a value of `selected`.
`label`	An alternative way of labeling options, which uses an attribute rather than element content. This attribute is particularly useful when using the `<optgroup>` element, which is covered a bit later in this chapter.

Creating Scrolling Select Boxes

As I mentioned earlier, it's possible to create scrolling menus where users can see a few of the options in a select box at a time. In order to do this, you just add the `size` attribute to the `<select>` element. The value of the `size` attribute is the number of options you want to be visible at any one time.

While scrolling select box menus are rarely used, they can give users an indication that several possible options are open to them and allow them to see a few of the options at the same time. For example, the following is the code for a scrolling select box that allows the user to select a day of the week (`ch05_eg10.html`):

```
<form action="http://www.example.org/days.aspx" name="frmDays" method="get">
  <select size="4" name="selDay">
    <option value="Mon">Monday</option>
    <option value="Tue">Tuesday</option>
    <option value="Wed">Wednesday</option>
    <option value="Thu">Thursday</option>
    <option value="Fri">Friday</option>
    <option value="Sat">Saturday</option>
    <option value="Sun">Sunday</option>
  </select>
<br /><br /><input type="submit" value="Submit" />
</form>
```

As Figure 5-14 shows, the user can clearly see that he or she has several options; to save space only a few of the available options are shown.

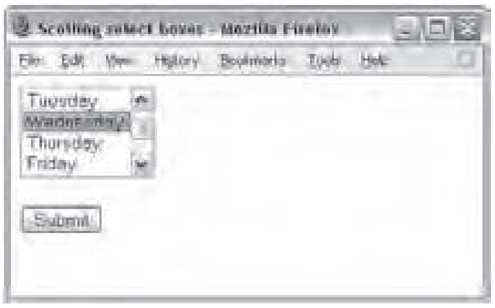

Scrolling select boxes - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Tuesday
Wednesday
Thursday
Friday
Submit

Figure 5-14

Note that the `multiple` attribute, which you meet in the next section is not used on this element.

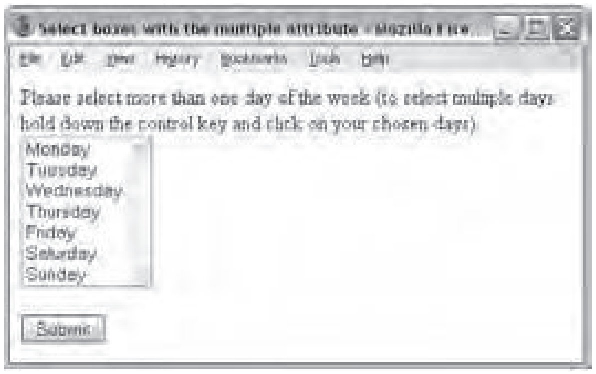
Selecting Multiple Options with the multiple Attribute

The `multiple` attribute allows users to select more than one item from a select box. The value of the `multiple` attribute should be the word `multiple` in order for it to be valid XHTML (although earlier versions of HTML allowed this attribute to appear without a value). When you use this attribute it is always a good idea to tell people how to select multiple items: by holding down the control key and clicking on the items they want to select.

The addition of this attribute automatically makes the select box look like a scrolling select box. Here you can see an example of a multiple-item select box that allows users to select more than one day of the week (`ch05_eg11.html`):

```
<form action="http://www.example.org/days.aspx" method="get" name="frmDays">
  Please select more than one day of the week (to select multiple days
    hold down the control key and click on your chosen days):<br />
  <select name="selDays" multiple="multiple">
    <option value="Mon">Monday</option>
    <option value="Tue">Tuesday</option>
    <option value="Wed">Wednesday</option>
    <option value="Thu">Thursday</option>
    <option value="Fri">Friday</option>
    <option value="Sat">Saturday</option>
    <option value="Sun">Sunday</option>
  </select>
<br /><br /><input type="submit" value="Submit">
</form>
```

The result is shown in Figure 5-15, where you can see that even without the addition of the `size` attribute, the select box is still represented in the same way as a scrolling one.


Select boxes with the multiple attribute - Mozilla Fire...
File Edit View History Bookmarks Tools Help
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
Submit

Please select more than one day of the week (to select multiple days hold down the control key and click on your chosen days):

Figure 5-15

Grouping Options with the <optgroup> Element

If you have a very long list of items in a select box, you can group them together using the `<optgroup>` element, which acts just like a container element for all the elements you want within a group.

The `<optgroup>` element can carry a `label` attribute whose value is a label for that group of options. In the following example, you can see how the options are grouped in terms of type of equipment (`ch05_eg12.html`):

```
<form action="http://www.example.org/info.aspx" method="get" name="frmInfo">
  Please select the product you are interested in:<br />
  <select name="selInformation">
    <optgroup label="Hardware">
      <option value="Desktop">Desktop computers</option>
      <option value="Laptop">Laptop computers</option>
    </optgroup>
    <optgroup label="Software">
      <option value="OfficeSoftware">Office software</option>
      <option value="Games">Games</option>
    </optgroup>
    <optgroup label="Peripherals">
      <option value="Monitors">Monitors</option>
      <option value="InputDevices">Input Devices</option>
      <option value="Storage">Storage</option>
   </optgroup>
</select>
<br /><br /><input type="submit" value="Submit" />
</form>
```

You will find that different browsers display `<optgroup>` elements in different ways. Figure 5-16 shows you how Safari on a Mac displays options held by `<optgroup>` elements, whereas Figure 5-17 shows you the result in Firefox on a PC.

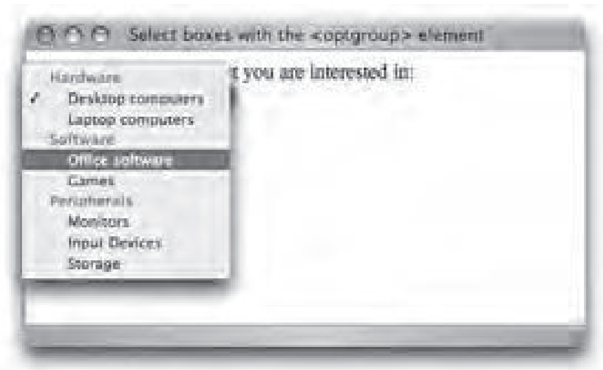


Figure 5-16

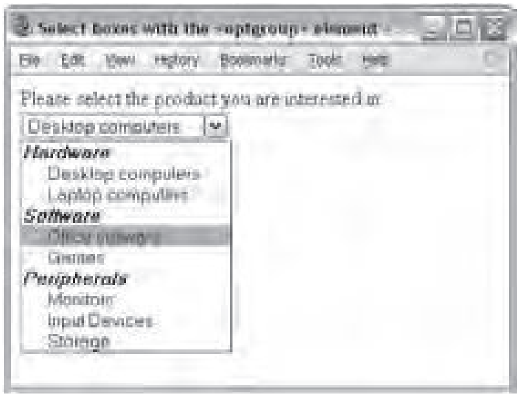

Select boxes with the <optgroup> element
File Edit View History Bookmarks Tools Help
Please select the product you are interested in:
Desktop computers
Hardware
Desktop computers
Laptop computers
Software
Office software
Games
Peripherals
Monitors
Input Devices
Storage

