

```

        body {background-color: #B0C4DE;}
    </style>
</head>
<body>
<!-- /Body container -->
    <!-- (background = border, padding = border width
        margin = centered table) -->
    <table border="0" cellpadding="4px" cellspacing="0"
        style="background-color: black;
        margin: 0 auto;">
    <tr>
    <td>
        <!-- Floating page -->
        <!-- (padding = page margin) -->
        <table border="0" cellpadding="5px" cellspacing="0"
            width="732px"
            style="background-color: #FFFFFF;">
        <tr valign="top">
        <td>
            <!-- Page content -->
            <p>Content goes here.</p>
            <!-- Page content -->
        </td>
        </tr>
        </table>
        <!-- /Floating page -->
    </td>
    </tr>
    </table>
<!-- /Body container -->
</body>
</html>

```

Tip

Note the comments in the code delimiting the individual tables and content areas. It is a best practice to follow standard code formatting (indentation, liberal white space, and so on) and add sufficient comments to easily keep track of all your tables, how they are formatted, and what they accomplish. ■

If you want more of a drop shadow effect, you can play with the borders of the floating page, setting two adjacent borders to a nonzero value, as shown in the following code:

```

<!-- Floating page -->
<!-- (padding = page margin) -->
<table border="0" cellpadding="5px" cellspacing="0"
    width="732px" height="900px"
    style="background-color: #FFFFFF;
    border-right: 4px solid black;
    border-bottom: 4px solid black;">

```

Part I: Creating Content with HTML

FIGURE 9-19

Another popular layout: floating page and multiple columns of content



This code will visually increase the width of the right and bottom borders, giving the page a more realistic, three-dimensional drop shadow effect.

Tip

Keep in mind that you can combine various techniques within the same document. For example, you can put a two-column layout on a floating page by nesting a two-column table in the content area of the floating page table. Then, within one of the columns, you can evenly space out a handful of graphics by nesting another table in the column. The possibilities are endless. ■

Odd graphics and text combinations

You can also use tables to combine text and graphics in nonstandard layouts. For example, note the header in Figure 9-22. The header graphic is actually several pieces, as shown in Figure 9-23.

A table with no padding and no spacing is used to put the pieces back together into a complete image, while enabling text to flow to the right of the face portion.

FIGURE 9-20

The floating page and two-column layout with visible tables



Code for the completed header is shown here:

```
<!-- Heading container -->
<table border="0" cellpadding="0" cellspacing="0">
  <tr>
    <td valign="top">
      
    </td>
    <td>
<!-- Nav and main graphic -->
<table border="0" cellpadding="0" cellspacing="0">
  <tr>
    <td width="100%">
      <!-- Nav bar -->
      <table border="0" cellpadding="0" cellspacing="0"
        width="100%">
        <tr>
          <td width="25%">
            <a href="archive/index.html" onfocus="this.blur()"
              onMouseOver="archive.src='images/archive_punch_on.gif'"
```

```
        onMouseOut="archive.src=`images/archive_punch_off.gif`"
      >
      
    </a>
  </td>
  <td width="25%">
    <a href="guest/index.html" onFocus="this.blur()"
onMouseOver="guest.src=`images/g_punch_on.gif`"
onMouseOut="guest.src=`images/g_punch_off.gif`" >
      </a>
  </td>
  <td width="25%">
    <a href="mailto:email@example.com" onFocus="this.blur()"
onMouseOver="email.src=`images/e_punch_on.gif`"
onMouseOut="email.src=`images/e_punch_off.gif`" >
      </a>
  </td>
  <td width="25%">
    <a href="about/index.html" onFocus="this.blur()"
onMouseOver="about.src=`images/a_punch_on.gif`"
onMouseOut="about.src=`images/a_punch_off.gif`">
      </a>
  </td>
</tr>
</table>
<!-- /Nav bar -->
</td>
</tr>
<tr>
  <td width="100%">
  </td>
</tr>
</table>
<!-- /Nav and main graphic -->
</td>
</tr>
<tr>
  <td height="158" valign="top">
  <p>SECONDARY CONTENT HERE</p>
  </td>
  <td valign="top">
    <p>MAIN CONTENT HERE</p>
  </td>
</tr>
</table>
<!-- /Heading container -->
```

FIGURE 9-21

A floating page can add a bit of simple design to your documents.



Note

The preceding listing has been formatted for legibility. However, when put into use on an actual Web page, all spaces and line breaks that aren't contained within tags themselves should be removed from between the `<td>` and `</td>` tags. For example, consider the following code from the listing:

```
<td width="25%">
  <a href="guest/index.html" onfocus="this.blur()"
    onMouseOver="guest.src='images/g_punch_on.gif'"
    onMouseOut="guest.src='images/g_punch_off.gif'" >
    </a>
  </td> ■
```

This code should be changed to resemble the following, where the only spaces and line breaks are within the angle brackets of a tag (`<` and `>`):

```
<td width="25%"><a href="guest/index.html"
onfocus="this.blur()"
onMouseOver="guest.src='images/g_punch_on.gif'"
```

Part I: Creating Content with HTML

```
onMouseOut="guest.src=`images/g_punch_off.gif`" ></a></td>
```

Using this technique, you can wrap text and graphics around each other in a variety of ways. For example, if the graphic used in the preceding example descended on the right as well, you could use three columns — pieces of the graphic in the first and third, text in the middle.

Caution

It's important to watch for errant white space in and around your tags when formatting a page using tables. For example, one single space within a `<td>` pair can create a visible seam in between the graphics that make up your header. To avoid this problem, place the line breaks in your code within the tags, between attributes and such. ■

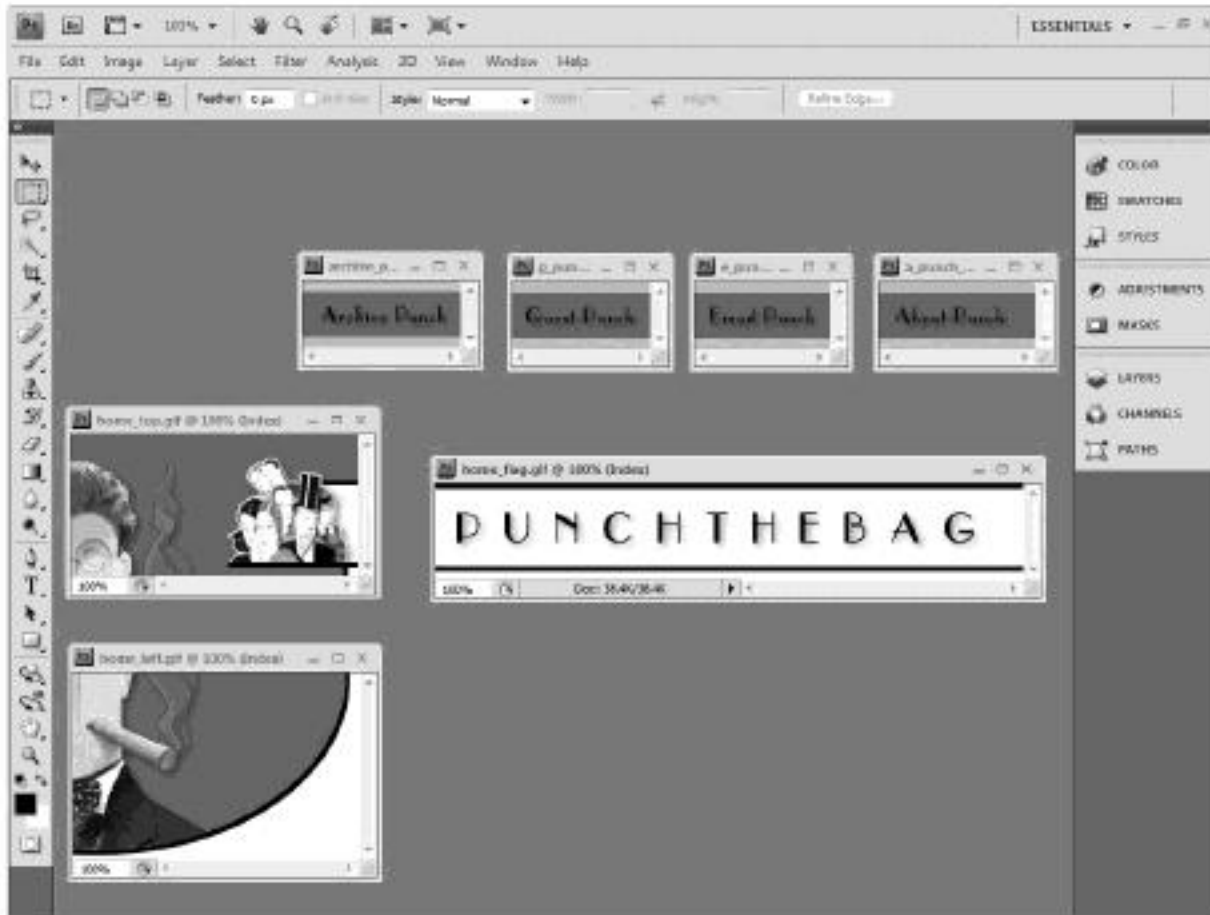
FIGURE 9-22

Presenting graphics and text in a nonstandard format



FIGURE 9-23

The various pieces of the header graphic



Navigational menus and blocks

The sample page header has its navigational elements in a row at the top of the page. You can construct similar, vertical layouts for your navigational elements using `rowspan` attributes in your tables. For example, consider the following code and the output shown in Figure 9-24:

