CHAPTER 6

DESIGNING TABLES AND FORMS

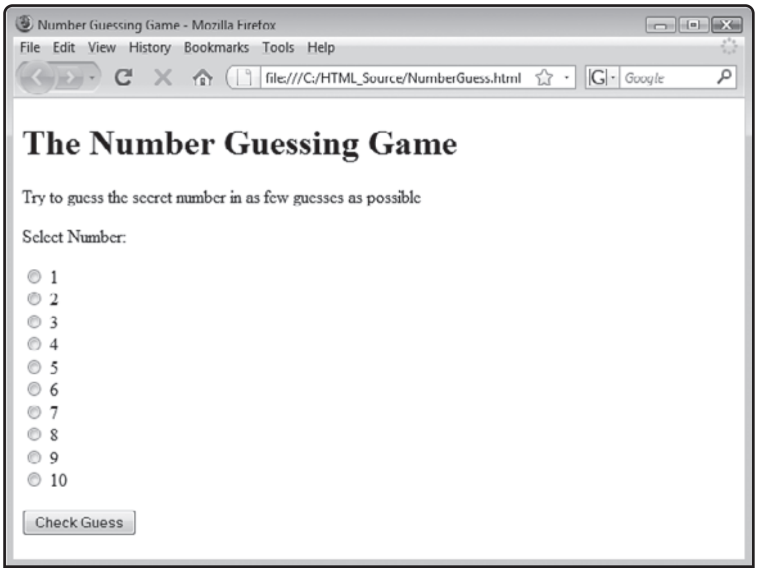
Displaying text is essential to web page development. So far, you have learned how to display text in headings, paragraphs, and lists and to emphasize text structure using an assortment of other elements. Another way of organizing text is using tables. Tables can help you streamline the display of any data that can be organized in rows and columns. Another important aspect of web development is the collection of input from the people who visit your web pages, which is essential to the creation of interactive websites. If you have already used search engines or purchased anything online, you already have plenty of experience working with forms. This chapter will expand your web development skills by teaching you how to display information in tables and collect it using forms.

Specifically, you will learn how to:

- Organize and display data using forms
- Create forms with borders, row, and column headers, and merged cells
- Collect user input using forms
- Work with different types of text fields, buttons, and drop-down lists
- Improve form organization using labels and fieldsets

Project Preview: The Number Guessing Game

This chapter's web project is the Number Guessing Game. This game challenges the player to try to guess a randomly generated number in the range of 1 to 10 in as few guesses as possible. The player makes guesses by clicking on one of ten radio controls that make up the application's form, as demonstrated in Figure 6.1.


Number Guessing Game - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/HTML_Source/NumberGuess.html
Google

The Number Guessing Game

Try to guess the secret number in as few guesses as possible

Select Number:

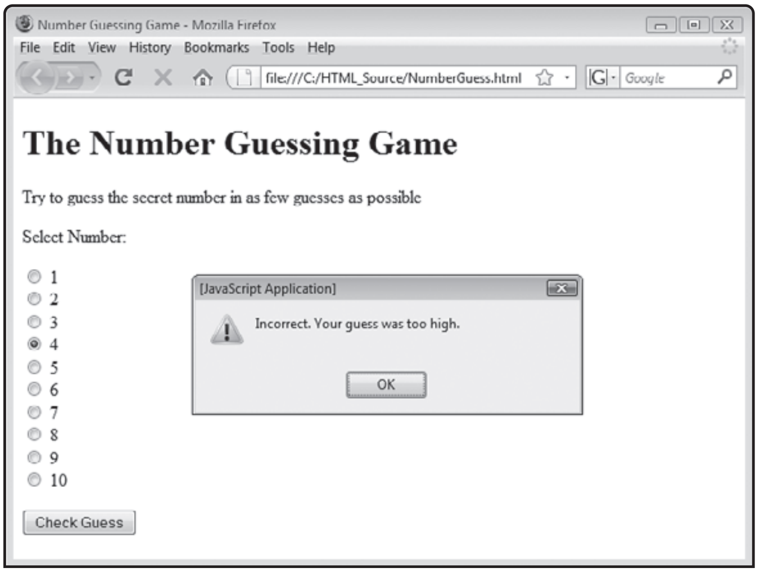
1
2
3
4
5
6
7
8
9
10

Figure 6.1

The game is made up of a form that consists of 10 radio controls and a button control.

Once the player has selected the radio control representing his guess, the button control labeled `Check Guess` must be clicked. The game will then compare the player's input against the game's secret number in order to determine whether the player has won the game or needs to keep on playing. Figure 6.2 shows an example of the message that the game displays in the event the player's guess was higher than the secret number.

Game play continues until the player finally guesses the secret number. Once the correct number has been selected, the message shown in Figure 6.3 is displayed, informing the player that he has won the game.


Number Guessing Game - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/HTML_Source/NumberGuess.html
Google

The Number Guessing Game

Try to guess the secret number in as few guesses as possible

Select Number:

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10

Check Guess

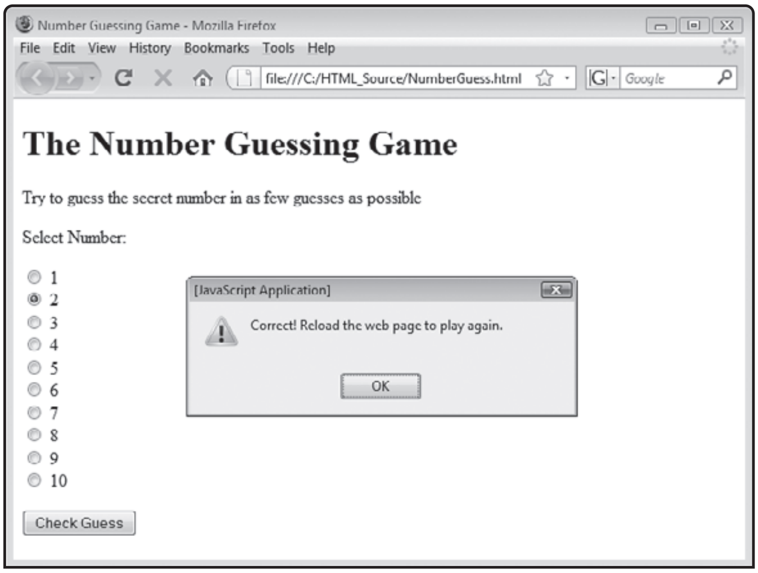
[JavaScript Application]

Incorrect. Your guess was too high.

OK

FIGURE 6.2

The player can use the messages provided by the game to help guide his next move.


Number Guessing Game - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/HTML_Source/NumberGuess.html
Google

The Number Guessing Game

Try to guess the secret number in as few guesses as possible

Select Number:

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10

Check Guess

[JavaScript Application]

Correct! Reload the web page to play again.

OK

FIGURE 6.3

The message congratulates the player and explains how to restart the game.

USING TABLES TO DISPLAY INFORMATION

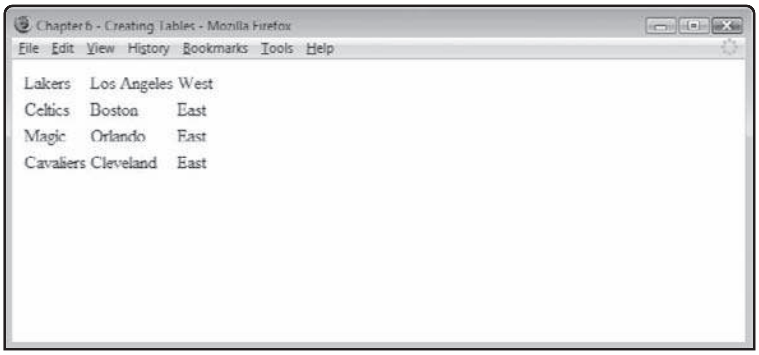
Tables provide a convenient way of displaying collections of related data that lend themselves to a tabular format. To create tables, you need to learn how to work with three elements: `table`, `tr`, and `td`. The `table` element is made up of the `<table>` tag, which marks the beginning of a table, and the `</table>` tag, which marks the end of a table. Inside the `table` element, rows are established using the `<tr>` and `</tr>` tags and individual cells within those rows are created using the `<td>` and `</td>` tags (the letters "td" stand for table data, which can be text, lists, forms, images, etc.).

Basic Table Elements

The following example demonstrates how to use the `table`, `tr`, and `td` elements to construct a simple table made up of four rows with three cells per row. As you can see, `<tr>` and `</tr>` tags are used to define each row in the table and `<td>` and `</td>` tags are used to enclose the content of every cell in each row.

```
<table>
  <tr>
    <td>Lakers</td>
    <td>Los Angeles</td>
    <td>West</td>
  </tr>
  <tr>
    <td>Celtics</td>
    <td>Boston</td>
    <td>East</td>
  </tr>
  <tr>
    <td>Magic</td>
    <td>Orlando</td>
    <td>East</td>
  </tr>
  <tr>
    <td>Cavaliers</td>
    <td>Cleveland</td>
    <td>East</td>
  </tr>
</table>
```

As Figure 6.4 demonstrates, the browser automatically sizes the table and its rows and cells based on the contents contained in the table. However, as you will learn in Chapter 8, you can use CSS to control table formatting.


Chapter6 - Creating Tables - Mozilla Firefox
File Edit View History Bookmarks Tools Help

Lakers	Los Angeles	West
Celtics	Boston	East
Magic	Orlando	East
Cavaliers	Cleveland	East

FIGURE 6.4

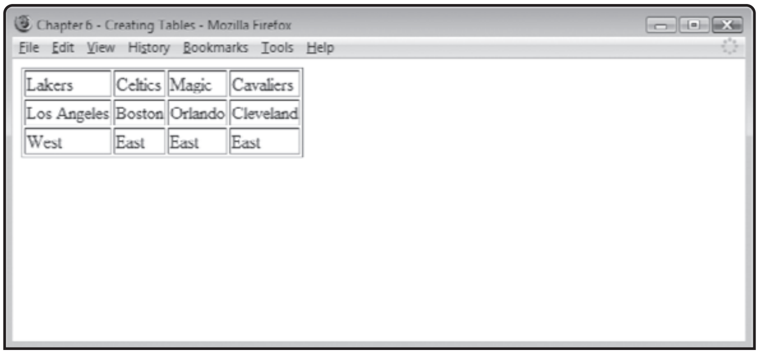
An example of a table made up of three rows and four columns.