Figure 5-17

An alternative option for grouping elements is to add an `<option>` element that carries the `disabled` attribute, which you will learn about shortly (`ch05_eg13.html`):

```
<form action="http://www.example.org/info.aspx" method="get" name="frmInfo">
  Please select the product you are interested in:<br />
  <select name="selInformation">
    <option disabled="disabled" value=""> — Hardware — </option>
      <option value="Desktop">Desktop computers</option>
      <option value="Laptop">Laptop computers</option>
    <option disabled="disabled" value=""> — Software — </option>
      <option value="OfficeSoftware">Office software</option>
      <option value="Games">Games</option>
    <option disabled="disabled" value=""> — Peripherals — </option>
      <option value="Monitors">Monitors</option>
      <option value="InputDevices">Input Devices</option>
      <option value="Storage">Storage</option>
  </select>
<br /><br /><input type="submit" value="Submit" />
</form>
```

As you will see later in the chapter, the use of the `disabled` attribute prevents a user from selecting the option that carries it. With the careful use of a couple of dashes, the groups of options become more clearly defined, as you can see in Figure 5-18.

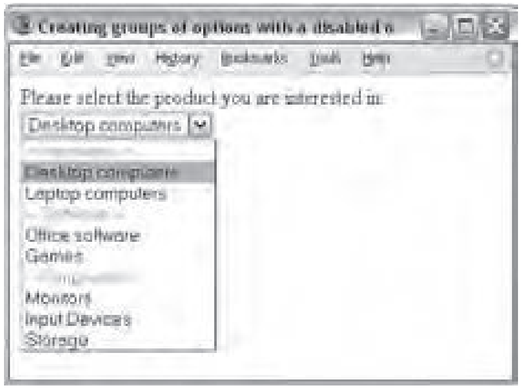

Creating groups of options with a disabled o
File Edit View History Bookmarks Tools Help
Please select the product you are interested in:
Desktop computers
Desktop computers
Laptop computers
Office software
Games
Monitors
Input Devices
Storage

Figure 5-18

You may occasionally see drop-down list boxes used for navigation, and sometimes they use JavaScript to take you directly to that page. The use of JavaScript to take you to a page without pressing a submit button is considered bad usability. One of the main reasons for this is that users can select the wrong section by accident; for example, if a user tries to select options using his or her up and down arrow keys, the script may fire as soon as he or she comes across the first option.

Attributes for Select Boxes

For completeness, the following is the full list of attributes that the `<select>` element can carry:

- `name`, `size`, and `multiple`, all of which you have met
- `disabled` and `tabindex`, which are covered later in the chapter
- All universal attributes
- UI event attributes

Meanwhile, the `<option>` element can carry the following attributes:

- `label`, which you have already seen
- `disabled`, which you learn more about later in the chapter
- All universal attributes
- UI event attributes

File Select Boxes

If you want to allow a user to upload a file to your web site from his or her computer, you will need to use a *file upload box*, also known as a *file select box*. This is created using the `<input>` element (again), but this time you give the `type` attribute a value of `file` (`ch05_eg14.html`):

```
<form action="http://www.example.com/imageUpload.aspx" method="post"
      name="fromImageUpload" enctype="multipart/form-data">
  <input type="file" name="fileUpload" accept="image/*" />
<br /><br /><input type="submit" value="Submit" />
</form>
```

When you are using a file upload box, the `method` *attribute of the* `<form>` *element must be* `post`.

There are some attributes in this example that you learned about at the beginning of the chapter.

- The `enctype` attribute has been added to the `<form>` element with a value of `multipart/form-data` so that each form control is sent separately to the server. This is required on a form that uses a file upload box.
- The `accept` attribute has been added to the `<input>` element to indicate the MIME types of the files that can be selected for upload. In this example, the `accept` attribute is indicating that any image format can be uploaded, as the wildcard character (the asterisk) has been used after the `image/` portion of the MIME type. Unfortunately, this is not supported by Firefox 3 or IE8, which means that any file at all (not just images) could be uploaded.

In Figure 5-19 you can see that when you click the Browse button in Firefox, a file dialog box opens up enabling you to browse to a file and select which one you want to upload. It is worth noting that different browsers sometimes show this control in slightly different ways (for example, Safari has a button saying Choose File instead of Browse).

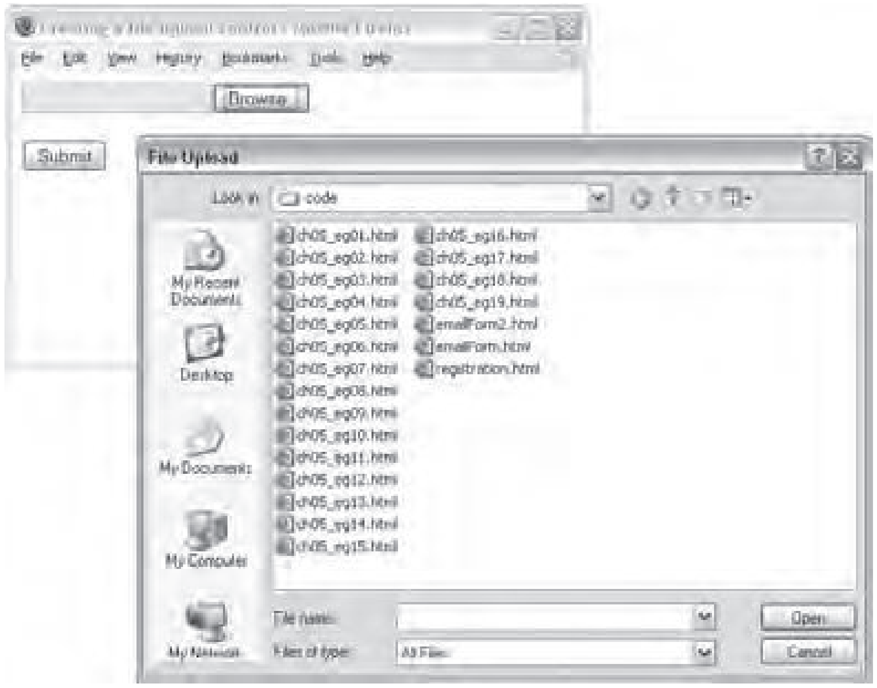


Figure 5-19

An `<input>` element whose `type` attribute has a value of `file` can take the following attributes:

- `name`, `value`, and `accept`, which you have already seen
- `tabindex`, `accesskey`, `disabled`, and `readonly`, which are covered later in the chapter
- All universal attributes
- UI event attributes

Hidden Controls

Sometimes you will want to pass information between pages without the user seeing it; to do this, you can use hidden form controls. It is important to note, however, that while users cannot see them in the web page, if they were to look at the source code for the page they would be able to see the values in the code. Therefore, hidden controls should not be used for any sensitive information that you do not want the user to see.