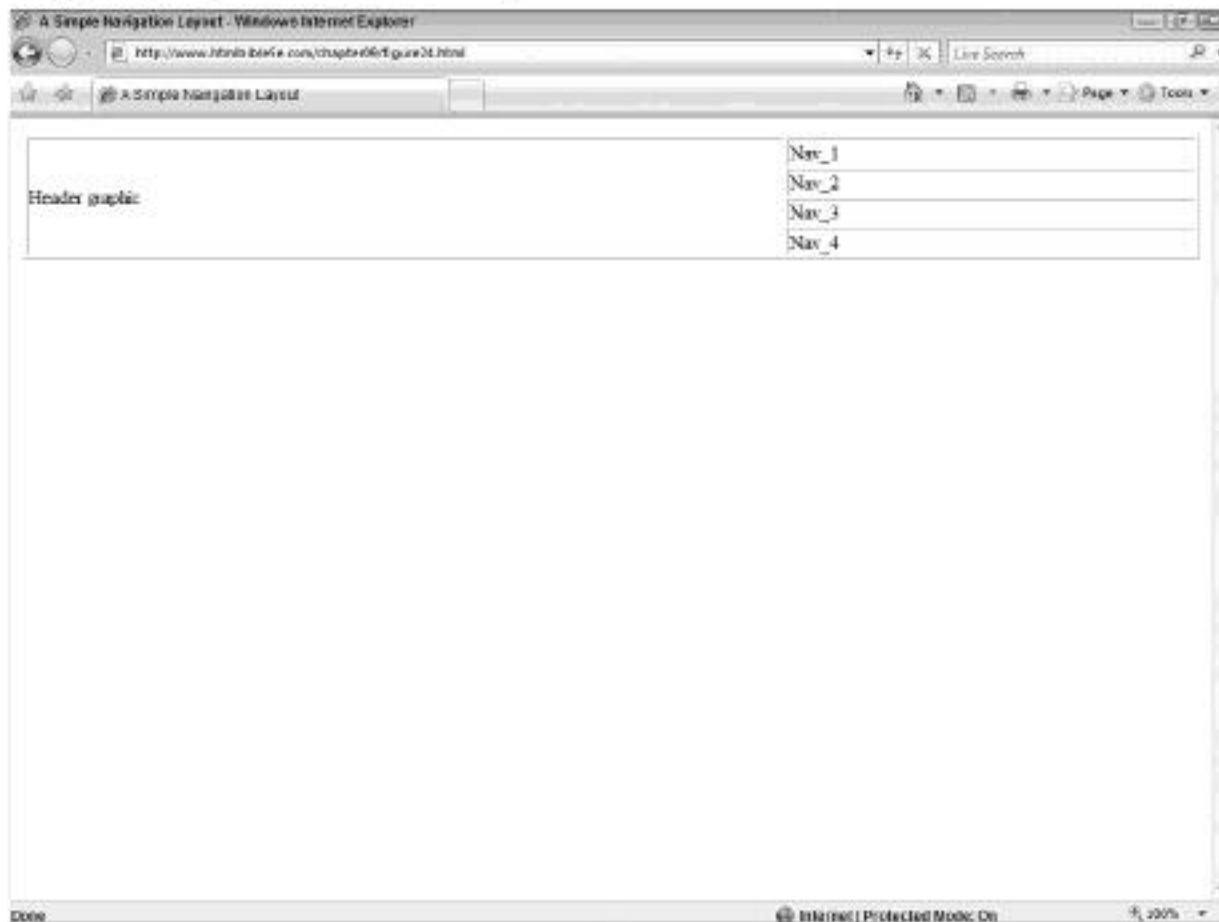
```
<table border="1" width="100%">
  <tr>
    <td rowspan="4">
      <p>Header graphic</p>
    </td>
    <td>
      <p>Nav_1</p>
    </td>
  </tr>
  <tr>
    <td>
      <p>Nav_2</p>
    </td>
  </tr>
  <tr>
    <td>
      <p>Nav_3</p>
    </td>
  </tr>
  <tr>
    <td>
      <p>Nav_4</p>
    </td>
  </tr>
</table>
```

Part I: Creating Content with HTML

```
<td>
  <p>Nav_2</p>
</td>
</tr>
<tr>
  <td>
    <p>Nav_3</p>
  </td>
</tr>
<tr>
  <td>
    <p>Nav_4</p>
  </td>
</tr>
</table>
```

FIGURE 9-24

Using `rowspan`, you can create vertically stacked elements.



Note

As you have no doubt realized, there are multiple ways to accomplish many of the designs shown in this chapter. For example, you could just as easily nest a one-column table in a cell instead of using `rowspan`. ■

Multiple columns

As covered earlier in this chapter, you can use tables to position elements in columns. This technique can be used for a variety of layout purposes:

- Providing navigation bars to the right or the left of text
- Putting text into columns
- More precise positioning controls, putting text next to graphics, and so forth

Columnar formatting is simple to accomplish, as shown in the following code:

```
<table border="1" cellspacing="0" cellpadding="5px"
  width="100%">
  <colgroup>
    <col width="50%">
    <col width="50%">
  </colgroup>
  <tr>
    <td colspan="2">Header graphic or navigation can go here</td>
  </tr>
  <tr>
    <td>First column content...</td>
    <td>Second column content...</td>
  </tr>
</table>
```

The output of this code is shown in Figure 9-25.

Note

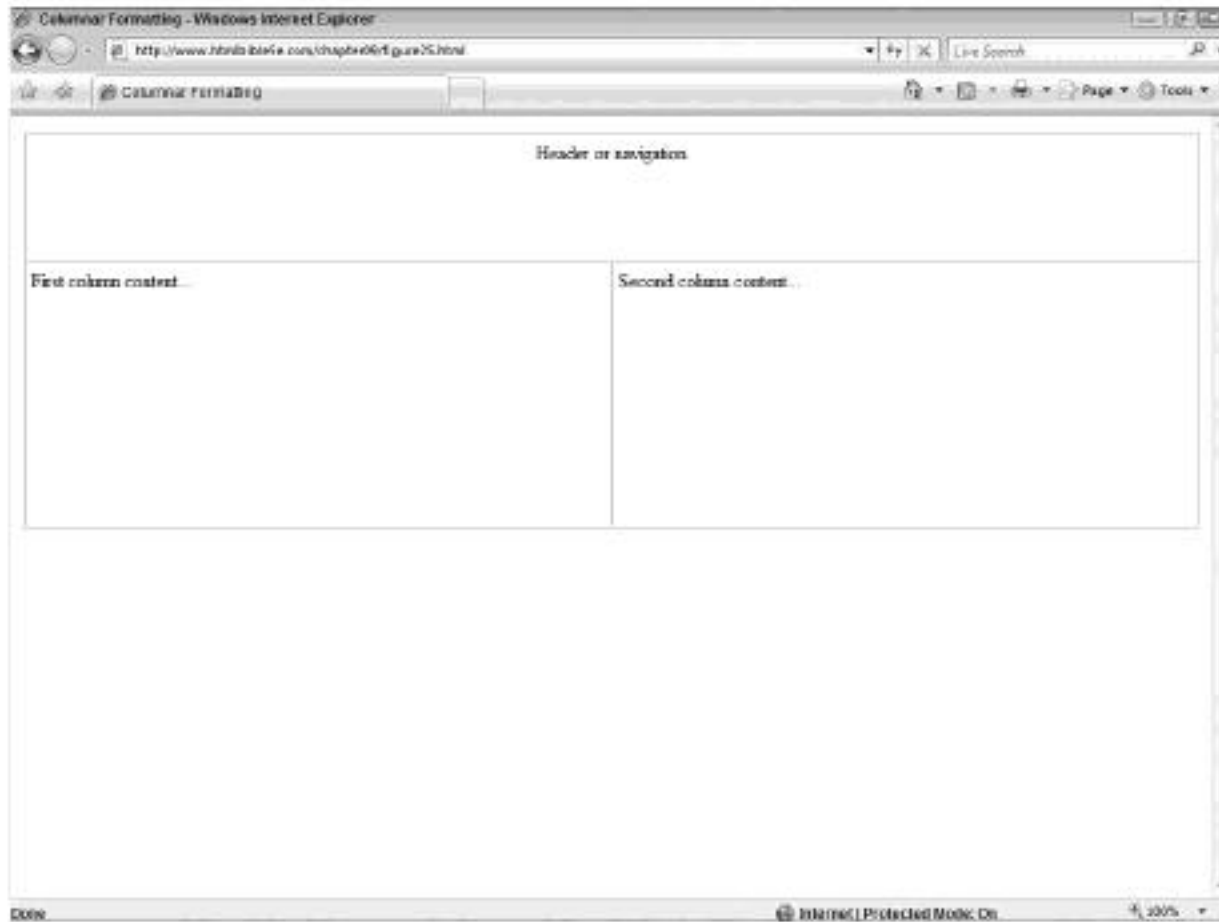
One caveat to creating columns with tables is that the content doesn't automatically wrap from one column to the next (as in a newspaper). You must split the text between the columns manually. ■

The columns do not have to be the same size or proportional to each other. You can define the columns in any size you need by using the appropriate formatting attributes. For example, to create a navigation column to the left that is 200 pixels wide and a text column to the right that is 400 pixels wide, you could use this column definition:

```
<colgroup>
  <col width="200px">
  <col width="400px">
</colgroup>
```

FIGURE 9-25

A simple two-column format



Summary

This chapter covered the basics of HTML tables. You learned how to define a table, what each table element is used for, and how to format table elements to achieve various desired effects.