Adding Borders to Your Tables

In order to make tables easier for people to view and understand, it is often helpful to add a border to them, showing the outer edges of the table while also distinguishing rows and cells. To accomplish this feat, all you have to do is to add the `border` attribute to the `form` element, as demonstrated here:

```
<table border = "1">
  <tr>
    <td>Lakers</td>
    <td>Celtics</td>
    <td>Magic</td>
    <td>Cavaliers</td>
  </tr>
  <tr>
    <td>Los Angeles</td>
    <td>Boston</td>
    <td>Orlando</td>
    <td>Cleveland</td>
  </tr>
  <tr>
    <td>West</td>
    <td>East</td>
    <td>East</td>
    <td>East</td>
  </tr>
</table>
```

As you can see, the `table` element's `border` attribute has been assigned a value of 1. This instructs the browser to draw a 1-pixel thick border around the table and its rows and cells. When rendered, this example displays the table shown in Figure 6.5.

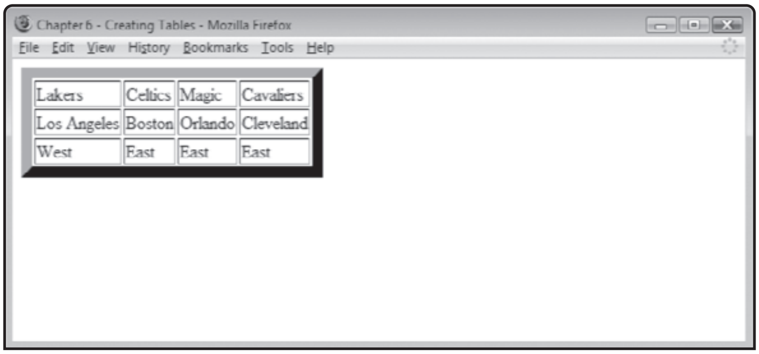

Chapter 6 - Creating Tables - Mozilla Firefox
File Edit View History Bookmarks Tools Help

Lakers	Celtics	Magic	Cavaliers
Los Angeles	Boston	Orlando	Cleveland
West	East	East	East

FIGURE 6.5

An example of a table with a border set to a thickness of 1.

Figure 6.6 shows an example of how the table would look if you changed the value assigned to the `border` attribute from 1 to 10.


Chapter 6 - Creating Tables - Mozilla Firefox
File Edit View History Bookmarks Tools Help

Lakers	Celtics	Magic	Cavaliers
Los Angeles	Boston	Orlando	Cleveland
West	East	East	East

FIGURE 6.6

An example of a table with a border set to a thickness of 10 pixels.

Note that in both examples, the browser automatically left aligns the table's data.

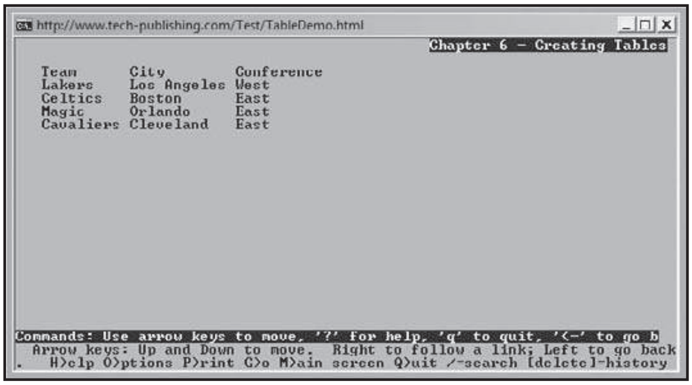
Playing Nice with Non-Graphic Browsers

Unfortunately, non-visual browsers can have trouble rendering tables. To help ensure that these browsers display table data in a useful fashion, use the `summary` and `scope` elements when defining (X)HTML tables. The `summary` element provides a text string that can be used by browsers that reply on text-to-voice speech synthesizers. This element is ignored by visual browsers. The `scope` element helps non-visual browsers determine how to lay out a table by providing header information that helps identify columns (`col`) and rows (`row`).

The following example demonstrates how to use both the `table` element's `summary` attribute and the `td` element's `scope` attribute.

```
<table border = "1" summary = "Popular NBA Basketball Teams">
  <tr>
    <td scope = "col">Team</td>
    <td scope = "col">City</td>
    <td scope = "col">Conference</td>
  </tr>
  <tr>
    <td scope = "row">Lakers</td>
    <td>Los Angeles</td>
    <td>West</td>
  </tr>
  <tr>
    <td scope = "row">Celtics</td>
    <td>Boston</td>
    <td>East</td>
  </tr>
  <tr>
    <td scope = "row">Magic</td>
    <td>Orlando</td>
    <td>East</td>
  </tr>
  <tr>
    <td scope = "row">Cavaliers</td>
    <td>Cleveland</td>
    <td>East</td>
  </tr>
</table>
```

Figure 6.7 demonstrates how this example looks when viewed using the Lynx text-only browser.



```
http://www.tech-publishing.com/Test/TableDemo.html
                                                              Chapter 6 - Creating Tables

   Team      City        Conference
   Lakers    Los Angeles West
   Celtics   Boston      East
   Magic     Orlando     East
   Cavaliers Cleveland   East

Commands: Use arrow keys to move, '?' for help, 'q' to quit, '<-' to go b
  Arrow keys: Up and Down to move.  Right to follow a link; Left to go back
.   H)elp O)ptions P)rint G)o M)ain screen Q)uit /=search [delete]=history
```

FIGURE 6.7

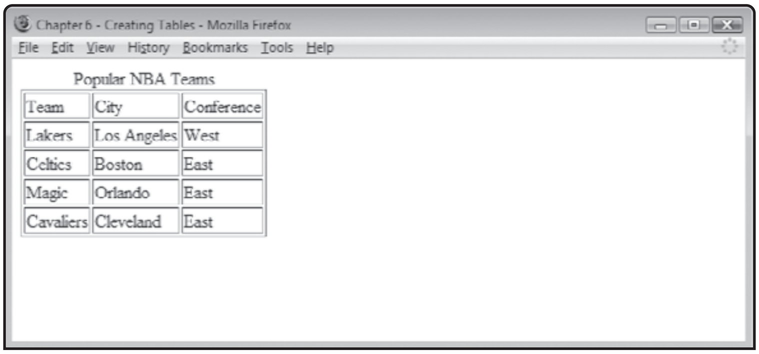
An example of a table displayed using the Lynx browser.

Assigning a Table Heading

You may find it helpful to display a heading for your tables, providing a high-level description of its contents. The way to do this is to use the `caption` element, as demonstrated in the following example.

```
<table border = "1" summary = "Popular NBA Basketball Teams">
  <caption>Popular NBA Teams</caption>
  <tr>
    <td scope = "col">Team</td>
    <td scope = "col">City</td>
    <td scope = "col">Conference</td>
  </tr>
  <tr>
    <td>Lakers</td>
    <td>Los Angeles</td>
    <td>West</td>
  </tr>
  <tr>
    <td>Celtics</td>
    <td>Boston</td>
    <td>East</td>
  </tr>
  <tr>
    <td>Magic</td>
    <td>Orlando</td>
    <td>East</td>
  </tr>
  <tr>
    <td>Cavaliers</td>
    <td>Cleveland</td>
    <td>East</td>
  </tr>
</table>
```

To add a caption to a table, you must embed the caption's text between `<caption>` and `</caption>` tags. You are limited to one `caption` element per table and that element should be placed after the table's opening tag. Figure 6.8 provides an example of how this table will look when loaded by the browser.


Chapter 6 - Creating Tables - Mozilla Firefox
File Edit View History Bookmarks Tools Help

Popular NBA Teams

Team	City	Conference
Lakers	Los Angeles	West
Celtics	Boston	East
Magic	Orlando	East
Cavaliers	Cleveland	East

FIGURE 6.8

An example of a table with a table heading.

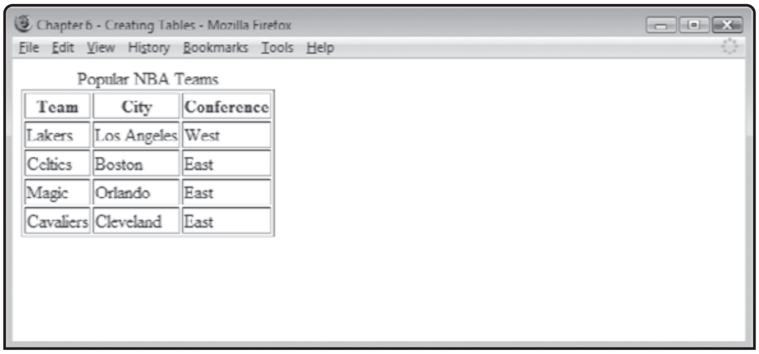
Note that the heading displayed by the `caption` element is always displayed above the table and not within the table itself.

Defining Heading Row and Column Headings

It is often useful to add a heading to a table at the top of each column (`col`) or to the left of each row (`row`). To do so, you use the `th` element, as demonstrated in the following example.

```
<table border = "1" summary = "Popular NBA Basketball Teams">
  <caption>Popular NBA Teams</caption>
  <tr>
    <th scope = "col">Team</th>
    <th scope = "col">City</th>
    <th scope = "col">Conference</th>
  </tr>
  <tr>
    <td>Lakers</td>
    <td>Los Angeles</td>
    <td>West</td>
  </tr>
  <tr>
    <td>Celtics</td>
    <td>Boston</td>
    <td>East</td>
  </tr>
  <tr>
    <td>Magic</td>
    <td>Orlando</td>
    <td>East</td>
  </tr>
  <tr>
    <td>Cavaliers</td>
    <td>Cleveland</td>
    <td>East</td>
  </tr>
</table>
```

Figure 6.9 shows how this table will look once loaded by a browser.


Chapter 6 - Creating Tables - Mozilla Firefox
File Edit View History Bookmarks Tools Help

Popular NBA Teams

Team	City	Conference
Lakers	Los Angeles	West
Celtics	Boston	East
Magic	Orlando	East
Cavaliers	Cleveland	East

FIGURE 6.9

An example of a table with a column heading.

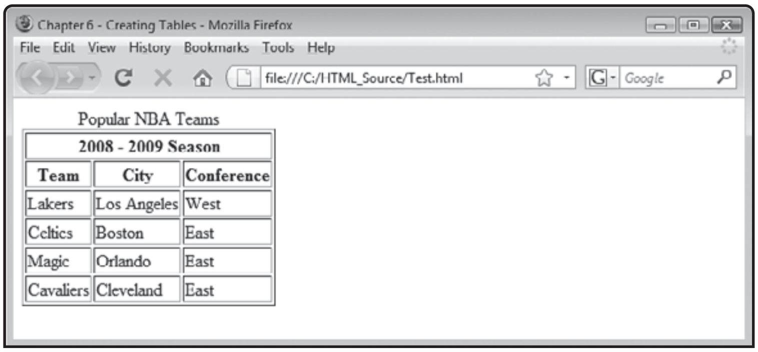
Row and column headings are typically displayed in bold to make them stand out and are usually centered within their cells.