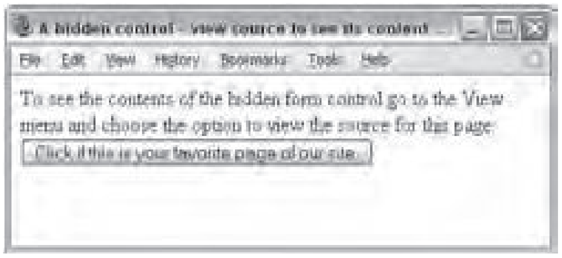
You may have come across forms on the Web that span more than one page. Long forms can be confusing and splitting them up can help a user. In such cases, it will often be necessary to pass values that a user has entered into the first form (on one page) onto the form in the second page, and then onto another page. Hidden elements are one way in which programmers can pass values between pages.

You create a hidden control using the `<input>` element whose `type` attribute has a value of `hidden`. For example, the following form contains a hidden form control indicating which section of the site the user was on when he or she filled in the form (`ch05_eg15.html`):

```
<form action="http://www.example.com/vote.aspx" method="get" name="fromVote">
  <input type="hidden" name="hidPageSentFrom" value="home page" />
  <input type="submit" value="Click if this is your favorite page of our
  site." />
</form>
```

Hidden form controls need both the `name` and `value` attributes in order to be sent with the rest of a form.

Figure 5-20 shows that the hidden form control is not shown on the page, but it is available in the source for the page.


A hidden control - view source to see its content ...
File Edit View History Bookmarks Tools Help
Click if this is your favorite page of our site.

To see the contents of the hidden form control go to the View menu and choose the option to view the source for this page.

Figure 5-20

As you will see in Chapter 8, you can also hide form controls using the CSS `display` *and* `visibility` *properties.*

Object Controls

The HTML 4.0 specification introduced the capability to use objects — embedded in an `<object>` element — inside the `<form>` element. For example, you may want to use an object that enables some kind of graphical interaction, and then store its value with the name of the object. However, this feature is not implemented in the main browsers at the time of this writing.

Try It Out Creating a Contact Form

In this example, you are going to combine several of the form controls to make up a contact form for our Example Café.

1. Create a new Transitional XHTML 1.0 document, with the skeleton in place. Then add a heading:

```
<?xml version="1.0" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">
<head>
  <title>Contact Us</title>
</head>
<body>

<h1>Contact Us</h1>
<p>Use the following form to send a message to Example Cafe</p>

</body>
</html>
```

2. The form is going to be placed in a table with two columns so that the instructions are in the left column, and the form controls are aligned in the right column. (Without this, the form controls would look uneven across the page.) This is quite a common technique in writing forms.

   In the first two rows you can add a text input for the visitor's e-mail address using an `<input>` element whose `type` attribute has a value of `text`. You can also set the size of the form control and the maximum number of characters a user can enter.

   Add the following table under the paragraph that tells people to use the form to send a message:

```
<table>
      <tr>
        <td>Your email</td>
        <td><input type="text" name="txtFrom" id="emailFrom"
          size="20" maxlength="250" /></td>
      </tr>
```

3. This first row of the table is followed by a second row containing a text area for their message. The size of the text area is specified using the `rows` and `cols` attributes:

```
<tr>
  <td>Message</td>
  <td><textarea name="txtBody" id="emailBody" cols="50"
  rows="10"></textarea></td>
</tr>
```

4. In the next row, you can add a select box so that the user can tell you how they heard of the café:

```
<tr>
  <td>How did you hear of us?</td>
  <td>
    <select name="selReferrer">
      <option value="google">Google</option>
      <option value="ad">Local newspaper ad</option>
      <option value="friend">Friend</option>
      <option value="other">Other</option>
     </select>
  </td>
</tr>
```

5. In the final row, add a checkbox to indicate whether the visitor wants to sign up for e-mail updates. This is created with the `<input>` element, whose `type` attribute has a value of `checkbox`. The checkbox should be selected by default, and this is indicated using the `checked` attribute:

```
  <tr>
    <td>Newsletter</td>
    <td><input type="checkbox" name="chkBody" id="newsletterSignup"
      checked="checked" />
      Ensure this box is checked if you would like to
      receive email updates</td>
  </tr>
</table>
```

6. Finally, you need to add a submit button, again using the `<input>` element so that the visitor can send the message to the café:

```
<input type="submit" value="Send message" />
```

7. Save the file as `emailForm.html` and open it in your browser; it should look something like Figure 5-21.

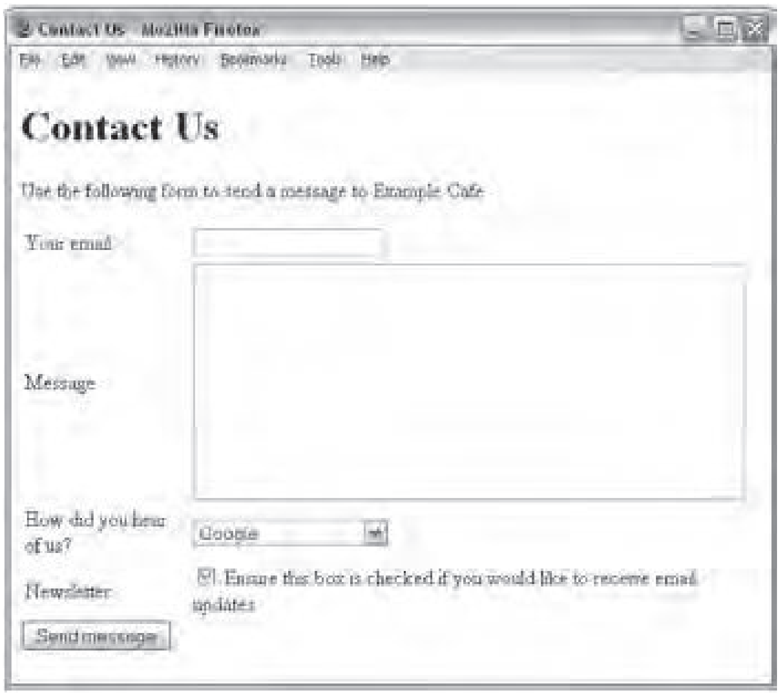

Contact Us - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Contact Us
Use the following form to send a message to Example Cafe
Your email
Message
How did you hear of us?
Google
Newsletter
Ensure this box is checked if you would like to receive email updates
Send message

Figure 5-21

Now that you've seen the basics of forms, it is time to look at more advanced features that you can use to enhance your forms.