This chapter also showed you the glamorous side of tables, how they can be used to provide complex formatting structures in HTML. As mentioned throughout this book, CSS provides a better mechanism for creating and controlling layout while maintaining the laudable goal of keeping presentation and content separate. That said, tables still provide a viable means to align, format, and lay out blocks of text.

From here you will learn about additional structured elements — namely, frames and forms (Chapters 10 and 11) and continue through the rest of the HTML element categories. Once you venture into Part III of this book, you will first learn about the basics of CSS (Chapters 25 through 28) before learning about tags for specific elements, such as in Chapter 30, which describes table- and text-specific CSS.

Frames

Several years ago, almost every document on the Web contained frames. The frameset structure provided an easy way to create multiple, separate scrolling areas in a user agent window and a flexible mechanism to modify the content of frames.

However, frames have turned out to be more of a fad. You can have many of the benefits provided by using frames through the infinitely more flexible and powerful CSS formatting methods.

That said, frames still have their uses and have even spawned their own official Document Type Definitions (DTDs) to handle their special tags and needs. This chapter introduces the concept of frames and shows you how to add them to your documents.

IN THIS CHAPTER

Frames Overview

Framesets and Frame Documents

Targeting Links to Frames

Nested Framesets

Inline Frames

Frames Overview

At their simplest level, frames provide multiple separately scrollable areas within one user window. Many non-Web applications use the technique of separate panes to provide organization and controls. For example, Figure 10-1 shows Windows Explorer using the left panes to display Favorite Links and Folders, and the right pane to display files within the selected folder.

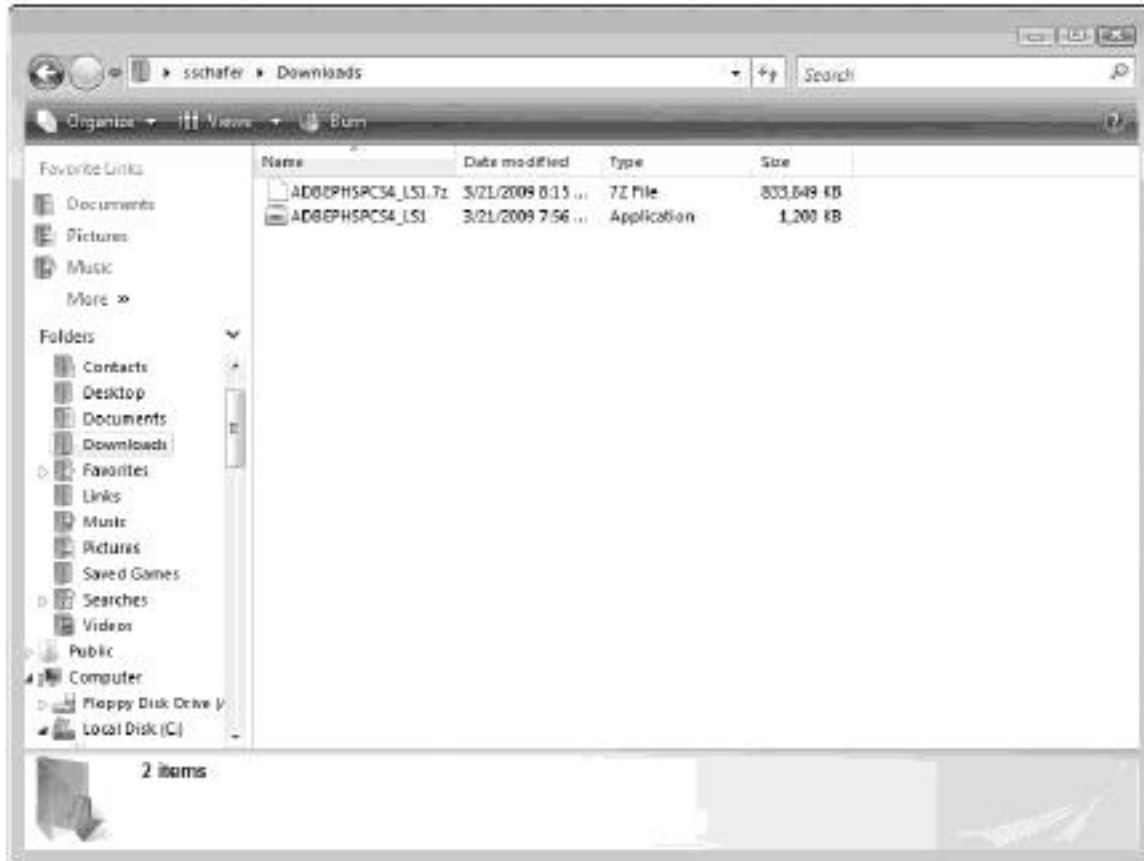
As you have no doubt noticed, the different panes in applications such as Windows Explorer can be manipulated separately from other panes. The same is true for documents utilizing frames.

For example, Figures 10-2 and 10-3 show the same document but the window in Figure 10-3 has been scrolled to view the bottom of the text in the document. This has caused the navigation bar to scroll as well, in this case almost off the screen, where part of it can no longer be immediately accessed.

Part I: Creating Content with HTML

FIGURE 10-1

Applications such as Windows Explorer use multiple panes to display a variety of information and controls.



Now take a look at Figure 10-4. Each element — the top banner, the navigation bar, and the main content — has been placed in a separate frame. When the main content is scrolled, the banner and the navigation menu remain static within their own regions.

Framesets and Frame Documents

Frames are a bit complex to implement, as they require a separate document to define the frame layout as well as individual documents to actually occupy the frames. This section describes the pieces of the defining document, the frameset, and shows you how to create a frame-based layout.

Creating a frameset

A frameset is created like any other HTML document except that its content is limited to frame-related tags. The following skeletal code is an example of a frameset document:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
"http://www.w3.org/TR/html4/frameset.dtd">
```

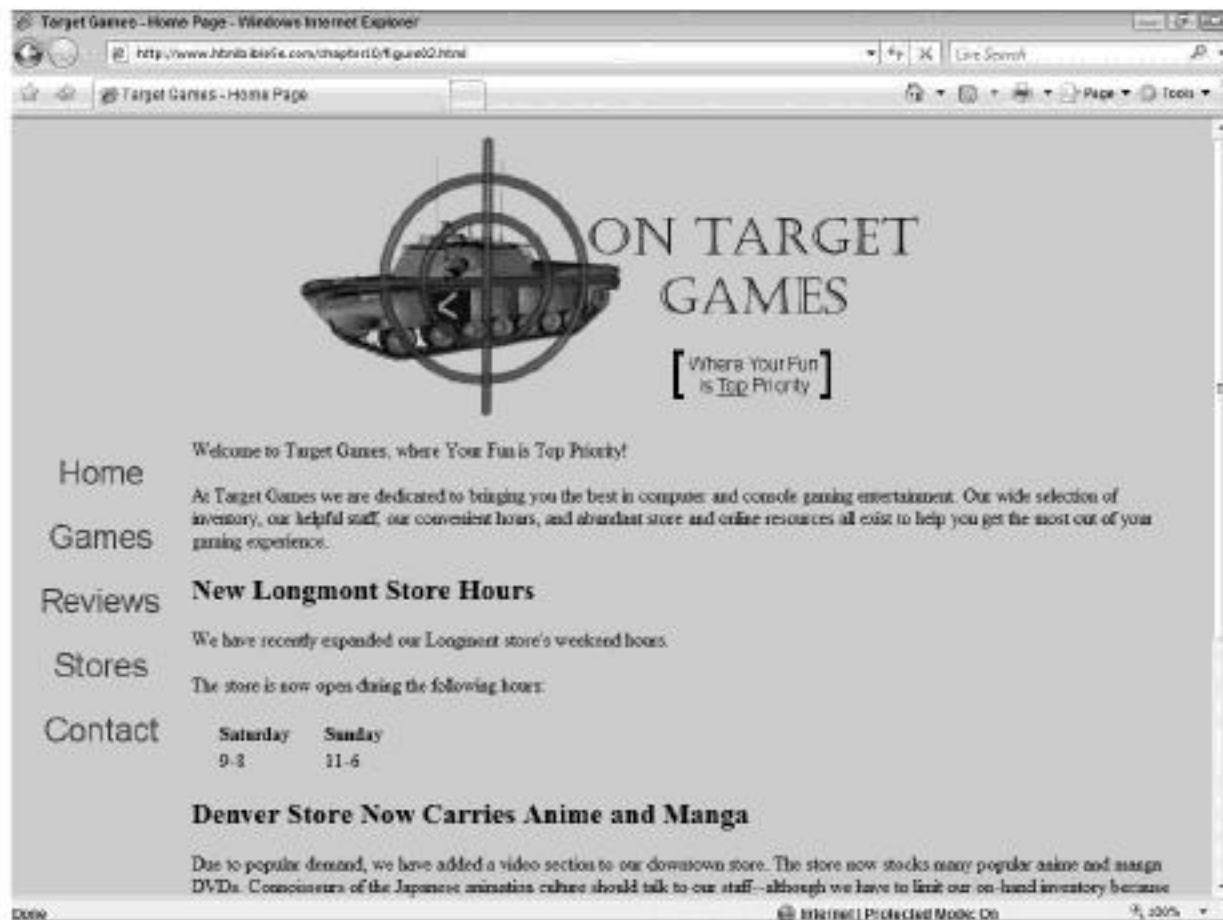
```