Merging Table Cells

One neat structural trick that you often see in (X)HTML tables is the use of merged cells, in which two or more cells are merged together to create a single large cell. Merged cells can be used to display multi-column or multi-row headings. To merge cells horizontally across a table, you need to add the `colspan` attribute to the appropriate `th` or `td` element. Similarly, to merge cells vertically in a table, you need to add the `rowspan` attribute to the appropriate `th` or `td` element. The following example demonstrates how to use the `colspan` attribute to set up a table header that spans the entire top row of a table.

```
<table border = "1" summary = "Popular NBA Basketball Teams">
  <caption>Popular NBA Teams</caption>
  <tr>
    <th colspan = "3">2008 - 2009 Season</th>
  </tr>
  <tr>
    <th scope = "col">Team</th>
    <th scope = "col">City</th>
    <th scope = "col">Conference</th>
  </tr>
  <tr>
    <td>Lakers</td>
    <td>Los Angeles</td>
    <td>West</td>
  </tr>
  <tr>
    <td>Celtics</td>
    <td>Boston</td>
    <td>East</td>
  </tr>
  <tr>
    <td>Magic</td>
    <td>Orlando</td>
    <td>East</td>
  </tr>
  <tr>
    <td>Cavaliers</td>
    <td>Cleveland</td>
    <td>East</td>
  </tr>
</table>
```

Note that in this example, `colspan` has been assigned a value of 3, instructing the browser to span the header across all three cells in the table. Figure 6.10 shows how the table looks when the (X)HTML page that contains it is loaded into the browser.


Chapter 6 - Creating Tables - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/HTML_Source/Test.html
Google

Popular NBA Teams

2008 - 2009 Season		
Team	**City**	**Conference**
Lakers	Los Angeles	West
Celtics	Boston	East
Magic	Orlando	East
Cavaliers	Cleveland	East

FIGURE 6.10

An example of how to merge table cells.

COLLECTING USER INPUT THROUGH FORMS

Communication on the Internet is a two way street, facilitated by things like email and instant messaging. One way that many websites interact with their visitors is through forms. A *form* is a location on a web page where visitors can provide input. Forms are made up of controls like text fields, check boxes, radio buttons, drop-down lists, and push buttons. The data collected by a control is that control's value. To work with a control, your visitors must select it, placing *focus* on the control. To submit form data, most forms require that visitors click on a Submit button, at which time the form's data is packed up and sent to a form handler.

Forms are defined using the `form` element, which is made up of an opening `<form>` tag and a closing `</form>` tag. Once filled in, forms are processed by a form handler. A *form handler* is a program or script that usually executes on a web server, although a form's handler can instead be a JavaScript located on the same document as the form. Its purpose is to validate and process the contents of forms. This may mean storing the user's input in a server database or file or using it to generate a highly customized web page.

HINT

When submitted, form data is sent to a form handler as name/value pairs. Therefore, one of the requirements of every form is that all form elements be named.

Forms have one required attribute. This attribute is `action`. It is used to specify the URL of the form handler. Another optional `form` element attribute that is usually used is the `method` attribute. This attribute specifies how to send data to a web server for processing by the form handler. The `method` attribute supports the following values.

- **get.** Sends form contents to the form handler by adding it to the URL string.
- **post.** Sends form contents to the form handler using the Hypertext Transfer Protocol.

TRICK

In addition to sending form data using the form element attribute's `get` and `post` options, you can also send form data via email. When you use this option, the form data is sent as a text file made up of name/value pairs. Though not practical for large amounts of data, this option can be used effectively in situations where small amounts of data are transmitted. To use this option, you must formulate the `form` element as demonstrated next. Note the use of the required `mailto:` keyword, which is separated from the desired email address by a colon.

```
<form action = "mailto:xxx@website.com" action = " post">
```

Including a statement like this one in your web documents opens them up to spammers who search the internet looking for email accounts to add to their email lists, so use this option sparingly.

If the `method` attribute is omitted, a default of `get` is assumed. The following example shows an XHTML document that can be used as a template for creating forms that send form data to a form handler. In this example, the form data is sent to a form handler script named processdata.cgi, which resides on the same server as the web document.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">

  <head>
    <meta http-equiv="Content-type" content="text/xhtml; charset=UTF-8" />
    <title>Chapter 6 - Building Forms</title>
  </head>

  <body>

    <form action = "/cgi-bin/processdata.cgi" method = "post">
    </form>

  </body>

</html>
```

As you can see, the `form` element is block level. In addition, it can only contain other block-level elements. If inline elements are to be used, they must be placed within block-level elements inside the form.

The `form` element provides a structure within which specialized form elements are stored. In addition to specialized form elements, forms can also contain regular (X)HTML elements, like paragraphs and headings, or any other (X)HTML element that adds to the forms structure.

Defining Controls Using the input Element

One of the most versatile and commonly used form elements is the `input` element. It can be used to generate a wide variety of different types of form controls. The `input` element is an inline element, so to use it you must enclose it within a block element as you will see demonstrated repeatedly in the many form examples that follow. The range of controls that you can generate using the `input` element is listed here:

- Text fields
- Password fields
- File controls
- Checkbox controls
- Radio buttons
- Button controls
- Image control
- Hidden control
- Submit button
- Reset button

To tell the browser what type of control you want when working with the `input` element, you must specify the control's type using the `input` element's `type` attribute. The browser will then generate the corresponding type of control when the form is rendered.

Creating Text Controls

If you add an `input` element to a form and set its `type` attribute's value to `text` (`input type = "text"`), a single-line text field control is added to your form. By default, most browsers will display the text field as a rectangular text control with a text background and an inset border. Most browsers will automatically set the text field's length to approximately 25 characters wide.

Text fields are perfect for collecting small amounts of text input from your visitors. If you need to collect more than a few words, you should use the `textarea` control, discussed later in this chapter. The following example demonstrates how to create a table containing a text control.

```
<p>
  <label for = "name">Enter your name:</label>
  <input type = "text" id = "name" name = "name" size = "20"
    maxlength = "20" value = "Enter your name here" />
</p>
```

As you can see, the `input` element that is defined in this example specifies a number of optional attributes that further configure the resulting text control. The attributes are explained here:

- **size.** Sets the width, in terms of number of characters, of the text field.
- **maxlength.** Defines the total number of characters of input (including blank spaces) that the text field can accept.
- **value.** Specifies a text string to be displayed inside the text field as its default input.

Figure 6.11 shows how the previous example looks when loaded into the web browser.

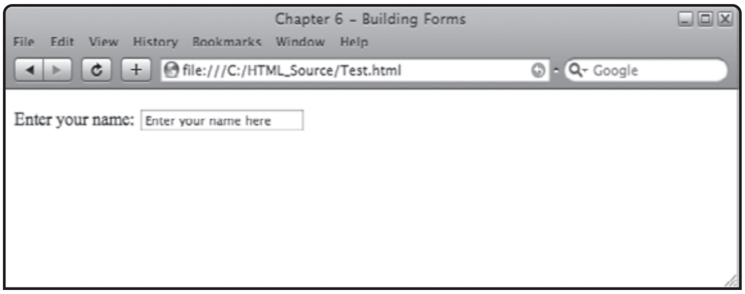

Chapter 6 – Building Forms
File Edit View History Bookmarks Window Help
file:///C:/HTML_Source/Test.html
Google
Enter your name:
Enter your name here

FIGURE 6.11

Using a text control to collect text input from your visitors.

Creating Password Controls

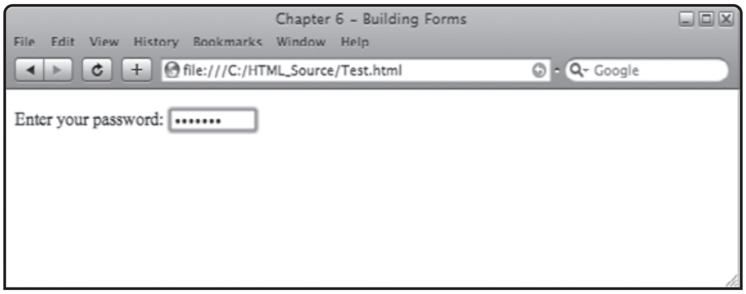
A password control is a text field with a twist—any text typed into it is masked using asterisks or bullet characters to prevent it from being seen by spying eyes. To add a password control to a form, all you have to do is add an `input` element to a form and set its `type` attribute's value to `password` (`input type = "password"`). A password control is a single-line text field. By default, most browsers will display the text field as a rectangular text control with a text background and an inset border.

The following example demonstrates how to add a password control to a form.

```
<p>
  <label for = "password">Enter your password:</label>
  <input type = "password" id = "password" name = "password"
    size = "10" maxlength = "10" />
</p>
```

As you can see, in this example the `input` element specifies a number of optional attributes. These include the `size` and `maxlength` attributes, which work exactly the same here as they do for the `text` control. If you want, you can use the `value` attribute to add a default value for the control, though its usage with this control is certainly questionable.

Figure 6.12 shows how the previous example looks when loaded into the web browser.


Chapter 6 – Building Forms
File Edit View History Bookmarks Window Help
file:///C:/HTML_Source/Test.html
Google

Enter your password: •••••••

FIGURE 6.12

Using a password control to prevent snooping eyes from seeing text input.

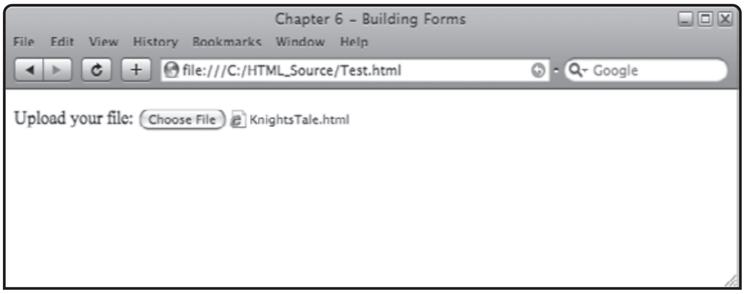
Creating File Controls

By setting the `input` element's `type` attribute to file (`input type= "File"`), you can add a control to your form that allows your visitors to select and upload files from their computer as part of the form's data. This control usually displays as a text field with a corresponding button that when clicked displays a standard file dialog. Visitors can specify the file to upload by selecting it via this dialog or by simply keying in the location of the file directly to the control's text field. You can control the width of the control's text field by setting the `input` element's `size` attribute to a desired length (in characters).

The following example demonstrates how to add a file control to a form.