Creating Labels for Controls and the <label> Element

Forms can be confusing enough at the best of times. I've received many insurance and tax forms that have left me scratching my head, and I'm sure I'm not the only one.

If you are creating a form for your site, it is worth spending time to provide good labeling so that the user knows what data he or she should be entering where. If visitors have difficulty understanding your form, they will be less likely to complete the form (in particular if they are purchasing something), or they are more likely to make a mistake when filling it in.

Some form controls, such as buttons, already have labels. For the majority of form controls, however, you will have to provide the label yourself.

For controls that do not have a label, you should use the `<label>` element. This element does not affect the form in any way other than telling users what information they should be entering (`ch05_eg16.html`).

```
<form action="http://www.example.org/login.aspx" method="post"
name="frmLogin">
  <table>
    <tr>
      <td><label for="Uname">User name</label></td>
      <td><input type="text" id="Uname" name="txtUserName" /></td>
    </tr>
    <tr>
      <td><label for="Pwd">Password</label></td>
      <td><input type="password" id="Pwd" name="pwdPassword" /></td>
    </tr>
  </table>
</form>
```

You can see that this form has been placed inside a table; this ensures that even if the labels are of different lengths, the text inputs are aligned in their own column. If a list of text inputs is not aligned, it can be harder to use.

As you can see here, the `<label>` element carries an attribute called `for`, which indicates the form control associated with the label. The value of the `for` attribute should be the same as the value of the `id` attribute on the corresponding form control. For example, the textbox form control, where a user enters his or her username, has an `id` attribute whose value is `Uname`, and the label for this textbox has a `for` attribute whose value is also `Uname`.

Figure 5-22 shows you what this login screen looks like.

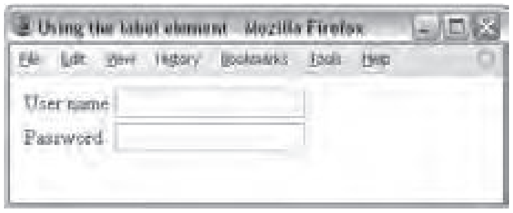

Using the label element - Mozilla Firefox
File Edit View History Bookmarks Tools Help
User name
Password

Figure 5-22

The label may be positioned before or after the control. For textboxes and drop-down select boxes, it is generally good practice to have the label on the left or above the form control, whereas for checkboxes and radio buttons it is often easier to associate the label with the correct form control if they are on the right.

You should have a new `<label>` element for each form control.

Another way to use the `<label>` element is as a containing element. When you use the `<label>` element this way, you do not need to use the `for` attribute because it applies to the form element that is inside it. This kind of label is sometimes known as an *implicit label*. For example:

```
<form action="http://www.example.org/login.aspx" method="post"
name="frmLogin">
  <label>Username <input type="text" id="Uname" name="txtUserName" /></label>
  <label>Password <input type="password" id="Pwd" name="pwdPassword" />
  </label>
</form>
```

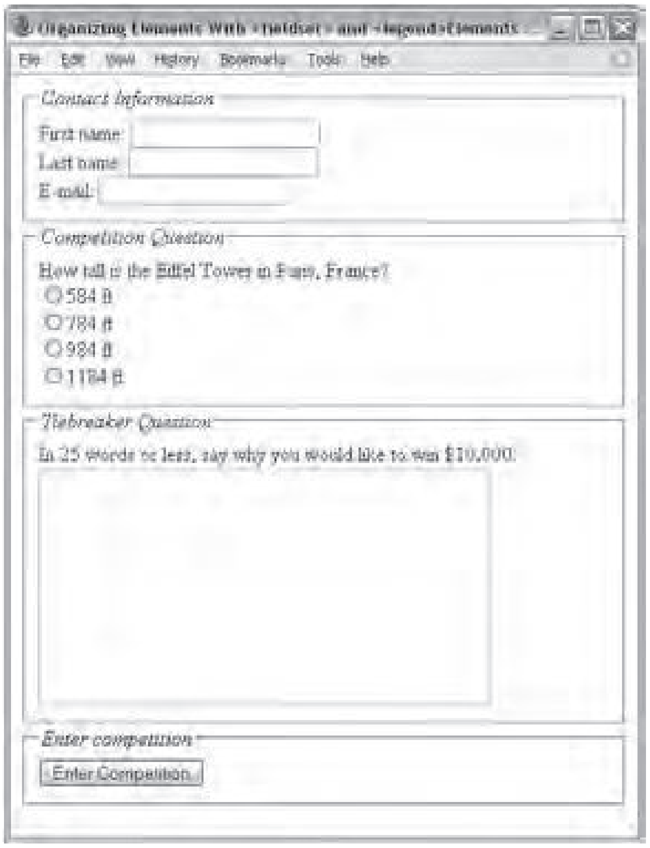
The drawback to this approach is that you cannot control where the label appears in relation to the form control, and you certainly cannot have the label in a different table cell from the form control, as the markup would not nest correctly.

Structuring Your Forms with <fieldset> and <legend> Elements

Large forms can be confusing for users, so it's good practice to group together related form controls. The `<fieldset>` and `<legend>` elements do exactly this — help you group controls.

- The `<fieldset>` element creates a border around the group of form controls to show that they are related.
- The `<legend>` element allows you to specify a caption for the `<fieldset>` element, which acts as a title for the group of form controls. When used, the `<legend>` element should always be the first child of the `<fieldset>` element.

Figure 5-23 shows these elements in action. You can see that the form has been divided into four sections: Contact Information, Competition Question, Tiebreaker Question, and Enter Competition.


Organizing Elements With <fieldset> and <legend> Elements
File Edit View History Bookmarks Tools Help

Contact Information
First name:
Last name:
E-mail:

Competition Question
How tall is the Eiffel Tower in Paris, France?
○ 584 ft
○ 784 ft
○ 984 ft
○ 1184 ft

Tiebreaker Question
In 25 words or less, say why you would like to win $10,000.