<html>
<head>
...
</head>
  <frameset attributes>
    <frame attributes></frame>
    <frame attributes></frame>
    ...
  </frameset>
</html>

```

FIGURE 10-2

A long document uses scroll bars to enable users to see the entire document.



Note the following about this code:

- The document uses the frameset DTD. The frameset DTD is essentially the same as the transitional DTD except for the addition of the frame-specific tags (and replacement of the `<body>` tag, covered shortly).
- There is no `body` element. Instead, the `<frameset>` tag provides the next level container under `<html>`.

Part I: Creating Content with HTML

- The `<frame>` tags, nestled inside the `<frameset>` tag, define the content for the frames and various properties of the frame itself.
- Other than the `<frameset>` and `<head>` sections, there is no other content in the document.

FIGURE 10-3

When the document is scrolled, the entire view, including the navigation bar on the left and the banner graphic on top, is moved.



The basics of the `frameset` and `frame` tags are covered in the next two sections.

The `frameset` tag

The `frameset` tag (`<frameset>`) defines the layout of the frames in the document. It does so by specifying whether the frames should be laid out in columns or rows and what each column's width should be.

The `frameset` tag has the following format:

```
<frameset cols|rows = "column_or_row_size(s)">
```

The column or row sizes can be specified as percentages of the user agent window; pixels; or an asterisk (*), which enables the user agent to assign the size. In the last case, the user agent

typically splits the remaining space across the columns or rows that specify * as their width. In any case, the resulting frameset will occupy the entire user agent window. The number of entries of the `cols` or `rows` attribute also defines how many frames will be used — each entry needs a corresponding `<frame>` tag within the `<frameset>` tag.

FIGURE 10-4

Frames enable one region to scroll while others remain static.



For example, consider these definitions:

```
<!-- Two columns, 25% of the window, the other
      75% of the window -->
<frameset cols = "25%, 75%">
<!-- Two columns, 25% of the window, the other
      75% of the window -->
<frameset cols = "25%, *">
<!-- Three rows, the first 50% of the window, the other
      two 25% of the window each -->
<frameset rows = "50%, *, *">
<!-- Two rows, the first 100 pixels high, the second is the
      size of the remaining window space -->
<frameset rows = "100px, 200px">
```


Note

In the last frameset example, the second row is defined at 200px. However, if the user agent's window is larger than 300 pixels high (the total of the rows defined), the second row will be expanded to fill the space. ■

The frame tag

While the frameset tag (<frameset>) is responsible for defining the layout of the entire page (in terms of number of frames and their size), the frame tag (<frame>) is responsible for defining properties of each frame.

The frame tag has the following minimal syntax:

```
<frame name="name_of_frame" src="url_of_content"></frame>
```

The `name` attribute gives the frame a unique name that can be referenced by URLs, scripts, and so on to control the frame's contents. The `src` attribute is used to specify the URL of the content the frame should display.

Using only these two attributes results in a frame with minimal margins, no borders, and automatic scroll bars. More information on controlling these frame attributes is covered in the next few sections.

Frame margins, borders, and scroll bars

The frame tag supports the additional attributes shown in Table 10-1.

TABLE 10-1

Frame Tag Attributes

| Attribute | Value(s) | Definition |
|--------------|---------------------------------------|--|
| frameborder | 0 - no border (default) 1 - border | Indicates whether the frame has a border or not |
| longdesc | url | A document's URL to use as a long description for the frame (note that this is largely unsupported by user agents) |
| marginheight | pixels | Sets the top and bottom margins for the frame — the distance of the frame's content from its border |
| marginwidth | pixels | Sets the left and right margins for the frame — the distance of the frame's content from its border |
| scrolling | yes no auto (default) | Controls whether the frame displays scroll bars to help scroll the content displayed in the frame |

As mentioned in Table 10-1, the `longdesc` attribute is not fully supported by most user agents. Use it if you need to specify a long description, but don't count on its functionality.

The margin attributes, `marginheight` and `marginwidth`, are self-explanatory, controlling the inside margin of the frame. They should be used to provide enough white space around the frame's content to help make the content clear.

Tip

When using images in a frame, consider setting the margins to zero so the graphic fills the frame entirely without superfluous white space. ■

The `frameborder` attribute controls whether or not the bounding border of the frame is visible. Figure 10-5 shows a frameset without borders, and Figure 10-6 shows the same frameset with borders.

Note

As of this writing, the latest crop of browsers (including the latest versions of Microsoft Internet Explorer and Firefox) display a white border for each frame, despite the `frameborder` setting. If `frameborder` is set to 1, the border appears as a 3-D, stylized bar, as shown in Figure 10-6. However, setting `frameborder` to 0 does not totally eradicate the border as expected. One, non-standards-compliant solution to remove the border entirely is to place the attribute `border="0"` in the `frameset` tag. ■

FIGURE 10-5

Without borders, the frame divisions are hard to distinguish, which may work well for a seamless page design.

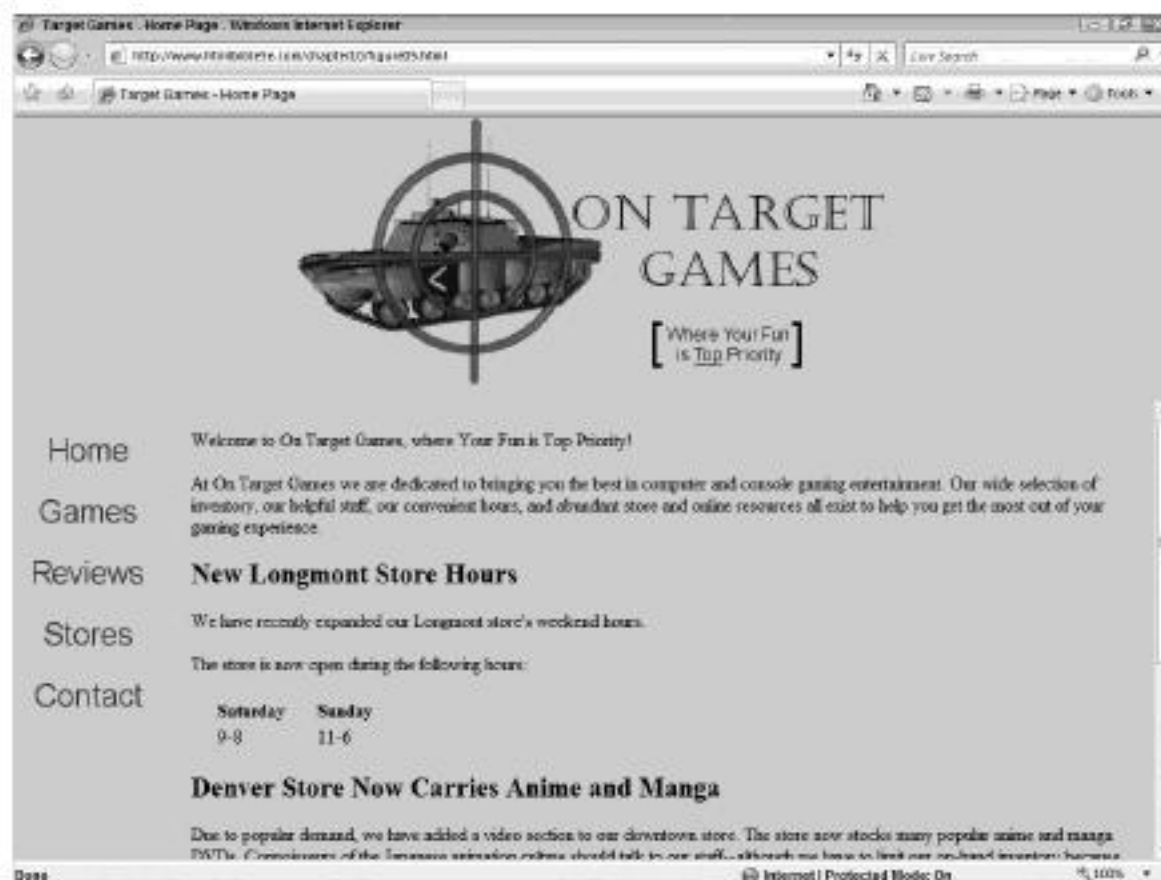
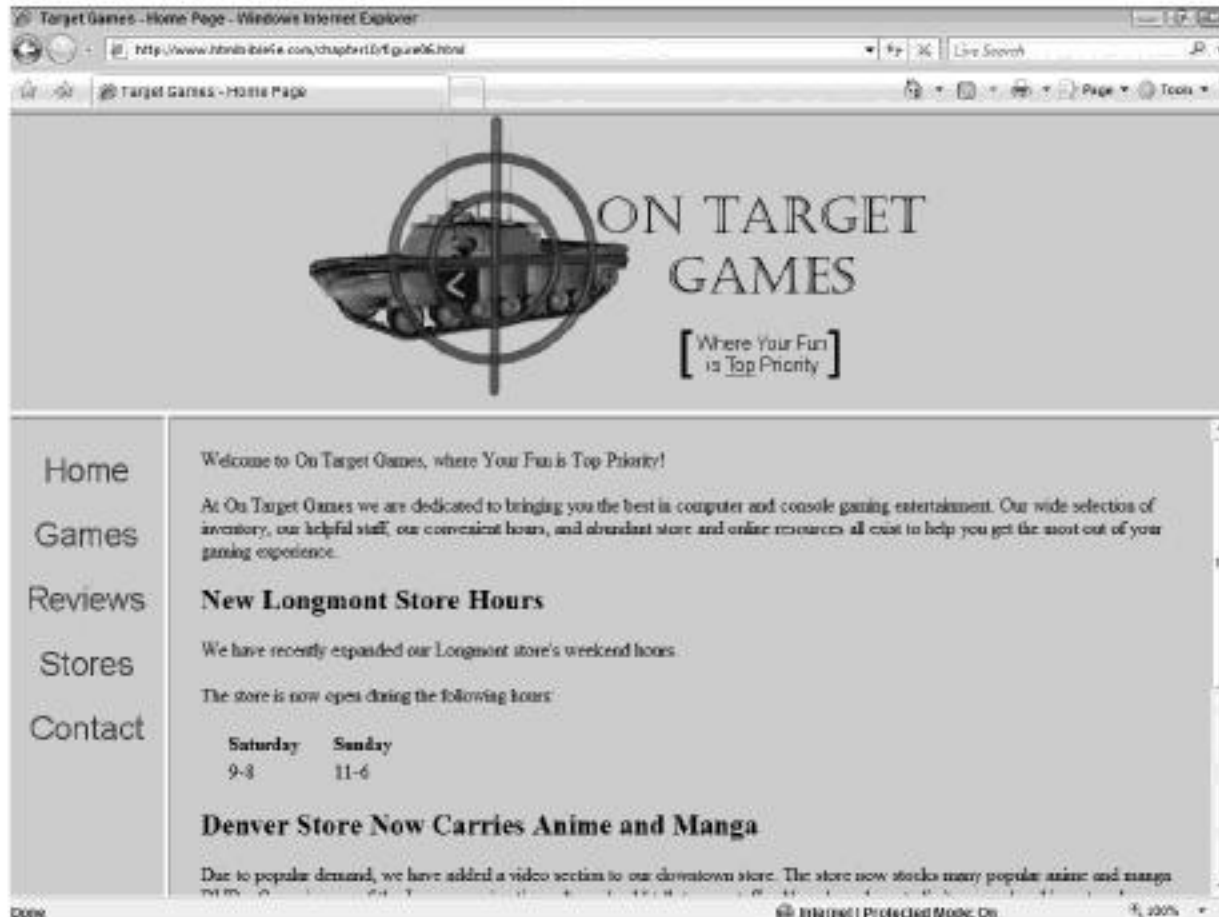


FIGURE 10-6

Frame borders can help users understand the layout of your document and where the edges of each frame are so they can better manipulate them.



The `scrolling` attribute controls whether the frame will display scroll bars. The default setting, `auto`, allows the user agent to decide. If the frame contains too much content to be displayed, the user agent will add scroll bars; if the content fits within the frame, the user agent will not display scroll bars. Use the `scrolling` attribute accordingly — if you want scroll bars all the time, or don't want scroll bars regardless of how the frame's content displays.

Permitting or prohibiting user modifications

The frame tag also has a `noresize` attribute that, when set, will not allow a user to modify the frame's size. The default is to allow the user to resize the frame.

To resize a frame, you position the pointer over the frame division and drag the border. Figures 10-7 and 10-8 show the left frame being enlarged. As a consequence, the right frame shrinks to compensate.

FIGURE 10-7

To resize a frame, position the pointer over the frame border until a double-headed arrow cursor appears.



Targeting Links to Frames

To change a frame's content, you must be able to target a frame. To do so, you use the name attribute to uniquely identify your frames. You can then use those names in scripts and anchor tags to direct new content to the frame.

Scripting languages can use the document's frame collection to target a frame. For example, JavaScript can reference the content of a frame named `news` by changing the value of the following property:

```
parent.news.location.href
```

For example, to fill the news frame with the content of `www.yahoo.com`, a script could use the following statement:

```
parent.news.location.href = "http://www.yahoo.com";
```

Part I: Creating Content with HTML

FIGURE 10-8

Dragging the cursor resizes the frames accordingly.



You can use similar methods and properties to otherwise manipulate the frame content and properties.

Cross-Ref

For more information on JavaScript and how it can be used to affect a document's properties, see Chapters 16 and 17. ■

When you use the frameset DTD, the anchor tag (`<a>`) supports the `target` attribute, which can be used to target a frame for content. The `target` attribute supports the values shown in Table 10-2.

Note

To understand the difference between the `target` attribute's `_parent` and `_top` values, you must understand nested frames, which are covered in the next section. ■

The easiest way to direct content to a frame is to use the frame's name in the `target` attribute of an anchor. This technique is often used to control one frame independently from another,

especially where one frame has a navigation control and the other displays variable content. For example, the following code provides a handful of navigation links in the left (menu) frame, and the content is displayed in the right (content) frame. Each button in the menu frame is wrapped in an appropriate anchor that specifies the content frame as the destination for the URL to which it links:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
"http://www.w3.org/TR/html4/frameset.dtd">
<html>
<head>
  <title>On Target Games - Menu</title>
  <style type="text/css">
    p { font: Arial;
        font-size: 24pt;
        color: blue; }
  </style>
</head>
<body style="background-color: #CBCC66;">
<table border="0" width="100%">
<colgroup>
<col style="text-align: center;"></col>
</colgroup>
<tr><td><a href="home.html" target="content"><p>Home</p></a></td></tr>
<tr><td><a href="games.html" target="content"><p>Games</p></a></td></tr>
<tr><td><a href="reviews.html" target="content"><p>Reviews</p></a></td></tr>
<tr><td><a href="stores.html" target="content"><p>Stores</p></a></td></tr>
<tr><td><a href="contact.html" target="content"><p>Contact</p></a></td></tr>
</table>
</body>
</html>
```

TABLE 10-2

Target Attribute Values

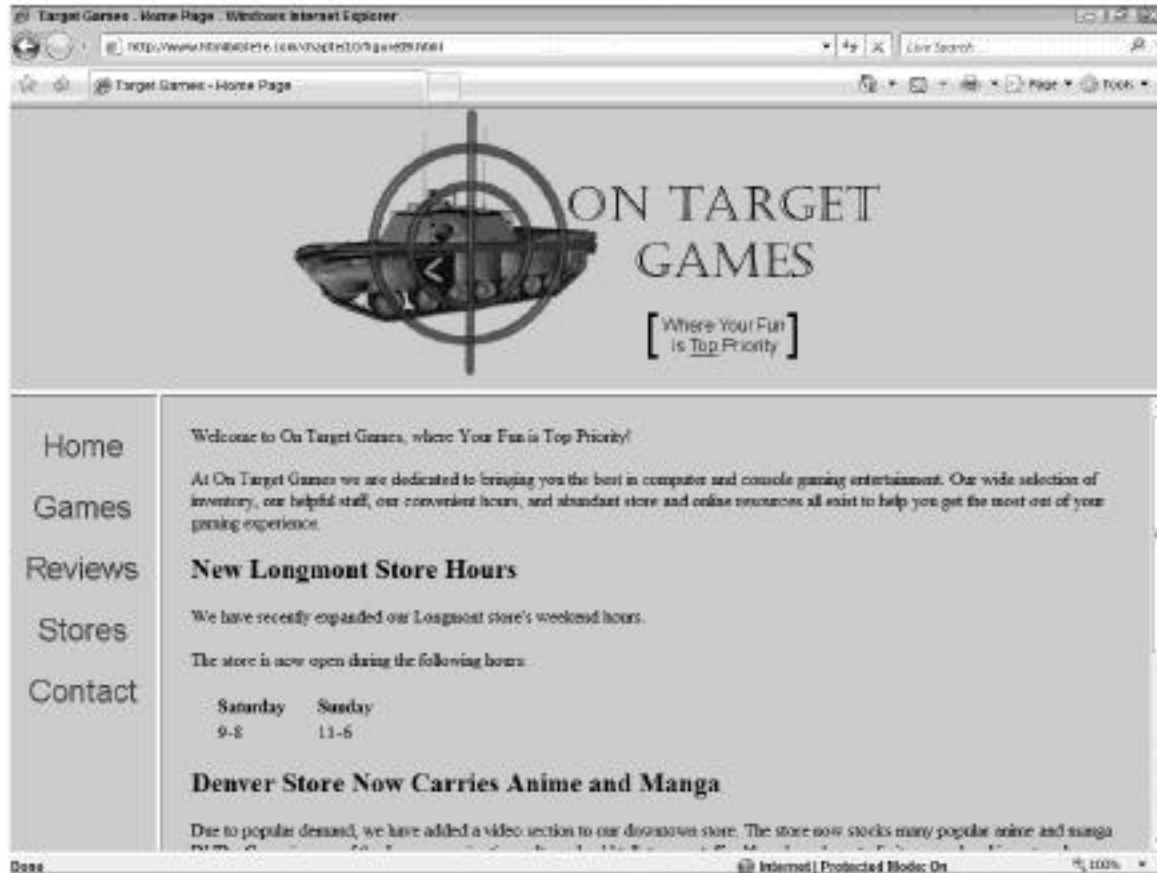
| Value | Definition |
|------------|--|
| frame_name | Displays the content in the frame specified by frame_name |
| _blank | Opens a new window to display the content |
| _parent | Displays the content in the parent frameset of the current frame |
| _self | Displays the content in the current frame |
| _top | Displays the content in the current window, without frames |

Note that each anchor specifies a different document, and that the document specified should be loaded into the content frame via the target attribute. Figure 10-9 shows what this code looks like in a browser; notice the menu on the left edge of the window.