```
<p>
  <label for = "file">Upload your file:</label>
  <input type = "file" id = "file" name = "file"
    size = "50" accept = "text/css, text/html, text/plain" />
</p>
```

Note the addition of the `accept` attribute, which is used to specify MIME types, restricting the types of files displayed in the standard dialog window to only those specified. Figure 6.13 shows how this example looks when loaded by the Internet Explorer browser.

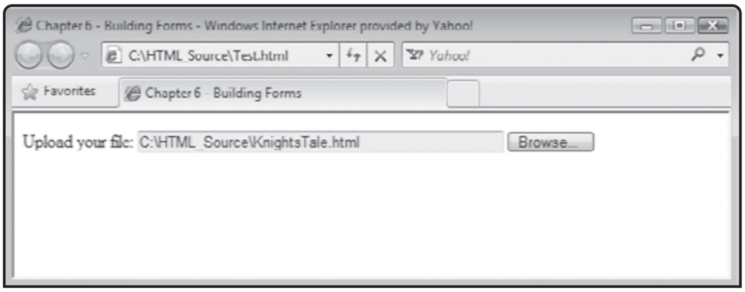

Chapter 6 – Building Forms
File Edit View History Bookmarks Window Help
file:///C:/HTML_Source/Test.html
Google
Choose File

Upload your file: KnightsTale.html

FIGURE 6.13

Using a file control to allow visitors to upload files as form input as seen using Internet Explorer.

Figure 6.13 shows the file control as it will appear on most web browsers. One exception is Apple Safari, which, as demonstrated in Figure 6.14, does not display a text field as part of the control. Instead, Safari displays only the file's name (without its path) and a small icon, identifying the selected file's type.


Chapter 6 - Building Forms - Windows Internet Explorer provided by Yahoo!
C:\HTML_Source\Test.html
Yahoo!
Favorites
Chapter 6 - Building Forms
Browse...

Upload your file: C:\HTML_Source\KnightsTale.html

FIGURE 6.14

Using a file control to allows visitors to upload files as form input as seen using Apple Safari.

Creating Checkbox and Radio Controls

Another way of collecting visitor input is through the use of checkbox and radio controls. A checkbox control is a control that displays a small square, which visitors can click on to select or unselect the control. A selected checkbox control displays a checkmark or similar character inside the control. An unselected checkbox simply appears as an empty box.

Checkbox controls are useful in situations when you want your visitors to select between two choices. You can display a series of checkbox controls to present visitors with individual choices, in which case each checkbox control works independently of the other checkbox controls. Checkbox controls are created by setting the `input` element's `type` attribute to `checkbox` (`type= "checkbox"`).

Checkbox controls are normally displayed within a `label` element, which provides descriptive text that lets visitors know what the checkbox control's choices represent. Usually, you will want to use the `input` element's `value` element to assign a value to your checkbox controls. This value is returned as part of the form's data to the form's handler if the checkbox control has been checked. If you want, you can pre-check a checkbox control by assigning the `input` element's checked attribute a value of `checked` (`checked = "checked"`).

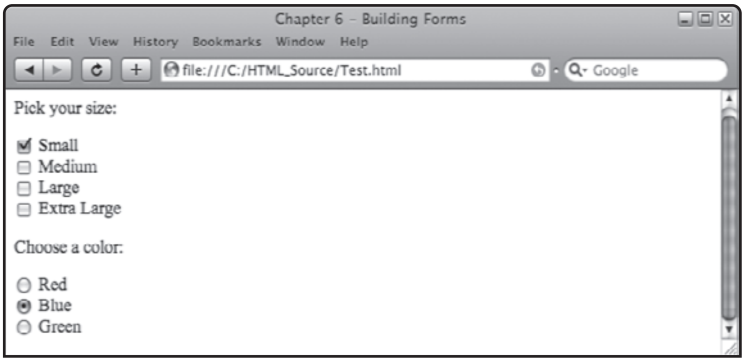
Radio controls are similar to checkbox controls in that they allow visitors to choose options. They differ from checkbox controls in that radio controls can be configured to work in groups (just assign every radio control the same name). When used in groups, radio controls work very much like buttons on stereo radios in that you may have many buttons to choose but only one can be selected (pressed) at a time.

Like checkbox controls you can and should always use the `input` element's `value` attribute to assign a value to a radio control; otherwise, the form handler won't be able to tell which radio button was pressed. You may also use the `checked` attribute to pre-select a given radio control. The following example demonstrates how to work with both the checkbox and radio controls.

```
<p>Pick your size:</p>
<p>
  <input type = "checkbox" id = "checkbox1" name = "checkbox1"
    value = "Small" checked = "checked" />
  <label for = "checkbox1">Small</label><br />
  <input type = "checkbox" id = "checkbox2" name = "checkbox2"
    value = "Medium" />
  <label for = "checkbox2">Medium</label><br />
  <input type = "checkbox" id = "checkbox3" name = "checkbox3"
    value = "Large"/>
  <label for = "checkbox3">Large</label><br />
  <input type = "checkbox" id = "checkbox4" name = "checkbox4"
    value = "Extra Large" />
  <label for = "checkbox4">Extra Large</label>
</p>

<p>Choose a color:</p>
```

```
<p>
  <input type = "radio" id = "radio1" name = "radio"
    value = "Red" />
  <label for = "radio1">Red</label><br />
  <input type = "radio" id = "radio2" name = "radio"
    value = "Blue" checked = "checked" />
  <label for = "radio2">Blue</label><br />
  <input type = "radio" id = "radio3" name = "radio"
    value = "Green" />
  <label for = "radio3">Green</label><br />
</p>
```

Figure 6.15 shows how this example will look when rendered by the browser.


Chapter 6 – Building Forms
File Edit View History Bookmarks Window Help
file:///C:/HTML_Source/Test.html
Google

Pick your size:

☑ Small
☐ Medium
☐ Large
☐ Extra Large

Choose a color:

○ Red
● Blue
○ Green

FIGURE 6.15

Letting visitors make selections using checkbox and radio button controls.

Creating Button Controls

By assigning an input element `type` attribute a value of `button` (`input type = "button"`) you can add a button control to a form. Button controls do not perform any predetermined functions. Instead, they are generic controls that are usually used to trigger the execution of JavaScript. Button controls can display text, set by assigning a text string to the `input` element's `value` attribute. The following example demonstrates how to add button controls to a form.

```
<p>
  <input type = "button" id = "button1" name = "button1"
    value = "Open" />
  <input type = "button" id = "button2" name = "button2"
    value = "Save" />
  <input type = "button" id = "button3" name = "button3"
```

```
    value = "Print" />
  <input type = "button" id = "button4" name = "button4"
    value = "Help" />
</p>
```

Figure 6.16 shows how the resulting form looks when rendered by a browser.

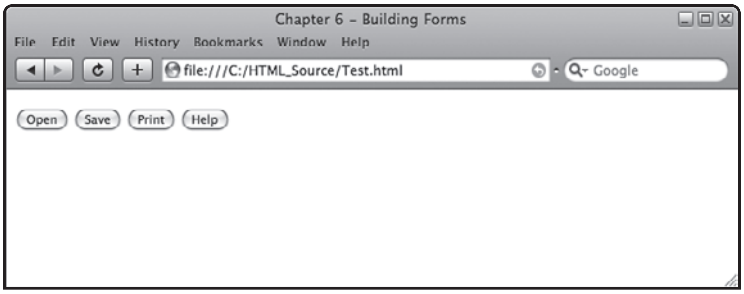

Chapter 6 – Building Forms
File Edit View History Bookmarks Window Help
file:///C:/HTML_Source/Test.html
Google
Open
Save
Print
Help

FIGURE 6.16

Buttons provide a way for your visitors to initiate the execution of specific commands.

Creating Image Controls

In addition to working with generic button controls, you also have the option of using graphics as a substitute for buttons. To set this up all you have to do is assign a `type` of `image` to an `input` element (`input type = "image"`). You can then use the input element's `src` attribute to specify the URL of the graphic to be displayed and the `alt` attribute to specify an alternative test string for non-graphic browsers.

HINT In most cases, you will want to use graphics that are designed to look like buttons when working with this type of control. There are plenty of free websites that you can visit that will help you create graphic buttons for your web pages. (Check out www.buttongenerator.com as an example of one such site.)

The following example demonstrates how to add a series of three graphic button controls to a form.

HINT If you want to re-create this example, you can download copies of the three graphics that are used from this book's companion web page (www.courseptr.com/downloads).

```
<p>
  <input type = "image" id = "image1" name = "image1"
    src = "start.gif" alt = "Click here to start" />
  <input type = "image" id = "image2" name = "image2"
```

```
    src = "quit.gif" alt = "Click here to quit" />
  <input type = "image" id = "image3" name = "image3"
    src = "help.gif" alt = "Click here for help" />
</p>
```

Figure 6.17 shows how the resulting form looks when this example is loaded into a web browser.

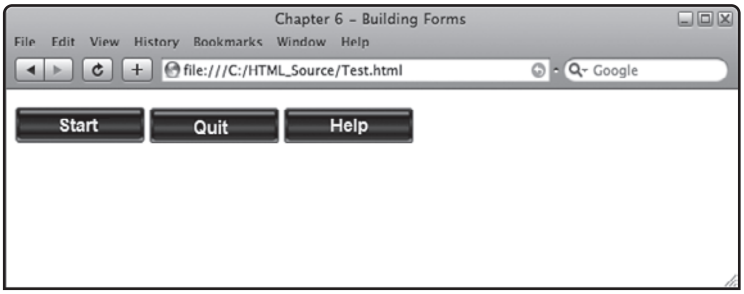

Chapter 6 – Building Forms
File Edit View History Bookmarks Window Help
file:///C:/HTML_Source/Test.html
Google
Start
Quit
Help

FIGURE 6.17
You can use any graphic you want when creating an image control.

Creating a Hidden Control

One type of form control that you never see is the hidden control. As its name implies, this control is kept hidden from the users. Though not often used, web developers sometimes use the control to store a small piece of information, like a calculated value, in the form. The data stored in a hidden control is passed along with the rest of the form's data when submitted to the form handler.

The following example demonstrates how to add a hidden control to a form. The contents of a hidden control can easily be manipulated by JavaScripts after the page loads.