Enter competition
Enter Competition

Figure 5-23

Let's take a look at the code for this example. You can see how the `<fieldset>` elements create borders around the groups of form controls, and how the `<legend>` elements are used to title the groups of controls. Remember, when you use the `<legend>` element, it must be the first child of the `<fieldset>` element (`ch05_eg17.html`).

```
<form action="http://www.example.org/competition.aspx" method="post"
name="frmComp">
 <fieldset>
  <legend><em>Contact Information</em></legend>
   <label>First name: <input type="text" name="txtFName" size="20" />
   </label><br />
   <label>Last name: <input type="text" name="txtLName" size="20" /></label>
   <br />
   <label>E-mail: <input type="text" name="txtEmail" size="20" /></label>
   <br />
  </fieldset>
  <fieldset>
  <legend><em>Competition Question</em></legend>
   How tall is the Eiffel Tower in Paris, France? <br />
   <label><input type="radio" name="radAnswer" value="584" />
      584ft</label><br />
   <label><input type="radio" name="radAnswer" value="784" />
      784ft</label><br />
   <label><input type="radio" name="radAnswer" value="984" />
      984ft</label><br />
   <label><input type="radio" name="radAnswer" value="1184" />
       1184ft</label><br />
  </fieldset>
  <fieldset>
    <legend><em>Tiebreaker Question</em></legend>
      <label>In 25 words or less, say why you would like to win $10,000:
        <textarea name="txtTiebreaker" rows="10" cols="40"></textarea>
      </label>
  </fieldset>
  <fieldset>
    <legend><em>Enter competition</em></legend>
       <input type="submit" value="Enter Competition" />
  </fieldset>
</form>
```

The `<fieldset>` element can take the following attributes:

- All the universal attributes
- The basic event attributes

If you use a table to format your form, the entire `<table>` element must appear inside the `<fieldset>` element. If a `<fieldset>` resides within a table that is used to format the page, then the entire fieldset must reside within the same cell.

The `<legend>` element can take the following attributes:

- `accesskey`, which you will learn about in the next section.
- `align`, which you have seen already, and is deprecated — you should use CSS positioning instead.
- All the universal attributes.
- UI event attributes.

Focus

When a web page featuring several links or several form controls loads, you may have noticed that you are able to use your Tab key to move between those elements (or Shift+Tab to move backward through elements). As you move between them, the web browser tends to add some type of border or highlighting to that element (be it a link or a form control). This is known as *focus*.

Only elements that a user can interact with, such as links and form controls, can receive focus. Indeed, if a user is expected to interact with an element, that element *must* be able to receive focus.

An element can gain focus in three ways:

- An element can be selected using a pointing device such as a mouse or trackball.
- Elements that can gain focus can be navigated between using the keyboard — often using the Tab key (or Shift+Tab to move backward through elements). As you are about to see, the elements in some documents can be given a fixed *tabbing order*, indicating the order in which elements gain focus when the user pressed the tab key.
- A web-page author can indicate that an element should receive focus when a user presses a keyboard shortcut known as an *access key*. For example, if the page author set the access key on a search box to be the key for the letter *s*, on a PC you would likely press the Alt key plus the access key (Alt+S), whereas on a Mac you would press the Control key with an access key (Control+S), and the corresponding form control would gain focus.

Tabbing Order

If you want to control the order in which elements can gain focus, you can use the `tabindex` attribute to give that element a number between 0 and 32767, which becomes part of the tabbing order. Every time the user presses the Tab key, the focus moves to the element with the next highest tabbing order (and again, Shift+Tab moves focus in reverse order).

The following elements can carry a `tabindex` attribute:

```
<a> <area> <button> <input> <object> <select> <textarea>
```

After a user has tabbed through all elements in a document that can gain focus, then focus may be given to other browser features (most commonly the address bar).

To demonstrate how tabbing order works, the following example gives focus to the checkboxes in a different order than you might expect (`ch05_eg18.html`):

```
<form action="http://www.example.com/tabbing.aspx" method="get"
  name="frmTabExample">
  <input type="checkbox" name="chkNumber" value="1" tabindex="3" /> One<br/>
  <input type="checkbox" name="chkNumber" value="2" tabindex="7" /> Two<br/>
  <input type="checkbox" name="chkNumber" value="3" tabindex="4" /> Three<br />
  <input type="checkbox" name="chkNumber" value="4" tabindex="1" /> Four<br/>
  <input type="checkbox" name="chkNumber" value="5" tabindex="9" /> Five<br/>
  <input type="checkbox" name="chkNumber" value="6" tabindex="6" /> Six<br/>
  <input type="checkbox" name="chkNumber" value="7" tabindex="10" />Seven<br />
  <input type="checkbox" name="chkNumber" value="8" tabindex="2" />Eight<br />
  <input type="checkbox" name="chkNumber" value="9" tabindex="8" /> Nine<br/>
  <input type="checkbox" name="chkNumber" value="10" tabindex="5" /> Ten<br/>
<input type="submit" value="Submit" />
</form>
```

In this example, the checkboxes receive focus in the following order:

```
4,  8,  1,  3,  10,  6,  2,  9,  5,  7
```

Figure 5-24 shows how Firefox 2 for PC will, by default, give a yellow outline to form elements as they gain focus (other browsers give different outlines — Internet Explorer uses blue lines). I have zoomed in on the item in focus so you can see it in closer detail.

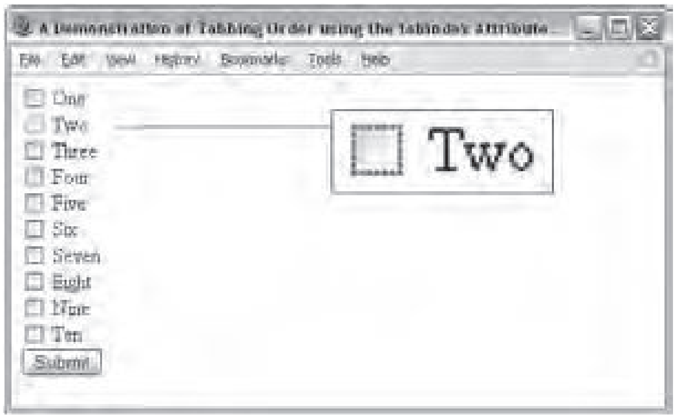

A Demonstration of Tabbing Order using the tabindex attribute
File Edit View History Bookmarks Tools Help
One
Two
Three
Four
Five
Six
Seven
Eight
Nine
Ten
Submit
Two

Figure 5-24

Elements that could gain focus but do not have a `tabindex` attribute are automatically given a value of `0`; therefore, when you specify a `tabindex` value, it should be `1` or higher, rather than `0`.

If two elements have the same value for a `tabindex` attribute, they will be navigated in the order in which they appear in the document. So, once the elements that have a `tabindex` of `1` or more have been cycled through, the browser will cycle through the remaining elements (which have a value of `0`) in the order in which they appear in the page.