Part I: Creating Content with HTML

FIGURE 10-9

In this simple frame-based navigation scheme, when the user clicks a link in the menu (left) frame, the content changes in the content (right) frame.



Nested Framesets

You have seen how to create rows and columns using framesets, but what if you want a little of both, as shown in the examples in this chapter (two rows, the second one having two columns)?

In such cases, you need to nest one frameset inside of another. For example, the following frameset code results in the layout shown in the document example used throughout this chapter (as in Figure 10-9, for example):

```
<!-- The master frameset, specifying two rows / -->
<frameset rows="250px,*">
  <!-- The first row, one column / -->
  <frame name="top" src="top.html" marginheight="0px"
    frameborder="0" scrolling="no"></frame>
  <!-- The nested frameset, specifying two columns / -->
  <frameset cols="130px,*">
    <frame name="menu" src="menu.html" frameborder="0"
      scrolling="no"></frame>
```

```
<frame name="content" src="maincontent.html"
      marginwidth="25px" marginheight="25px"
      frameborder="0" scrolling="auto"></frame>
</frameset>
</frameset>
```

To achieve the layout, a column-based frameset is nested inside the second row of the row-based frameset. In essence, the second row of the top frameset becomes its own frameset. You could conceivably nest other framesets within this layout, but using more than two or three frames tends to clutter the document and confuse the user.

Note

The `_parent` and `_top` values of the anchor tag's `target` attribute were mentioned earlier in this chapter. Looking at the example in this section, you can see how those two values would each affect the target.

The `_parent` value causes the content to load within the frameset — that is, the immediate parent of the current frame. For example, using `_parent` in a link within the content frame would cause the specified content to load in the area defined for the column-based frameset.

The `_top` value causes the content to load within the top-most frameset. For example, using `_top` in a link within the content frame causes the specified content to load in the area defined for the row-based frameset, effectively taking up the entire user agent window. ■

Inline Frames

Inline frames were conceived as a method to enable smaller pieces of content to be incorporated in scrollable containers within a larger document. Although you can use regular framesets to create individually scrolling regions, the layout is somewhat hampered by the stringent row and column layout design inherent in framesets.

Figure 10-10 shows a sample inline frame placed in a document. Note that the frame is truly “inline” within the objects around it.

Note

Inline frames are not fully supported by all user agents. Inline frames are safe to use only if you are relatively certain that your entire audience will be using an `inline-frame-compatible` browser to view your documents. If this is not the case, you should stay away from inline frames, or code your documents to offer incompatible browsers an alternative.

If you do decide to use inline frames, keep in mind that, like other frame constructs, your documents will validate against frameset DTDs only. ■

Inline frames are accomplished with the `<iframe>` tag. This tag has the following minimal format:

```
<iframe src="url_of_content"></iframe>
```


Part I: Creating Content with HTML

FIGURE 10-10

Inline frames define separate scrollable regions truly inline within the document.



The inline frame tag has a handful of additional attributes, as shown in Table 10-3.

TABLE 10-3

Inline Frame Tag Attributes

| Attribute | Value(s) | Definition |
|--------------|---------------------------------------|--|
| align | Left right top middle bottom | Alignment of the frame to surrounding text |
| frameborder | 0 = no border 1 = border (default) | Indicates whether the frame has a visible border or not |
| height | pixels % | Height of the frame |
| longdesc | url | URL to a document containing the long description of the frame |
| marginheight | pixels | Size of the internal top and bottom margins of the frame |

| Attribute | Value(s) | Definition |
|-------------|-----------------|---|
| marginwidth | pixels | Size of the internal left and right margins of the frame |
| name | name_ of_ frame | Name of the frame (for use in scripting and otherwise referencing the frame and its properties) |
| scrolling | Yes no auto | Indicates whether the frame has scroll bars or not |
| src | url | URL of the content to display in the frame |
| width | pixels % | The width of the frame |

These attributes function exactly like their frame-based kin. It is recommended that you use as many attributes as possible to clearly specify how your inline frame layout will be rendered.

The following code snippet shows how the inline frame was inserted into the document displayed in Figure 10-10:

content.html

```
...
<p>Welcome to On Target Games, where Your Fun is Top Priority!</p>
<iframe src="newsflash.html" align="right"
  style="margin-left:10px;"></iframe>
<p>At On Target Games we are dedicated to bringing you the best in
computer and console gaming entertainment. Our wide selection of
inventory, our helpful staff, our convenient hours, and abundant
store and online resources all exist to help you get the most out of
your gaming experience.</p>
...
```

newsflash.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
  "http://www.w3.org/TR/html4/frameset.dtd"><html>
<head>
  <title>On Target Games - News Flash!</title>
</head>
<body > <!-- style="background-color: #CBCC66;" /-->
<h3>Gopher Hunt Slips<br />Into Next Year</h3>
<p>Rodent Studios has recently announced that their highly
anticipated game, "Gopher Hunt," will miss the Christmas season and
is now slated for release in early 2008. Gopher Hunt is the
semi-sequel to "Badger Brigade," Rodent Studios smash hit of last
year.</p>
<p>Company president Samuel Perry could not be reached for comment,
but all release dates for the game have been removed from the
company website. As you may recall, "Badger Brigade" is one of the
few titles to which On Target Games awarded 5-stars to and the only
game to stay on the best seller list for a whopping 24 week.</p>
```

```
<p>We hope this slip isn't a sign of bigger problems at RS, but we  
will keep you in the loop.</p>  
</body>  
</html>
```

Summary

This chapter introduced the concept of frames, including the inline frame construct. Using frames or inline frames, you can insert separately scrollable and formatted regions inside a larger document. As with most older HTML technologies, you should take care when choosing to use frames; in many instances, you're better off learning and using CSS instead.

The next chapter covers how to use HTML to collect data via forms. Following that, Chapters 12 and 13 round out our coverage of HTML with images, colors, and multimedia.

Forms

HTML's somewhat humble beginnings were send only; that is, the user could receive data sent from a Web server, but the server could not receive data sent from the user. This was quickly identified as a deficiency of HTML. Because most user agents were being run in graphical environments that included rich user interfaces, creating a similar interface to allow users to submit data back to a server seemed a natural extension.

Today, HTML forms present a complex yet flexible framework to allow users basic controls over data. These controls can be used to provide input back to scripts or to submit data. This chapter delves into the particulars of HTML forms.

Understanding Forms

HTML forms simply place a handful of GUI controls on the user agent screen to allow the user to enter data. The controls can allow text input and selection of predefined options from a list, radio buttons or check boxes, or other standard GUI controls.

After the data is entered into the fields, a special control is used to pass the entered data on to a program that can do something useful with it. Such programs are typically referred to as *form handlers* because they “handle” the form data submitted to the server.

The following code shows a basic HTML form whose output is shown in Figure 11-1:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
```

IN THIS CHAPTER

Understanding Forms

Inserting a Form

Field Labels

Text Input Boxes

Password Input Boxes

Radio Buttons

Check Boxes

List Boxes

Large Text Input

Hidden Fields

Buttons

Images

File Fields

Submit and Reset Buttons

Tab Order and Keyboard
Shortcuts

Preventing Changes

Fieldsets and Legends

Using Events with Forms

Form Scripts and Script
Services

```
<title>A Simple Form</title>
</head>
<body>
<form name="sample" id="sample" action="formhandler.cgi"
method="post">
  <table cellpadding="20">
    <tr><td>
      <!-- Text boxes -->
      <table border="0">
        <tr>
          <td><p><label for="fname">First Name: </label></p></td>
          <td><p><input type="text" name="fname" id="fname"
            size="20" /></p></td>
        </tr><tr>
          <td><p><label for="lname">Last Name: </label></p></td>
          <td><p><input type="text" name="lname" id="lname"
            size="20" /></p></td>
        </tr>
      </table>
      <!-- Text area -->
      <p><label for="address">Address:</label><br />
        <textarea name="address" id="address"
          cols=20 rows=4 /></textarea>
      </p>
      <!-- Password -->
      <table border="0">
        <tr>
          <td><p><label for="password">Password:</label></p></td>
          <td><p><input type="password" name="password" id="password"
            size="20" /></p></td>
        </tr>
      </table>
    </td>
    <td>
      <!-- Select list -->
      <p><label for="products">What gaming consoles do you<br />
        own or are you interested in?</p>
      <select name="consoles[]" id="consoles" multiple="multiple"
        size="4" />
        <option id="PS3">Sony Playstation 3
        <option id="XBOX">XBOX 360
        <option id="NINTENDO">Nintendo Wii
        <option id="OTHER">Other
      </select>
    </td>
  </tr>
  <tr>
    <td colspan="2">
      <!-- Check boxes -->
      <fieldset>
        <legend>Contact me via: </legend>
        <p><input type="checkbox" name="email" id="email"
          Checked />
      </fieldset>
    </td>
  </tr>
</table>
</form>
</body>
</html>
```

```
<label for="email">Email</label><br />  
    <input type="checkbox" name="postal" id="postal" />  
    <label for="postal">Postal Mail</label></p>  
</fieldset>  
</td>  
</tr>  
<tr>  
<td>  
    <!-- Radio buttons -->  
    <fieldset>  
        <legend>How many games do you buy in a year?</legend>  
        <p><input type="radio" name="buy" value="onethree"  
            id="onethree" checked="checked" />  
            <label for="onethree">1-3</label><br />  
        <input type="radio" name="buy" value="fiveten" id="fiveten" />  
            <label for="fiveten">5-10</label><br />  
        <input type="radio" name="buy" value="tenfifteen"  
            id="tenfifteen" />  
            <label for="tenfifteen">10-15</label></p>  
    </fieldset>  
</td>  
<td>  
    <!-- Submit and Reset buttons -->  
    <p>  
        <input type="submit" />&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~  
        <input type="reset" />  
    </p>  
  
    <!-- Generic Button -->  
    <p>  
        <input type="button" name="Leave" value="Leave site!" />  
    </p>  
    <!-- Image -->  
    <input type="image" name="sirveybutton"  
        src="images/SirVeyButton.jpg" />  
    <!-- Hidden field -->  
    <input type="hidden" name="referredby" value="Google" />  
</td>  
</tr>  
</table>  
</form>  
</body>  
</html>
```