```
<p>
  <input type = "hidden" id = "hidden" name = "hidden"
    value = "12/31/2012" />
</p>
```

Creating Submit and Reset Button Controls

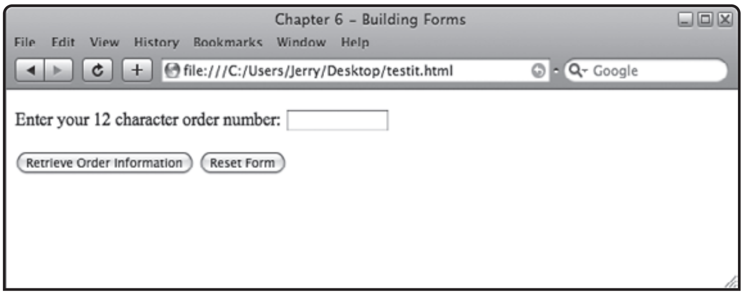
In addition to creating generic button controls, as demonstrated earlier, you can add two specialized button controls to your forms. If you assign a `type` of `submit` (`input type = "submit"`) to an `input` element, you add a Submit button that when clicked, submits form data to the form handler. The form handler is set by adding the `action` attribute to the `form` element. You can assign the text to be displayed on the Submit button by assigning a text string to the `type` attribute's `value` attribute. If the `value` attribute is omitted, a default text string that says Submit Query or perhaps just Submit is displayed.

If you assign a `type` of `reset` (`input type = "reset"`) to an `input` element, you add a `reset` button to your form that when clicked, resets and clears the contents of a form back to its initial values. You can assign the text string displayed on the Reset button using the `input` control's `value` attribute. If omitted, a default string of `Reset` is usually displayed. Few developers use the Reset button anymore because experience has shown that it is easily accidentally clicked, frustrating visitors when the data that they have provided unrepentantly disappears.

The following example demonstrates how to add a Submit and Reset button to a form.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">

 <head>
   <meta http-equiv="Content-type" content="text/xhtml; charset=UTF-8" />
   <title>Chapter 6 - Building Forms</title>
 </head>

 <body>
   <form action = "/cgi-bin/processdata.cgi" method = "post">
     <p>
      <label for = "order_no">Enter your 12 character order number:</label>
      <input type = "text" id = "order_no" name = "order_no" size = "12"
        maxlength = "12" />
     </p>
     <p>
      <input type = "submit" id = "submit_button" name = "submit_button"
        value = "Retrieve Order Information" />
      <input type = "reset" id = "reset_button" name = "reset_button"
        value = "Reset Form" />
     </p>
   </form>
 </body>

</html>
```

Figure 6.18 shows how the form created in this example looks when rendered by the browser. Note that the default text displayed on the Submit and Reset controls has been modified in this example.


Chapter 6 – Building Forms
File Edit View History Bookmarks Window Help
file:///C:/Users/Jerry/Desktop/testit.html
Google
Retrieve Order Information
Reset Form

Enter your 12 character order number:

FIGURE 6.18

Controlling form submission and form reset.

Adding Buttons Using the button Element

As you have seen, you can use the `input` element to create all kinds of controls. (X)HTML also supports a number of other form controls, each of which is created using its own specific element. The `button` element is one such control. To work with the `button` element, you must specify a value of `button`, `submit`, or `reset` for its required type attribute. The `button` element works just like the controls created using the `input` element's `button`, `submit`, and `reset` types.

The following example demonstrates how to use the `button` element to add a button to a form.

```
<p>
  <button type = "button" id = "home_page" name = "home_page">
    Click here to return home
  </button>
</p>
```

Figure 6.19 shows how the resulting form looks when this example is loaded into a web browser.

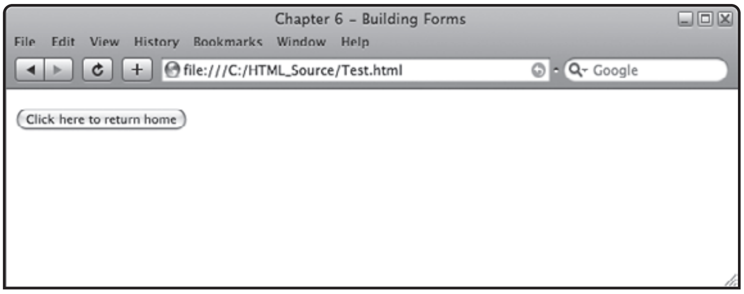

Chapter 6 – Building Forms
File Edit View History Bookmarks Window Help
file:///C:/HTML_Source/Test.html
Google
Click here to return home

FIGURE 6.19

Defining a custom button control.

Adding a Multiline Text Field Using the textarea Element

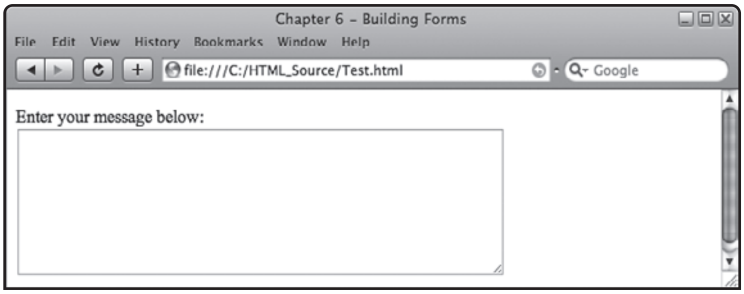
Earlier you learned how to add a single-line text field to a form by assigning a type of `text` to the `input` element's `type` attribute. This control is perfect for collecting small amounts of text.

However, if you need to collect anything longer than a single line, you will want to use the `textarea` element to create a multiline text field instead. The size of the resulting control is established by assigning a numeric value to the element's `rows` and `cols` attributes. If the amount of text that visitors enter exceeds the size of the control, the browser will automatically display scroll bars.

The `textarea` control is a block-level element, with a starting and ending tag. The following example demonstrates how to add a `textarea` control to a form, making it 10 rows tall and 60 characters wide.

```
<p>
  <label for = "contact_us">Enter your message below:<br /></label>
  <textarea id = "contact_us" name = "contact_us" rows = "10" cols="60">
  </textarea>
</p>
```

In this example, the `textarea` control is displayed as a large multiline text field, as demonstrated in Figure 6.20.


Chapter 6 – Building Forms
File Edit View History Bookmarks Window Help
file:///C:/HTML_Source/Test.html
Google

Enter your message below:

FIGURE 6.20

Multiline text fields are perfect for collecting large amounts of text from your visitors.

Even though it is a block-level control, the `textarea` element cannot contain other elements. It can remain empty or it can contain text. If you want, you can pre-populate a `textarea` control with text, as demonstrated here:

```
<p>
  <label for = "contact_us">Enter your message below:<br /></label>
  <textarea id = "contact_us" name = "contact_us" rows = "10" cols="60">
  To send us a message, overtype this text with your own text.
  </textarea>
</p>
```

Adding Drop-Down Lists to Forms

Another type of form control that you have no doubt come across many times on the internet is the drop-down control. You can add drop-down list controls of your own to your forms using the `select` and `option` elements. You can even group and label list contents using the `optgroup` element. Two forms of this control are supported. The first is a single-line drop-down list that expands when clicked to show a list of options from which the user can select. The second form of this control is a multiline control that displays a specified number of rows using the `select` element's optional `size` attribute.

Creating Drop-Down Lists Using the select Element

The first step in creating a drop-down list is to use the `select` element to establish the control, as shown here:

```
<p>
  <label for = "select_team">Select your team:</label>
  <select id = "select_team" name = "select_team">
  </select>
</p>
```

As shown in this example, by default the browser will render a single-line drop down list.

Adding Items to Drop-Down Lists Using the option Element

To create useful drop-down lists, you must populate the `select` element with items, which is done using instances of the `option` element. The `option` element is made up of an opening `<option>` and a closing `</option>` tag. The `option` element can only contain text. This text is displayed as an option in the drop-down list when displayed. Each option appears on a line by itself. The following example shows a form that consists of a single-line drop-down list that contains a listing of every team in the NBA.

```
<p>
  <select id = "select_team" name = "select_team">
    <option value = "" selected = "selected">- Pick a team -</option>
    <option>Celtics</option>
    <option>Nets</option>
    <option>Knicks</option>
    <option>76ers</option>
    <option>Raptors</option>
    <option>Mavericks</option>
    <option>Rockets</option>
    <option>Grizzlies</option>
    <option>Hornets</option>
    <option>Spurs</option>
    <option>Bulls</option>
    <option>Cavaliers</option>
    <option>Pistons</option>
    <option>Pacers</option>
    <option>Bucks</option>
    <option>Nuggets</option>
    <option>Timberwolves</option>
    <option>Trail Blazers</option>
    <option>City Thunder</option>
    <option>Jazz</option>
    <option>Hawks</option>
    <option>Bobcats</option>
    <option>Heat</option>
    <option>Magic</option>
    <option>Wizards</option>
    <option>Warriors</option>
    <option>Clippers</option>
    <option>Lakers</option>
    <option>Suns</option>
    <option>Kings</option>
  </select>
</p>
```

Figure 6.21 shows how the drop-down list looks when displayed by the browser.

As you can see in Figure 6.21, the drop-down list displays a small arrow button that when clicked, displays all of the options that have been added. To select an entry from the list, all the user has to do is click it. The selected option will be returned as the control's assigned value when passed to the form's handler. The list will then collapse and display the selected entry in a single field. As shown in Figure 6.21, the browser will automatically display a scroll bar in the event the list of options is too long to be displayed at one time. If you include the optional `width` attribute as part of the `select` element, you can pre-configure the width of the drop-down list.

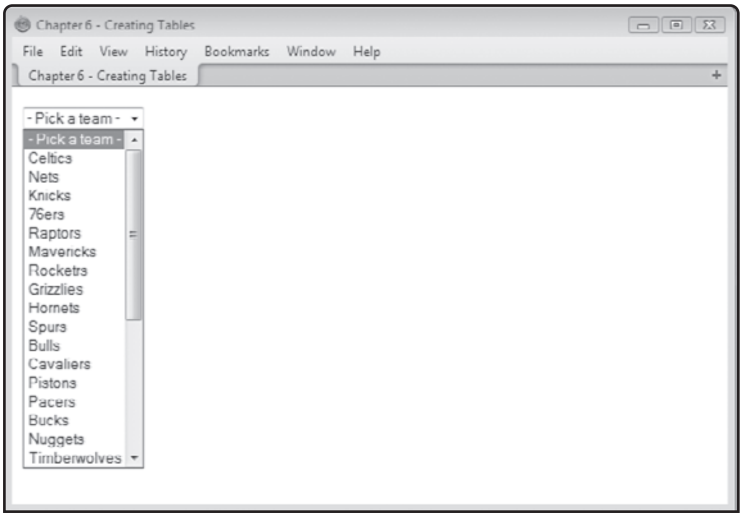


FIGURE 6.21

Drop-down lists let you present your visitors with a pre-determined list of options from which to choose.