Note that if an element is disabled, it cannot gain focus and does not participate in the tabbing order. Also, if you use a Mac you may need to check your Keyboard Shortcuts settings in system preferences; at the bottom of the window where it says "Full Keyboard Access," this needs to have the "All Controls" option selected.

Access Keys

Access keys act just like keyboard shortcuts. The access key is a single character from the document's character set that is expected to appear on the user's keyboard. When this key is used in conjunction with another key or keys (such as Alt with IE on Windows, Alt and Shift with Firefox on Windows, and Control on an Apple), the browser automatically goes to that section. (Exactly which key must be used in conjunction with the access key depends upon the operating system and browser.)

The access key is defined using the `accesskey` attribute. The value of this attribute is the character (and key on the keyboard) you want the user to be able to press (in conjunction with the other key/keys that are dependent upon the operating system and browser).

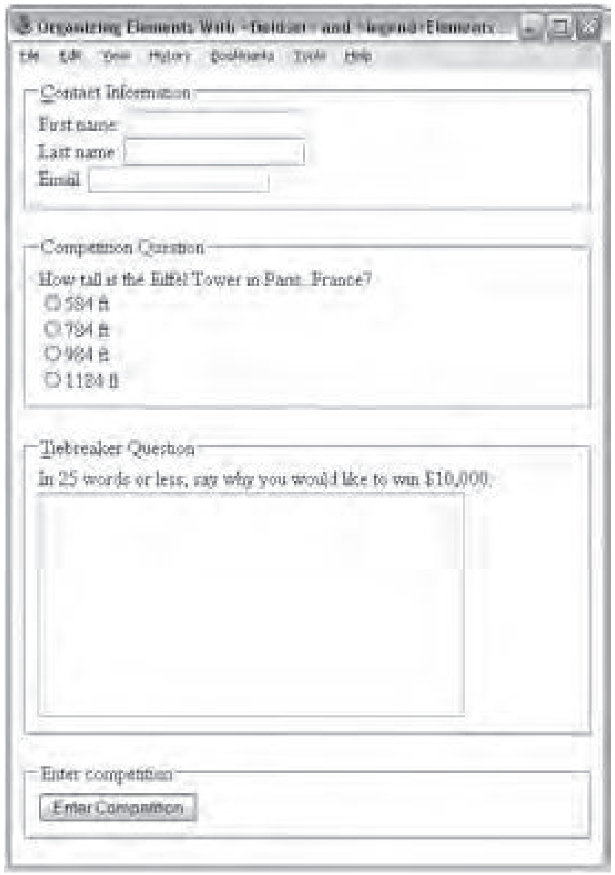
The following elements can carry an access key attribute:

```
<a> <area> <button> <input> <label> <legend> <textarea>
```

To see how access keys work, you can revisit the example of a competition form (`ch05_eg17.html`), which was covered in the section "Structuring Your Forms with <fieldset> and <legend> Elements" earlier in this chapter. Now the `accesskey` attributes can be added to the `<legend>` elements:

```
<legend accesskey="c"><u>C</u>ontact Information</legend>
<legend>Competition Question</legend>
<legend accesskey="t"><u>T</u>iebreaker Question</legend>
<legend>Enter competition</legend>
```

The new version of this file is `ch05_eg19.html` in the download code. (Extra `<br />` elements have been added to show how the screen scrolls to the appropriate section when an access key is used.) As a hint to users that they can use the access keys as shortcuts, information has also been added to the information in the `<legend>` element by underlining the access key. Figure 5-25 shows how this updated example looks in a browser.


Organizing Elements With <fieldset> and <legend> Elements...
File Edit View History Bookmarks Tools Help
Enter Competition

Contact Information

First name

Last name

Email

Competition Question

How tall is the Eiffel Tower in Paris, France?

584 ft

784 ft

984 ft

1184 ft

Tiebreaker Question

In 25 words or less, say why you would like to win $10,000.

Enter competition

Figure 5-25

The effect of an access key being used depends upon the element that it is used with. With `<legend>` elements, such as those shown previously, the browser scrolls to that part of the page automatically and gives focus to the first form control in the section. When used with form controls, those elements gain focus. As soon as the element gains focus, the user should be able to interact with it (either by typing in text controls or pressing the Enter or Return key with other form controls).

When using letters a–z, it does not matter whether you specify an uppercase or lowercase access key, although, strictly speaking, it should be lowercase.

Disabled and Read-Only Controls

Throughout the chapter, you have seen that several of the elements can carry attributes called `disabled` and `readonly`:

- The `readonly` attribute prevents users from changing the value of the form controls themselves, although it may be modified by a script. The name and value of any `readonly` control *will* be sent to the server. The value of this attribute should be `readonly`.
- The `disabled` attribute disables the form control so that users cannot alter it. A script can be used to re-enable the control, but unless a control is re-enabled, the name and value will not be sent to the server. The value of this attribute should be `disabled`.

A `readonly` control is particularly helpful when you want to stop visitors from changing a part of the form, perhaps because it must not change (for example, if you put terms and conditions inside a text area).

The `disabled` attribute is particularly helpful when preventing users from interacting with a control until they have done something else. For example, you might use a script to disable a submit button until all of the form fields contain a value.

The following table indicates which form controls work with the `readonly` and `disabled` attributes.

Element	**readonly**	**disabled**
`<textarea>`	Yes	Yes
`<input type="text" />`	Yes	Yes
`<input type="checkbox" />`	No	Yes
`<input type="radio" />`	No	Yes
`<input type="submit" />`	No	Yes
`<input type="reset" />`	No	Yes
`<input type="button" />`	No	Yes
`<select>`	No	Yes
`<option>`	No	Yes
`<button>`	No	Yes

The following table indicates the main differences between the `readonly` and `disabled` attributes.

Attribute	**readonly**	**disabled**
Can be modified	Yes by script, not by user	Not while disabled
Will be sent to server	Yes	Not while disabled
Will receive focus	Yes	No
Included in tabbing order	Yes	No

Sending Form Data to the Server

When your browser requests a web page and when the server sends a page back to the browser, you use the Hypertext Transfer Protocol (HTTP).

There are two methods that a browser can use to send form data to the server — HTTP `get` and HTTP `post` — and you specify which should be used by adding the `method` attribute on the `<form>` element.

If the `<form>` element does not carry a `method` attribute, then by default the `get` method will be used. If you are using a file upload form control, you must choose the `post` method (and you must set the `enctype` attribute to have a value of `multipart/form-data`). Let's take a closer look at each of these methods.

HTTP get

When you send form data to the server using the HTTP `get` method, the form data is appended to the URL that is specified in the `action` attribute of the `<form>` element.

The form data is separated from the URL using a question mark. Following the question mark, you get the name/value pairs for each form control. Each name/value pair is separated by an ampersand (`&`).