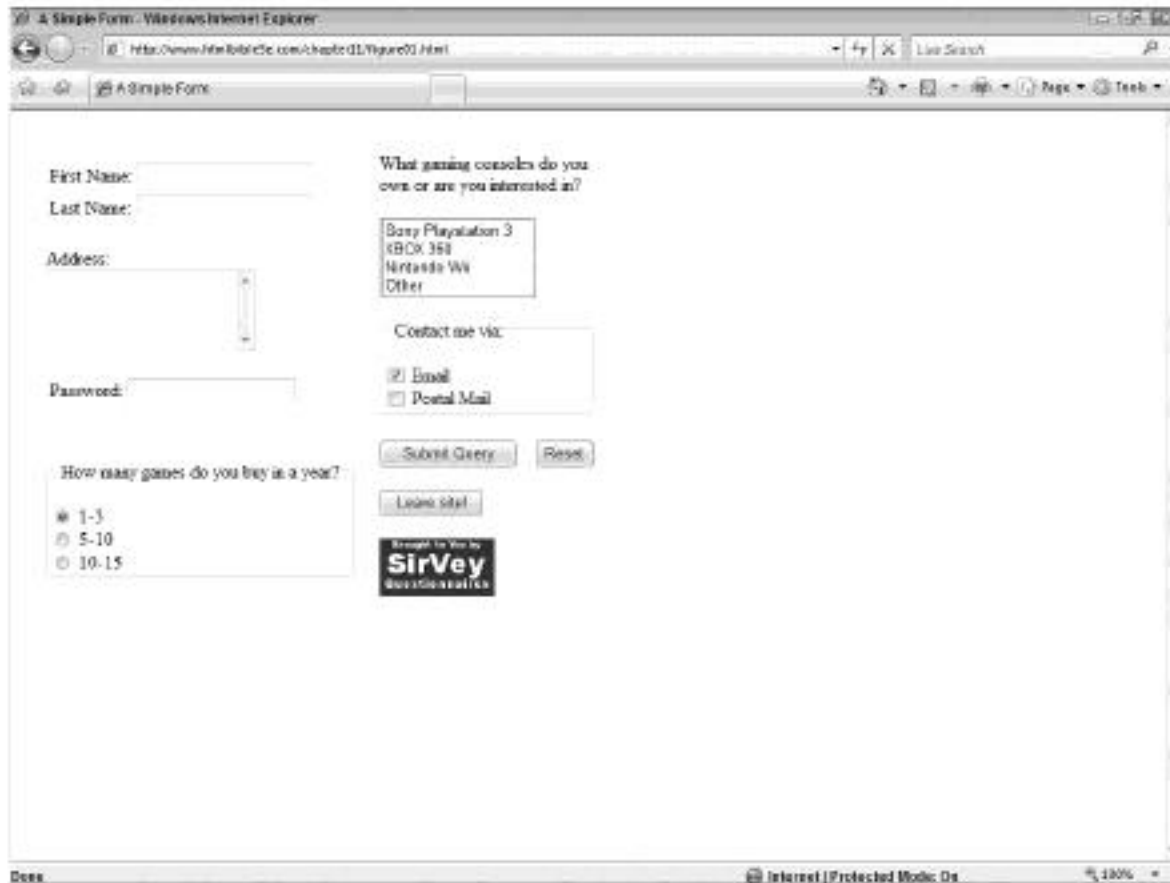
The individual form fields are covered in the following sections.

Note

Many form tags do not have closing tags. However, XML and its variants require that all elements be closed. If you are coding for XML or one of its variants (such as XHTML), be sure to close your tags by including the closing slash (/) at the end of tags that lack a formal closing tag. ■

FIGURE 11-1

A simple HTML form

A screenshot of a web browser window titled "A Simple Form - Windows Internet Explorer". The address bar shows "http://www.htmllib.com/chapter11/figure11.html". The form contains several fields: "First Name:" and "Last Name:" (text boxes), "Address:" (a text box with a vertical scrollbar), "Password:" (a text box), and "What gaming consoles do you own or are you interested in?" (a list box with options: Sony Playstation 3, Xbox 360, Nintendo Wii, and Other). Below the address field is a "Contact me via:" section with radio buttons for "Email" (selected) and "Postal Mail". At the bottom left, there is a question "How many games do you buy in a year?" with three radio button options: "1-3", "5-10", and "10-15". At the bottom right, there are "Submit Query" and "Reset" buttons, and a "Learn More" link. A "SirVey" logo is also present. The browser's status bar at the bottom indicates "Internet | Protected Mode: On" and "100%".

Inserting a Form

You insert a form into your document by placing form fields within `<form>` tags. The entire form or any of the tags can be formatted like any other element in your document, and can be placed within any element capable of holding other elements (paragraphs, tables, and so on).

The `<form>` tag has the following minimum format:

```
<form action="url_to_send_data" method="get/post">
```

The `action` attribute defines a URL where the data from the form should be sent to be handled. Although you can use just about any URL, the destination should be a script or other construct capable of correctly interpreting and doing something useful with the data.

Note

Form actions and form data handlers are covered in the section “Form Scripts and Script Services” later in this chapter. ■

The second attribute, `method`, controls how the data is sent to the handler. The two valid values are `GET` and `POST`. Each value corresponds to the HTTP protocol of the same name.

HTTP GET

The HTTP `GET` protocol attaches data to the actual URL text to pass the data to the destination specified in the `action` attribute. You have probably noticed URLs that resemble the following:

```
http://www.on-target-games.com/forms.cgi?id=45677&data=Taarna
```

The data appears after the question mark and is in name/value pairs. For example, the name `id` has the value of `45677`, and the name `data` has the value of `Taarna`.

Note

In most cases, the name corresponds to field names from the form and may relate to variables in the data handler. ■

Because the data is passed in the text of the URL, it is easy to implement — you can pass data by simply adding appropriate text to the URL used to call the data handler. However, `GET` is also inherently insecure. Never use `GET` to send confidential, unencrypted data to a handler because the data is clearly visible in most user agents and can be easily sniffed by hackers.

HTTP POST

The HTTP `POST` method passes data encoded in the HTTP data stream. As such, it is not typically visible to a user and is therefore a more secure method to pass data, but it can be harder to implement. Thankfully, HTML forms and most other Web technologies make passing data via `POST` a trivial task.

Additional <form> attributes

The `<form>` tag has many additional attributes, which are listed in Table 11-1.

Although you may not need these attributes in simple forms, these attributes can be very useful. The `accept`, `accept-charset`, and `enctype` attributes are invaluable for processing nontextual and international data. The `id` and `name` attributes should be used to uniquely identify a form in your document, especially if you use more than one form in the same document.

Note

Although you can set a field's `id` and `name` to the same value, it's important to understand the use of each. The `id` attribute is used primarily in client-side scripts (like JavaScript) to uniquely identify and manipulate a control. The `name` attribute is used to uniquely reference a field value when a form is passed to a form handler on the server side. ■

TABLE 11-1

<form> Tag Attributes

| Attribute | Values |
|----------------|---|
| accept | A comma-separated list of content types that the handler's server will accept |
| accept-charset | A comma-separated list of character sets the form data may be in |
| enctype | The content type of the form data |
| id | A unique identifier for the form object (replaces the name attribute) |
| name | The name of the form (deprecated, use the id attribute instead) |
| target | Where to open the handler URL (deprecated) |

Field Labels

The <label> tag defines textual labels for form fields. It has the following format:

```
<label for="id_of_related_tag">text_label</label>
```

For example, the following code defines a label for a text box:

```
<p><label for="FirstName">First Name: </label>  
<input type="text" id="FirstName" value="" size="30"  
maxlength="40" /></p>
```

The purpose of the <label> tag is related to accessibility. Most users can rely upon the layout of your forms to determine which labels go with what fields. However, if the user agent does not have a visual component, or if the user is visually impaired, the form's visual layout cannot be relied upon to match labels and fields. The <label> tag's for attribute ensures that the user agent can adequately match labels with fields.