TRICK

By default, the list entry selected by the user is passed as part of the form's data to the form handler. If you want to preselect an option, just add the optional `selected` attribute to the desired option (`selected = "selected"`). Typically, the first option defined in a `select` element is made the default value as was the case in the previous example.

If you want, you can substitute the value that is returned when an option is selected by specifying an optional value attribute to each option element, as demonstrated here:

```
<option value = "Boston Celtics"> Boston</option>
<option value = "New Jersey Nets"> Nets</option>
<option value = "New York Knicks"> Knicks</option>
<option value = "Philadelphia 76ers"> 76ers</option>
```

When one of these four options is selected, the assigned value is returned in place of the text that the user saw.

Adding Categories to Drop-Down Lists Using the optgroup Element

Sometimes it helps to organize `option` elements into labeled groups. This is done by embedding `optgroup` elements within the `select` element. The `optgroup` element has a required `label` attribute that is displayed within the drop-down lists. Most browsers display the `optgroup` element label in bold or italics font. By placing instances of this element at different locations within this list, you can organize its `option` elements into groups, as demonstrated in the following example.

```
<p>
  <select id = "select_team" name = "select_team">
    <option value = "" selected = "selected">- Pick a team -</option>
    <optgroup label = "Atlantic">
    <option>Celtics</option>
    <option>Nets</option>
    <option>Knicks</option>
    <option>76ers</option>
    <option>Raptors</option>
    <optgroup label = "Southwest">
    <option>Mavericks</option>
    <option>Rockets</option>
    <option>Grizzlies</option>
    <option>Hornets</option>
    <option>Spurs</option>
    <optgroup label = "Central">
    <option>Bulls</option>
    <option>Cavaliers</option>
    <option>Pistons</option>
    <option>Pacers</option>
    <option>Bucks</option>
    <optgroup label = "Northwest">
    <option>Nuggets</option>
    <option>Timberwolves</option>
    <option>Trail Blazers</option>
    <option>City Thunder</option>
    <option>Jazz</option>
    <optgroup label = "Southeast">
    <option>Hawks</option>
    <option>Bobcats</option>
    <option>Heat</option>
    <option>Magic</option>
    <option>Wizards</option>
    <optgroup label = "Pacific">
    <option>Warriors</option>
    <option>Clippers</option>
    <option>Lakers</option>
    <option>Suns</option>
    <option>Kings</option>
  </select>
</p>
```

Figure 6.22 shows the resulting drop-down list that is created when this example is loaded by the browser. As you can see, each `optgroup` element is displayed as a heading in the resulting drop-down list. Note that the `optgroup` elements are strictly informational; they cannot be selected. Only `option` elements can be selected.

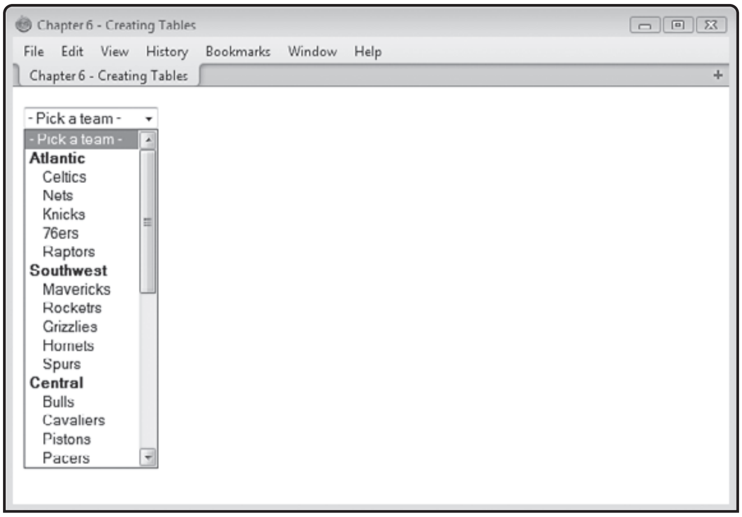


FIGURE 6.22
You can group drop-down list options to improve the organization of the list.

Creating a Multiline Drop-Down List

As previously stated, you can use the `select` element to create a multiline control. To do so, you must set the optional `multiple` attribute to `multiple` (`multiple = "multiple"`) as demonstrated in the following example. Multiline controls allow users to select one or more options by holding down the Shift, Control, or Command key while selecting different list options.

```
<p>
  <select id = "select_team" name = "select_team" size = "20"
    multiple = "Multiple">
    <optgroup label = "Atlantic">
    <option>Celtics</option>
    <option>Nets</option>
    <option>Knicks</option>
    <option>76ers</option>
    <option>Raptors</option>
    <optgroup label = "Southwest">
    <option>Mavericks</option>
    <option>Rockets</option>
    <option>Grizzlies</option>
    <option>Hornets</option>
    <option>Spurs</option>
    <optgroup label = "Central">
    <option>Bulls</option>
    <option>Cavaliers</option>
    <option>Pistons</option>
    <option>Pacers</option>
    <option>Bucks</option>
    <optgroup label = "Northwest">
    <option>Nuggets</option>
    <option>Timberwolves</option>
    <option>Trail Blazers</option>
    <option>City Thunder</option>
    <option>Jazz</option>
    <optgroup label = "Southeast">
    <option>Hawks</option>
    <option>Bobcats</option>
    <option>Heat</option>
    <option>Magic</option>
    <option>Wizards</option>
    <optgroup label = "Pacific">
    <option>Warriors</option>
    <option>Clippers</option>
    <option>Lakers</option>
    <option>Suns</option>
    <option>Kings</option>
  </select>
</p>
```

Here, a multiline drop-down list has been defined. Note that the `size` attribute has also been specified, setting the size of the control to 20 lines. Omitting the `size` attribute puts you at the mercy of the browser, resulting in inconsistent list presentation. Figure 6.23 shows the form that is displayed when this example is displayed.

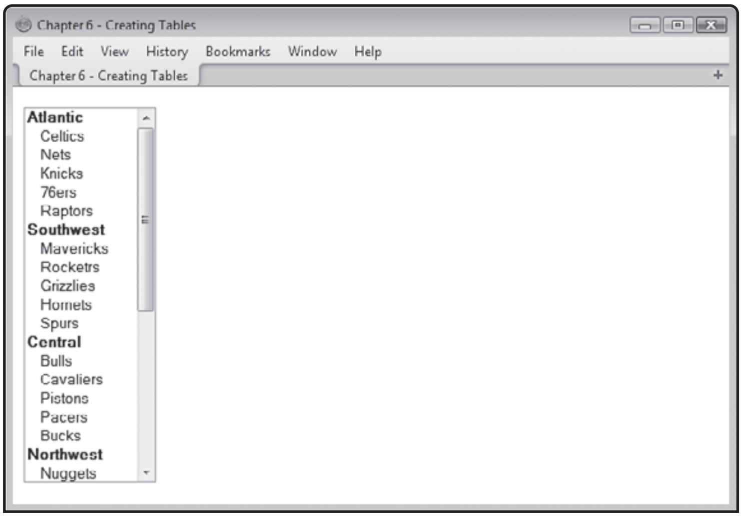


FIGURE 6.23

Multiline drop-down lists don't collapse down to a single line and support multiple selections.

REFINING FORM STRUCTURE

Like the rest of your (X)HTML pages, you need to ensure that your forms have a structure that is intuitive and structurally useful. You can use normal (X)HTML elements like headings and paragraphs to provide structure to your forms. In addition, (X)HTML provides several form-specific elements that provide additional structure.

Adding Descriptive Text to Controls Using the Label Element

As you have already seen numerous times in this chapter, you can use the `label` element to add descriptive text to forms. The `label` element is an inline element. As demonstrated in the following example, the `label` element is used to display text on forms.

```
<p>
  <label for = "username">Enter your name:</label>
  <input type = "text" id = "username" name = "username" size = "20"
    maxlength = "20" value = "Enter your name here" />
</p>
```

Here, a `label` element is used to supply a descriptive label for an element named `username`. The `label` element's optional `for` attribute is used to associate the `label` control with a specific element, based on that element's name.

Working with the fieldset Element

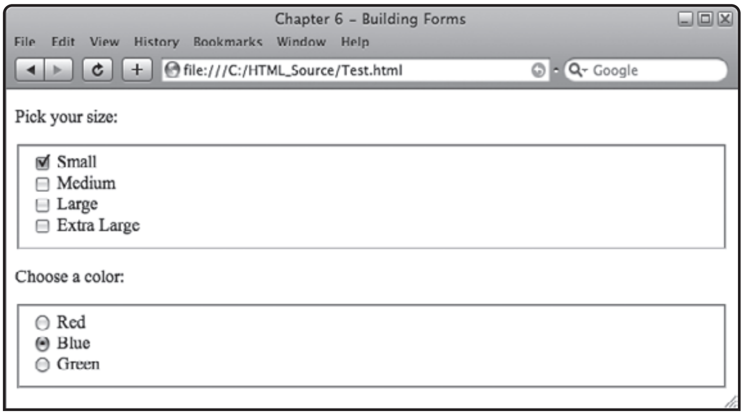
Another structure form control is the `fieldset` element, which is a block-level element that is used to group related collections of form elements into groups. By grouping controls together into logical collections, you can improve the readability of your forms. Grouping related form elements together in this manner makes them easier for your visitors to work with because it makes them more intuitive.

By default, `fieldset` elements display a thin border around all of the form controls that are embedded within them. To better understand the advantage of using the `fieldset` element to help add structure to your forms, take a look at this example.

```
<p>Pick your size:</p>
<fieldset>
  <input type = "checkbox" id = "checkbox1" name = "checkbox1"
    value = "Small" checked = "checked" />
  <label for = "checkbox1">Small</label><br />
  <input type = "checkbox" id = "checkbox2" name = "checkbox2"
    value = "Medium" />
  <label for = "checkbox2">Medium</label><br />
  <input type = "checkbox" id = "checkbox3" name = "checkbox3"
    value = "Large"/>
  <label for = "checkbox3">Large</label><br />
  <input type = "checkbox" id = "checkbox4" name = "checkbox4"
    value = "Extra Large" />
  <label for = "checkbox4">Extra Large</label>
</fieldset>

<p>Choose a color:</p>
<fieldset>
  <input type = "radio" id = "radio1" name = "radio"
    value = "Red" />
  <label for = "radio1">Red</label><br />
  <input type = "radio" id = "radio2" name = "radio"
    value = "Blue" checked = "checked" />
  <label for = "radio2">Blue</label><br />
  <input type = "radio" id = "radio3" name = "radio"
    value = "Green" />
  <label for = "radio3">Green</label><br />
</fieldset>
```

Here, a form has been visually organized into two logical sections using two `fieldset` elements. The first `fieldset` element was used to group a collection of four checkbox controls and the second `fieldset` element was used to group a set of three radio controls.