For example, take the following login form, which you saw when the password form control was introduced:

```
<form action="http://www.example.com/login.aspx" method="get">
 Username:
 <input type="text" name="txtUsername" value="" size="20" maxlength="20"><br />
 Password:
 <input type="password" name="pwdPassword" value="" size="20" maxlength="20">
 <input type="submit" />
</form>
```

When you click the submit button, your username and password are appended to the URL `http://www.example.com/login.aspx` in what is known as the *query string*. It should look like this:

```
http://www.example.com/login.aspx?txtUsername=Bob&pwdPassword=LetMeIn
```

Note that when a browser requests a URL with any spaces or unsafe characters such as /, \ , =, &, and + (which have special meanings in URLs), they are replaced with a hex code to represent that character. This is done automatically by the browser, and is known as *URL encoding*. When the data reaches the server, the server will usually un-encode the special characters automatically.

One of the advantages of passing form data in a URL is that it can be bookmarked. If you look at searches performed on major search engines such as Google, they tend to use the `get` method so that the page can be bookmarked.

The `get` method, however, has some disadvantages. Indeed, when sending sensitive data such as the password shown here, or credit card details, you should not use the `get` method because the sensitive data becomes part of the URL and is in full view to everyone (and could be bookmarked).

You should not use the HTTP `get` method when:

- ❑ You are dealing with sensitive information, such as passwords or credit card details (because the sensitive form data would be visible as part of a URL).
- ❑ You are updating a data source such as a database or spreadsheet (because someone could make up URLs that would alter your data source).
- ❑ Your form contains a file upload control (because uploaded files cannot be passed in the URL).
- ❑ Your users might enter non-ASCII characters such as Hebrew or Cyrillic characters.

In these circumstances, you should use the HTTP `post` method.

HTTP post

When you send data from a form to the server using the HTTP `post` method, the form data is sent transparently in what is known as the *HTTP headers*. While you do not see these headers, they are not, strictly speaking, secure on their own. If you are sending sensitive information such as credit card details, the data should be sent under a *Secure Sockets Layer*, or *SSL*, and they should be in encrypted.

If the login form you just saw was sent using the `post` method, it could be represented like this in the HTTP headers:

```
User-agent: MSIE 7
Content-Type: application/x-www-form-urlencoded
Content-length: 35
...other headers go here...
txtUserName=Bob&pwdPassword=LetMeIn
```

Note that the last line is the form data, and that it is in exactly the same format as the data after the question mark in the `get` method — it would also be URL-encoded so any spaces or unsafe characters such as `/`, `\` , =, &, and + (which have special meanings in URLs) are replaced with a hex code to represent that character as they were in HTTP `get` requests.

There is nothing to stop you from using the `post` method to send form data to a page that also contains a query string. For example, you might have one page to handle users who want to subscribe to or unsubscribe from a newsletter, and you might choose to indicate whether a user wanted to subscribe or unsubscribe in the query string. Meanwhile, you might want to send their actual contact details in a form that uses the `post` method because you are updating a data source. In this case, you could use the following `<form>` element:

```
<form action="http://www.example.com/newsletter.asp?action=subscribe"
      method="post">
```

The only issue with using the HTTP `post` method is that the information the user entered on the form cannot be bookmarked in the same way it can when it is contained in the URL. So you cannot use it to retrieve a page that was generated using specific form data as you can when you bookmark a page generated by most search engines, but it is good for security reasons.

Try It Out Contact Form Revisited

It is time to revisit the contact form from the earlier Try It Out section in this chapter. This time, you will use techniques learned in the later part of the chapter to add a new field and to make it more usable.

1. Open the file `emailForm.html` that you made earlier in the chapter and save it as `emailForm2.html` so that you have a different copy to work with.

2. You should place `<label>` elements around the instructions that described the purpose of the form control. This `<label>` element should carry the `for` attribute, whose value is the value of the `id` attribute on the corresponding form control, like this one:

```
<tr>
  <td><label for="emailFrom">Your email</label></td>
  <td><input type="text" name="txtFrom" id="emailFrom"
    size="20" tabindex="1" maxlength="250" /></td>
</tr>
```

3. Add in a new single-line text input to the beginning of the form, indicating to whom the message is being sent. This input should be read-only:

```
<tr>
  <td><label for="emailTo">To</label></td>
  <td><input type="text" name="txtTo" readonly="readonly"
    id="emailTo" size="20" value="Example Cafe" /></td>
</tr>
```

4. Set the tab index so that the input that allows visitors to enter their e-mail addresses receives focus first, followed by the text area where the visitors enter their messages:

```
<tr>
  <td><label for="emailFrom">Your email</label></td>
  <td><input type="text" name="txtFrom" id="emailFrom" size="20"
    tabindex="1" maxlength="250" /></td>
</tr>

<tr>
  <td><label for="emailBody">Message</label></td>
  <td><textarea name="txtBody" id="emailBody" cols="50" rows="10"
tabindex="2"></textarea></td>
</tr>
```

5. Now it is time to split the form into two sections using the `<fieldset>` element. In order to make sure that the elements nest correctly, each fieldset will need its own table. The first section will indicate that it is for information about the visitor's message.

```
<fieldset>
    <legend>Your message:</legend>
    <table>
      <tr>
        <td><label for="emailTo">To</label></td>
        <td><input type="text" name="txtTo" readonly="readonly" id="emailTo"
          size="20" value="Example Cafe" /></td>
      </tr>

      <tr>
        <td><label for="emailFrom">Your email</label></td>
        <td><input type="text" name="txtFrom" id="emailFrom" size="20"
          tabindex="1" maxlength="250" /></td>
      </tr>

      <tr>
        <td><label for="emailBody">Message</label></td>
        <td><textarea name="txtBody" id="emailBody" cols="50" rows="10"
          tabindex="2"></textarea></td>
      </tr>

    </table>
  </fieldset>
```

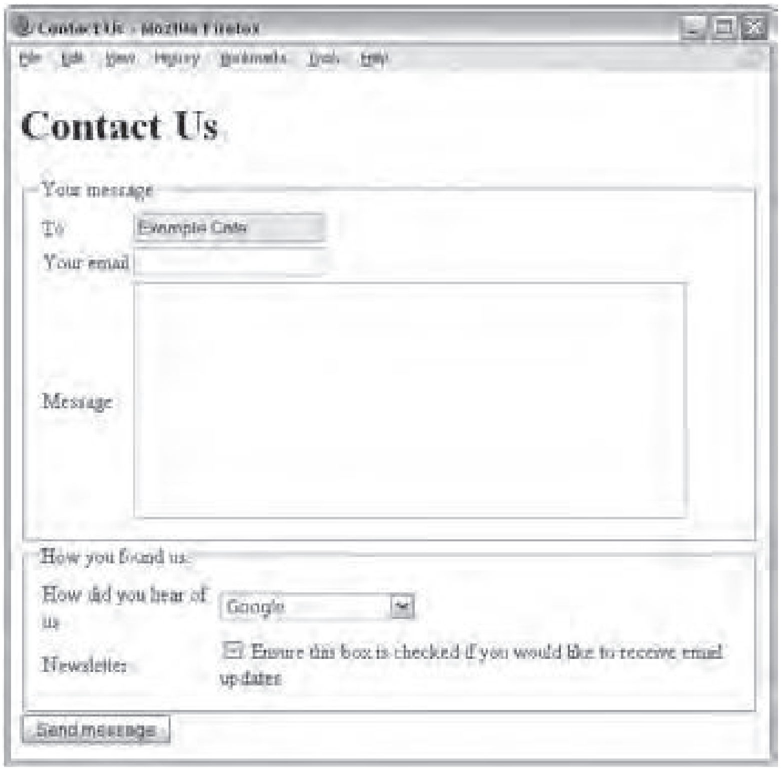
The second section is for information about the company (how the user found the site and if the user wants to be on the mailing list):

```
<fieldset>
    <legend>How you found us:</legend>
    <table>
      <tr>
        <td><label for="emailBody">How did you hear of us</label></td>
        <td>
        <select name="selReferrer">
              <option value="google">Google</option>
           <option value="ad">Local newspaper ad</option>
           <option value="friend">Friend</option>
           <option value="other">Other</option>
        </select>
          </td>
      </tr>

      <tr>
        <td><label for="newsletterSignup">Newsletter</label></td>
        <td><input type="checkbox" name="chkBody" id="newsletterSignup"

       checked="checked" /> Ensure this box is checked if you would like
        to receive email updates</td>
      </tr>
    </table>
  </fieldset>
```

This extended registration form is now a lot more usable. If you save the file again and open it in your browser, you should find something that resembles Figure 5-26.


Contact Us - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Send message

Contact Us

Your message

To	Example Cafe
Your email	
Message	

How you found us

How did you hear of us	Google
Newsletter	✓ Ensure this box is checked if you would like to receive email updates

Figure 5-26

Summary

This chapter has introduced you to the world of creating online forms, which are a vital part of many sites. In most cases, when you want or need to directly collect information from a visitor to your site you will use a form, and you have seen several different examples of forms in this chapter.

You have learned how a form lives inside a `<form>` element and that inside a form there are one or more form controls. You have seen how the `<input>` element can be used to create several kinds of form controls, namely single-line text input controls, checkboxes, radio buttons, file upload boxes, buttons, and hidden form controls. There are also the `<textarea>` elements for creating multiple line text inputs and the `<select>` and `<option>` elements for creating select boxes.

Once you have created a form with its form controls, you need to ensure that each element is labeled properly so that users know what information they should enter or which selection they will be making. You can also organize larger forms using the `<fieldset>` and `<label>` elements and aid navigation with `tabindex` and `accesskey` attributes.

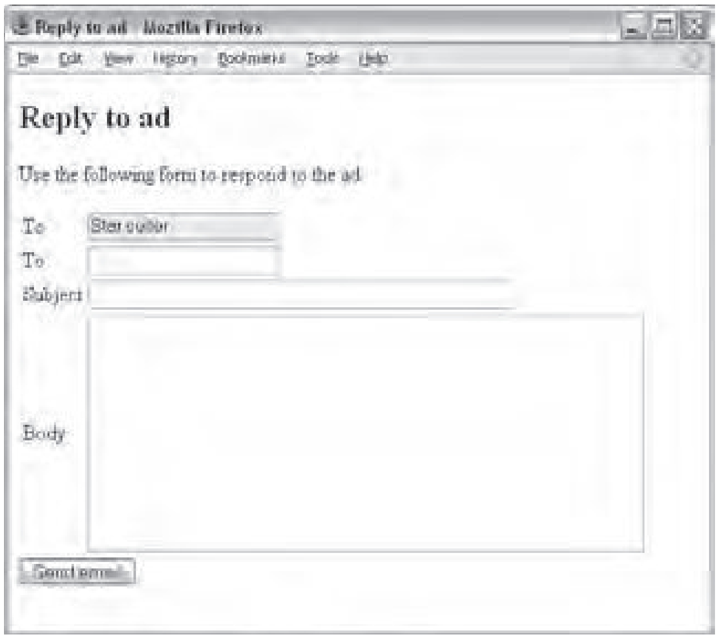
Finally, you learned when you should use the HTTP `get` or `post` methods to send form data to the server.

Next, it is time to look at the last of our core XHTML chapters, which covers framesets. You will see more about form design in Chapter 12, which covers some design issues that will make your forms easier to understand.

Exercises

The answers to all the exercises are in Appendix A.

1. Create an e-mail feedback form that looks like the one shown in Figure 5-27.


Reply to ad - Mozilla Firefox
File Edit View History Bookmarks Tools Help

Reply to ad

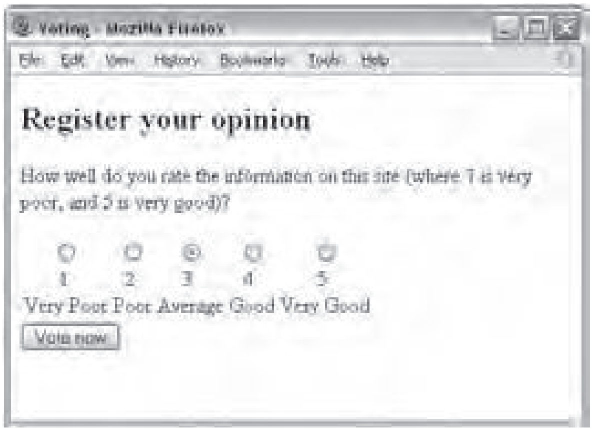
Use the following form to respond to the ad

To: Star cutter
To:
Subject:
Body:

Figure 5-27

Note that the first textbox is a `readonly` textbox so that the user cannot alter the name of the person the mail is being sent to.

2. Create a voting or ranking form that looks like the one shown in Figure 5-28.



Register your opinion

How well do you rate the information on this site (where 1 is very poor, and 5 is very good)?

○	○	◉	○	○
1	2	3	4	5
Very Poor	Poor	Average	Good	Very Good

Figure 5-28

Note that the following `<style>` element was added to the `<head>` of the document to make each column of the table the same fixed width, with text aligned in the center (you'll see more about this in Chapter 7).

```
<head>
  <title>Voting</title>
  <style type="text/css">td {width:100; text-align:center;}</style>
</head>
```