Figure 6.24 shows how the resulting form looks when the web page containing it is rendered by the browser.


Chapter 6 – Building Forms
File Edit View History Bookmarks Window Help
file:///C:/HTML_Source/Test.html
Google

Pick your size:

[✓] Small
☐ Medium
☐ Large
☐ Extra Large

Choose a color:

○ Red
● Blue
○ Green

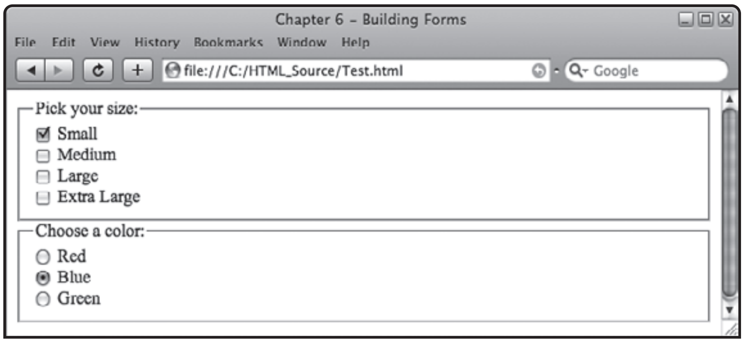
FIGURE 6.24
Using `fieldset` elements to group collections of controls.

You can make your `fieldset` elements even more useful if you include a `legend` element when working with them. A `legend` element allows you to specify a text string that is displayed as a special type of heading at the top of the `fieldset` element. Most browsers place the text specified by the `legend` element in the upper-left corner of the fieldset, embedding it within the `fieldset` element's border. When used, the `legend` element must be embedded within a `fieldset` element, usually as its first statement.

As an example of the impact that the `legend` element has on a `fieldset`, look at the following example.

```
<fieldset>
  <legend>Pick your size:</legend>
  <input type = "checkbox" id = "checkbox1" name = "checkbox1"
    value = "Small" checked = "checked" />
  <label for = "checkbox1">Small</label><br />
  <input type = "checkbox" id = "checkbox2" name = "checkbox2"
    value = "Medium" />
  <label for = "checkbox2">Medium</label><br />
  <input type = "checkbox" id = "checkbox3" name = "checkbox3"
    value = "Large"/>
  <label for = "checkbox3">Large</label><br />
  <input type = "checkbox" id = "checkbox4" name = "checkbox4"
    value = "Extra Large" />
  <label for = "checkbox4">Extra Large</label>
</fieldset>

<fieldset>
  <legend>Choose a color:</legend>
  <input type = "radio" id = "radio1" name = "radio"
    value = "Red" />
  <label for = "radio1">Red</label><br />
  <input type = "radio" id = "radio2" name = "radio"
    value = "Blue" checked = "checked" />
  <label for = "radio2">Blue</label><br />
  <input type = "radio" id = "radio3" name = "radio"
    value = "Green" />
  <label for = "radio3">Green</label><br />
</fieldset>
```

By adding a `legend` element to a `fieldset` element, you can strengthen the structure of the `fieldset` element's contents. Figure 6.25 shows how the form created by the previous examples will look when rendered by the browser.


Chapter 6 – Building Forms
File Edit View History Bookmarks Window Help
file:///C:/HTML_Source/Test.html
Google

Pick your size:
- [✓] Small
- [ ] Medium
- [ ] Large
- [ ] Extra Large

Choose a color:
- ○ Red
- ● Blue
- ○ Green

FIGURE 6.25

Using the `legend` element to add a caption to a `fieldset`.

A Complete Form Example

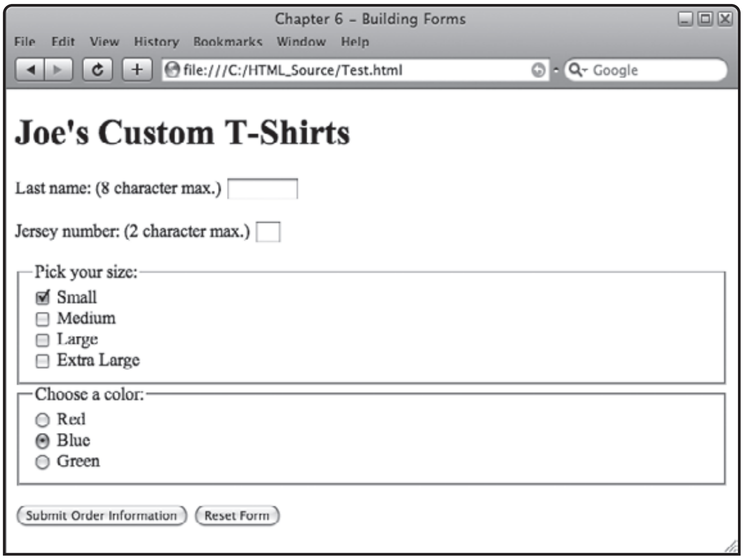
To help tie together everything that you have learned in this chapter about working with forms, let's look at a more complete example of a form that is made up of a number of different elements.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">

  <head>
    <meta http-equiv="Content-type" content="text/xhtml; charset=UTF-8" />
    <title>Chapter 6 - Building Forms</title>
  </head>

  <body>

    <h1>Joe's Custom T-Shirts</h1>

    <form action = "/cgi-bin/processdata.cgi" method = "post">

      <p>
        <label for = "name">Last name: (8 character max.)</label>
        <input type = "text" id = "name" name = "name" size = "8"
          maxlength = "8" />
      </p>

      <p>
        <label for = "number">Jersey number: (2 character max.)</label>
        <input type = "text" id = "number" name = "number" size = "2"
          maxlength = "2" />
      </p>

      <fieldset>
        <legend>Pick your size:</legend>
        <input type = "checkbox" id = "checkbox1" name = "checkbox1"
          value = "Small" checked = "checked" />
        <label for = "checkbox1">Small</label><br />
        <input type = "checkbox" id = "checkbox2" name = "checkbox2"
          value = "Medium" />
        <label for = "checkbox2">Medium</label><br />
        <input type = "checkbox" id = "checkbox3" name = "checkbox3"
          value = "Large"/>
        <label for = "checkbox3">Large</label><br />
        <input type = "checkbox" id = "checkbox4" name = "checkbox4"
          value = "Extra Large" />
        <label for = "checkbox4">Extra Large</label>
      </fieldset>

      <fieldset>
        <legend>Choose a color:</legend>
        <input type = "radio" id = "radio1" name = "radio"
          value = "Red" />
        <label for = "radio1">Red</label><br />
        <input type = "radio" id = "radio2" name = "radio"
          value = "Blue" checked = "checked" />
        <label for = "radio2">Blue</label><br />
        <input type = "radio" id = "radio3" name = "radio"
          value = "Green" />
        <label for = "radio3">Green</label><br />
      </fieldset>

      <p>
        <input type = "submit" id = "submit_button" name = "submit_button"
          value = "Submit Order Information" />
        <input type = "reset" id = "reset_button" name = "reset_button"
          value = "Reset Form" />
      </p>
    </form>

  </body>

</html>
```

In this example, a form is defined that will submit the data it collects to a program running on the web server using the `form` method's `post` method. The first four `form` elements on the page are a pair of labels and text controls. The first text control is named `name` and configured to be eight characters long and to accept eight characters of input. The second text control is named `number` and is set to a size and length of two characters. Note that both pairs of controls are embedded within their own paragraphs, keeping them separate and meeting XHTML Strict 1.0's requirement that all form elements be placed within block-level elements.

Next, a set of five checkbox controls is displayed. Note that a `fieldset` control has been used to visually group the checkbox controls together. A `legend` element has been added to assign it a descriptive header. A `label` element has been included for each checkbox control, providing a descriptive text string that identifies each checkbox's value. Also, note that the first of the four checkbox controls has been pre-selected.

A second `fieldset` is then used to group four radio controls. Again, a `legend` element has been specified. Note that the second radio button has been selected. The last two controls on the form are a Submit and a Reset button. Figure 6.26 shows how this table looks when rendered by the browser.


Chapter 6 – Building Forms
File Edit View History Bookmarks Window Help
file:///C:/HTML_Source/Test.html
Google

Joe's Custom T-Shirts

Last name: (8 character max.)

Jersey number: (2 character max.)

Pick your size:
- [✓] Small
- [ ] Medium
- [ ] Large
- [ ] Extra Large

Choose a color:
- ○ Red
- ● Blue
- ○ Green

Submit Order Information | Reset Form

FIGURE 6.26

An example of a complete form made up of multiple form controls.

ADVICE ON GOOD FORM DESIGN

As you have seen, there is a lot to form design. You have to know how to work with a number of different types of elements and element attributes. In addition to creating forms that are functionally correct, you also need to consider the structural and organizational aspects of your forms. Specifically, you need to make sure that the forms you create are well organized, easy to fill out, and as intuitive as you can make them.

To design forms that are really useful and easy for your visitors to understand, their layout should provide a clear and intuitive path that is easy for your visitors to scan and understand without a lot of work. It is often a good idea to provide your visitors with instructions on how to fill out your forms, letting them know what information they will need to have on hand to fill out your form and what you will do with the information they provide to you. If you do not want to crowd the web page with instructional text, you might instead display a link at the top of your form to a help page that explains how to complete the form.

As you develop your forms, make sure you always provide labels for every control. Also, include visual markers where appropriate. Be explicit regarding information that you want to collect and the format you need it in. For example, if you are using a text field to collect a person's birth date and you want that information entered using a format of mm/dd/yy, then make sure the control's label tells the user what format to use when keying in their birth date.

Always keep an eye on the overall layout of your forms. Present form controls in a logical matter. Group related controls together and when appropriate take advantage of fieldsets as a means of grouping related elements together. This can help make your forms easier for your visitors to scan. It also helps to point out any relationships between related controls.

If your forms have required fields, mark them as such so that your visitors know precisely what information they need to provide. One way of doing this is to place an asterisk beside these fields. Alternatively, you might make the labels for these fields bold or display them in a different color.

Always keep in mind that filling out large and complex forms is not a lot of fun. This can discourage your visitors. As a general rule, do your best to keep things short and simple. Whenever possible, streamline your forms by eliminating the collection of unnecessary optional data. Also, when possible, provide default values for your controls. If your visitors need to change these values, they can; otherwise, they can simply accept these values and move along.

If you have added text controls to your form and limited the amount of text they can contain using the `maxlength` attribute, make sure that you let your visitors know about the limitation. If your website makes use of multiple forms, be consistent in their design and presentation.

HINT Once you have learned how to work with JavaScript, covered in Chapter 9, make sure that you include scripts in your web page that validate form content and user feedback when errors are found. Also, consider providing examples of valid input when informing a visitor that invalid input has been collected in one of the form's controls.

Back to the Number Guessing Game

All right, now it is time to return your attention to this chapter's project, the Number Guessing Game. The object of this game is to guess a randomly generated secret number in as few guesses as possible. To submit guesses, the player must click on one of ten radio buttons, representing guesses of 1 to 10. Once the player makes a selection, he must click on a button labeled Check Guess in order to signal the end of the current turn. The game will then process the player's input using a JavaScript and determine whether the player's guess was correct, too low, or too high.

Designing the Application

The development of this game will be performed in a number of steps, as outlined here:

1. Create a new XHTML document.
2. Outline the document's markup.
3. Develop the document's JavaScript.
4. Load and test the HTML page.

Step 1: Creating a New XHTML Document

As has been the case with all of the web projects that you have worked on up to this point in the book, the first step in the creation of this game project is to create a new text file. Do so using your preferred code or text editor, naming the file NumberGuess.html and storing it in the same folder as all of your other (X)HTML projects.

Step 2: Developing the Document's Markup

The next step in the creation of the Number Guessing Game is to assemble the web document's markup. Create these statements by adding the following elements to the NumberGuess.html file.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">

  <head>

    <meta http-equiv="Content-type" content="text/xhtml; charset=UTF-8" />
    <title>Number Guessing Game</title>
  </head>

  <body>

  </body>

</html>
```

As you can see, all of the statements needed to create a well-formed XHTML page have been provided, including the web page's `meta` and `title` elements. Once you have completed this task, it's time to add the statements representing the document's content, which consists of a heading, followed by a paragraph, and then a form. To do so, embed the following statements into the document's `body` section.

```
<h1>The Number Guessing Game</h1>
<p>Try to guess the secret number in as few guesses as possible</p>

<form action = "NumberGuess.html">

  <p>Select Number:</p>
    <input type = "radio" id = "radio1" name = "radio" value = "1" />
    <label for = "radio1">1</label><br />

    <input type = "radio" id = "radio2" name = "radio" value = "2" />
    <label for = "radio2">2</label><br />

    <input type = "radio" id = "radio3" name = "radio" value = "3"/>
    <label for = "radio3">3<label><br />

    <input type = "radio" id = "radio4" name = "radio" value = "4" />
    <label for = "radio4">4</label><br />

    <input type = "radio" id = "radio5" name = "radio" value = "5" />
    <label for = "radio5">5</label><br />

    <input type = "radio" id = "radio6" name = "radio" value = "6" />
    <label for = "radio6">6</label><br />

    <input type = "radio" id = "radio7" name = "radio" value = "7" />
    <label for = "radio7">7</label><br />

    <input type = "radio" id = "radio8" name = "radio" value = "8" />
    <label for = "radio8">8</label><br />

    <input type = "radio" id = "radio9" name = "radio" value = "9" />
    <label for = "radio9">9</label><br />
    <input type = "radio" id = "radio10" name = "radio" value = "10" />
    <label for = "radio10">10</label>
  </p>
  <p>
    <button type = "button" name = "validate" onclick = "check_guess()">
      Check Guess
    </button>
  </p>

</form>
```

The first two statements display a heading and a paragraph that explains the rules for playing the game. Next, a form named `guess_form` is created. The data collected by this form will remain local and will not be passed on to a web server for further processing. As a result, the `form` elements required `action` attribute has been set to `NumberGuess.html` and the optional `method` attribute has been omitted.

The form itself is made up of a series of radio buttons, created using 10 pairs of `input` and `label` elements. Each `input` element is assigned a numeric value from 1 to 10, representing the range of guesses allowed by the game. The `label` element that follows each `input` element displays the radio control's assigned value for the player to see.

The last element that makes up the form is created using a `button` element. This creates a button control labeled `Check Guess`. When clicked, a JavaScript function named `check_guess()`, located in a script in the document's `head` section, is executed.

HINT

When called, the `check_guess()` function will process the contents of the form and determine whether the player has won the game or has entered a guess that is too high or too low. You will learn all about JavaScript functions and their execution in Chapter 9.

Step 3: Developing the Document's Script

The Number Guessing Game includes a JavaScript that needs to be located in the document's `head` section. The statements that make up this script, shown below, should be embedded in the `head` section, immediately following the document's `title` element.

```
<script type = "text/javascript">
<!-- Start hiding JavaScript statements

  var randomNo = 0;
  var player_choice;
  randomNo = 1 + Math.random() * 9; //Create random number from 1-10
  randomNo = Math.round(randomNo);  //Convert number to an integer

  function check_guess() {

    var radioButtons = document.getElementsByName('radio');
    for (i = 0; i < 10; i++) {
      if (radioButtons[i].checked == true) {
        player_choice = radioButtons[i].value

        //Analyze the player's guess
        if (player_choice == randomNo) {  //See if the guess is correct

          window.alert("Correct! Reload the web page to play again.");
        } else {
          if (player_choice > randomNo) { //See if the guess is high
            window.alert("Incorrect. Your guess was too high.");
          } else {                //See if the player's guess is low
            window.alert("Incorrect. Your guess was too low.");
          }
        }

      }
    }

  }

// End hiding JavaScript statements -->
</script>
```

At a high level, what this script does is declare a pair of variables that will be used to store both the game's randomly generated number as well as the player's guesses. Next, a random number between 1 and 10 is generated and assigned to the `randomNo` variable. The rest of the JavaScript consists of a custom function that is executed each time the player clicks on the game's Check Guess button. When executed, this function retrieves the player's choice and compares it to the game's random number. If the two values match, the player wins and the game ends. Otherwise, a message is displayed in a popup dialog window that lets the player know whether the guess was too low or too high.

The Finished HTML Document

At this point, your copy of the GuessNumber.html document should be complete. To make sure that you have everything in order, look at the following example, which shows what the finished document should look like.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">

  <head>
    <meta http-equiv="Content-type" content="text/xhtml; charset=UTF-8" />
    <title>Number Guessing Game</title>

    <script type = "text/javascript">
    <!-- Start hiding JavaScript statements

      var randomNo = 0;
      var player_choice;

      randomNo = 1 + Math.random() * 9; //Create random number from 1-10
      randomNo = Math.round(randomNo);  //Convert number to an integer

      function check_guess() {

        for (i = 0; i < document.guess_form.radio.length; i++) {
          if (document.guess_form.radio[i].checked == true) {
            player_choice = document.guess_form.radio[i].value;

            //Analyze the player's guess
            if (player_choice == randomNo) {  //See if the guess is correct

              window.alert("Correct! Reload the web page to play again.");
            } else {
              if (player_choice > randomNo) { //See if the guess is high
                window.alert("Incorrect. Your guess was too high.");
              } else {                //See if the player's guess is low
                window.alert("Incorrect. Your guess was too low.");
              }
            }

          }
        }

        if (document.guess_form.radio.value == "5") {
          window.alert("Yup!");
        }

      }

    // End hiding JavaScript statements -->
    </script>

  </head>

  <body>

    <h1>The Number Guessing Game</h1>
    <p>Try to guess the secret number in as few guesses as possible</p>

    <form name = "guess_form" action = "NumberGuess">

      <p>Select Number:</p>

        <input type = "radio" id = "radio1" name = "radio"
        value = "1" />
        <label for = "radio1">1</label><br />

         <input type = "radio" id = "radio2" name = "radio"
        value = "2" />
        <label for = "radio2">2</label><br />

        <input type = "radio" id = "radio3" name = "radio"
        value = "3"/>
        <label for = "radio3">3<label><br />
        <input type = "radio" id = "radio4" name = "radio"
        value = "4" />
        <label for = "radio4">4</label><br />

        <input type = "radio" id = "radio5" name = "radio"
        value = "5" />
        <label for = "radio5">5</label><br />

        <input type = "radio" id = "radio6" name = "radio"
        value = "6" />
        <label for = "radio6">6</label><br />

        <input type = "radio" id = "radio7" name = "radio"
        value = "7" />
        <label for = "radio7">7</label><br />

        <input type = "radio" id = "radio8" name = "radio"
        value = "8" />
        <label for = "radio8">8</label><br />

        <input type = "radio" id = "radio9" name = "radio"
        value = "9" />
        <label for = "radio9">9</label><br />

        <input type = "radio" id = "radio10" name = "radio"
        value = "10" />
        <label for = "radio10">10</label>
      </p>

      <p>
        <button type = "button" name = "validate" onclick = "check_guess()">
          Check Guess
        </button>
      </p>

    </form>
```

```
  </body>

</html>
```

Step 4: Loading and Testing the Number Guessing Game

Once you are ready, load your new game into your web browser and see how things have turned out. To do so, open your browser and execute the following procedure.

1. Click on File > Open File. A standard file open dialog will appear.
2. Navigate to the folder where you stored the web page and select it.
3. Click on the Open button. The browser will load and render our XHTML document, allowing game play to begin.

Assuming that everything works as expected, load your XHTML document using one or two other browsers in order to ensure that its content displays consistently.

Summary

This chapter showed you how to create and work with tables and forms. Using the information and examples provided, you should be able to display any type of tabular data and collect visitor input using all kinds of form controls, including text, password, multiline text, drop-down list, radio, checkbox, and different types of buttons controls. In addition, this chapter showed you how to add borders, row, and column headers to your tables and even to merge multiple cells together in order to create larger cells. You also learned how to use fieldsets to group form controls together into logical collections in order to make your forms easier to use. On top of all this, you learned how to create the Number Guessing Game.

Challenges

1. **As currently designed, the Number Guessing Game provides the player with a base minimum of instructions. Consider modifying the game to provide the player with additional information on how to play.**
2. **Consider making the game more challenging by expanding the range of numbers from 1 to 10 to 1 to 20 or even higher.**