Text Input Boxes

One of the most frequently used fields of HTML forms is the simple text field. This field allows for the input of smaller pieces of text — names, addresses, search terms, and so on.

The text input field tag has the following format:

```
<input type="text" id="id_of_field" value="initial_value"  
size="size_of_field" maxlength="max_characters_allowed" />
```

Although not all of the attributes previously listed are strictly required, they do represent the minimum attributes that you should always use with your text boxes. The following sample text

box is designed to accept a name, appears 30 characters long, accepts a maximum of 40 characters, and has no initial value:

```
<p>Name: <input type="text" id="username" value=""
      size="30" maxlength="40" /></p>
```

The following code example defines a text box to accept an e-mail address. It appears 40 characters wide, accepts only 40 characters, and has an initial value of `info@oasisoftranquility.com`:

```
<p>Email: <input type="text" id="email"
      value="info@oasisoftranquility.com" size="40"
      maxlength="40" /></p>
```

Password Input Boxes

The password input box is similar to the text box but visually obscures data entered into the box by displaying asterisks instead of the actual data entered into the field. The following example displays a password field that accepts 20 characters:

```
<p>Password: <input type="password" id="password" value=""
      size="20" maxlength="20" /></p>
```

The password field accepts the same attributes as the text field.

Caution

The password field only visibly obscures the data onscreen to help stop casual snoops from seeing what a user inputs into a field. It does not encode or in any way obscure the information at the data level. As such, be careful how you use this field. ■

Radio Buttons

Radio buttons are groups of small, round buttons that enable the user to choose one option in each group. The name “radio” button comes from how old-fashioned radios used to be tuned — you pushed one of many buttons to tune to a preset station. When one button was pushed, the rest were reset to the out, or off, position. Like those buttons, form radio buttons are mutually exclusive: Only one of the group can be set. When one is selected, the others in the group are deselected.

The radio button field has the following format:

```
<p><input type="radio" id="control_id" name="group_name" [checked="checked"]
      value="value_if_selected" /> Descriptive Text for Button</p>
```

Note that the `value` attribute defines what value is returned to the handler if the button is selected. This attribute should be unique between buttons in the same group. However, the `name` attribute should be the same for all buttons in a group.

Part I: Creating Content with HTML

The following example code defines a group of radio buttons that enables users to select their gender:

```
<p>Gender:<br />
<input type="radio" id="gender_male" name="gender" value="male" /> Male
<input type="radio" id="gender_female" name="gender" value="female" />
  Female</p>
```

If you want a button selected by default, add the `checked` attribute to the appropriate button's tag. For example, to have "Male" checked by default, you would change the preceding code to the following:

```
<p>Gender:<br />
<input type="radio" id="gender_male" name="gender" value="male"
  checked="checked" /> Male
<input type="radio" id="gender_female" name="gender" value="female" />
  Female</p>
```

It is good form to always set a default button checked in a group of radio buttons.

Tip

XML and its variants do not allow attributes without values. HTML will allow the `checked` attribute to be used with or without a value. To ensure that your code remains as compliant as possible, specify a checked box with the `checked` attribute as `checked="checked"` instead of just `checked`. ■

Check Boxes

Check boxes are small, square boxes used to select non-mutually exclusive options. They are so named because when selected, they display a checkmark (or more commonly an X) in the box like the check boxes in paper lists.

The checkbox field has the following format:

```
<input type="checkbox" id="id_of_field" [checked="checked"]
  value="value_if_selected" />
```

As you can see, other than the mutually exclusive issue, check boxes are very similar in definition to radio buttons. The following example displays a check box that enables users to select whether they want to receive solicitation e-mails:

```
<p><input type="checkbox" id="spam_me" checked="checked" value="spamme" />
Add me to your email list</p>
```

Note that the `checked` attribute can be used to preselect check boxes in your forms. Also, just like radio buttons, the `value` attribute is used as the value of the check box if it is selected. If no value is given, selected check boxes are given the value of "on."

List Boxes

List boxes enable users to pick one or more textual items from a list. The list can be presented in its entirety, with each element visible, or as a pull-down list from which users can scroll to their choices.

List boxes are implemented using `<select>` and `<option>` tags, and optionally the `<optgroup>` tag.

The `<select>` tag provides the container for the list and has the following format:

```
<select id="id_of_field" size="items_to_show" [multiple="multiple"] />
```

The `<option>` tag defines the items for the list. Each item is given its own `<option>` tag. This tag has the optional attributes shown in Table 11-2.

TABLE 11-2

`<option>` Tag Attributes

| Attribute | Values |
|-----------|---|
| label | A shorter label for the item that the user agent can use |
| selected | Indicates that the item should be initially selected |
| value | The value that should be sent to the handler if the item is selected; if this attribute is omitted, the text of the item is sent instead. |

The following is an example of a minimum set of `<option>` tags:

```
<option>Sunday</option>
<option>Monday</option>
<option>Tuesday</option>
<option>Wednesday</option>
<option>Thursday</option>
<option>Friday</option>
<option>Saturday</option>
```

Occasionally, you might want to group options of a list together for clarity. For this you use `<optgroup>` tags, which encapsulate items that should be in that group. For example, the following code defines two groups for the preceding list of options, weekend and weekday:

```
<optgroup label="Weekend">
  <option>Sunday</option>
  <option>Saturday</option>
</optgroup>
<optgroup label="Weekday">
```

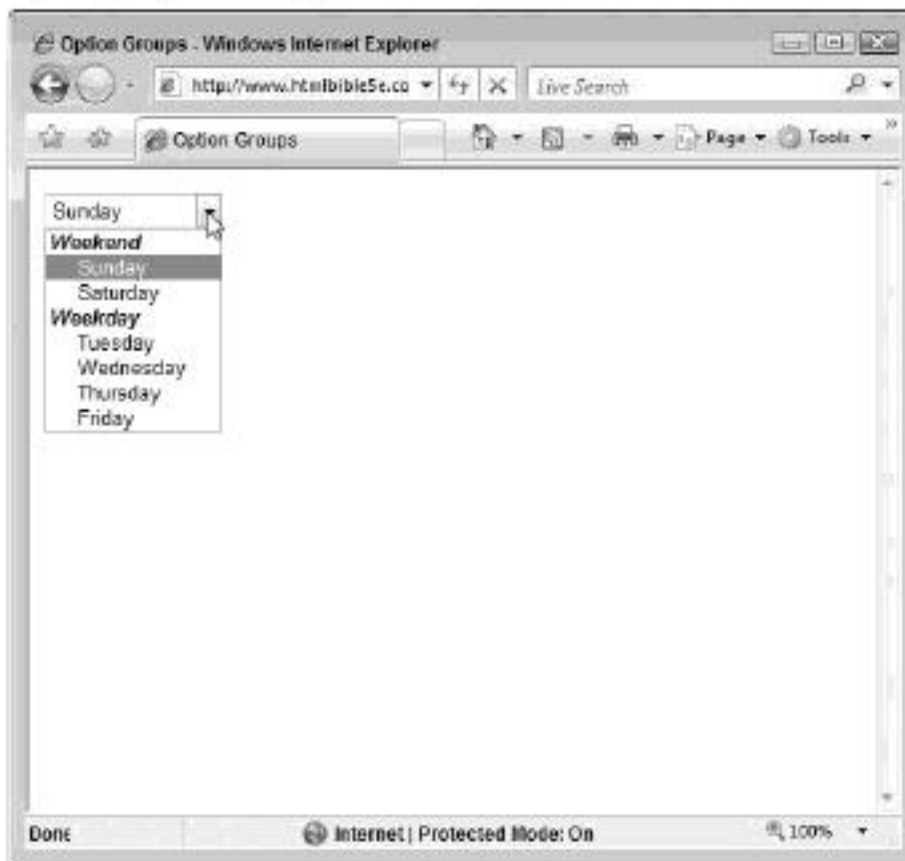
Part I: Creating Content with HTML

```
<option>Monday</option>
<option>Tuesday</option>
<option>Wednesday</option>
<option>Thursday</option>
<option>Friday</option>
</optgroup>
```

Different user agents display option groups differently, but the default behavior is to display the option group labels above the options to which they apply, as shown in Figure 11-2.

FIGURE 11-2

Option groups are displayed in the list as nonselectable items.



The code to combine all three tags to create a list would resemble the following:

```
<p>Select the days you are available:
<select id="AvailDays[]" name="AvailDays[]" size="5"
multiple="multiple">
  <optgroup label="Weekend">
    <option>Sunday</option>
    <option>Saturday</option>
  </optgroup>
  <optgroup label="Weekday">
```

```
<option>Monday</option>
<option>Tuesday</option>
<option>Wednesday</option>
<option>Thursday</option>
<option>Friday</option>
</optgroup>
</select>
</p>
```

Note

Notice the brackets ([]) after the select field's ID. These brackets are used because some languages that form handlers are written in require notification if a field will contain multiple items. In the preceding case, the select field has included the `multiple` attribute, so the field can indeed return multiple values. The brackets signal the form handler to expect this, whether multiple values are actually returned or not.

As of this writing, only handlers written in PHP are known to require this convention. However, the use of brackets will not harm handlers written in other languages, so it's a good habit to adopt. ■

Large Text Input

For large pieces of text, you can use the `<textarea>` tag. This tag can accept textual input of up to 1,024 characters and uses a multiline text box for input.

The `<textarea>` tag has the following format:

```
<textarea id="id_of_field" name="name_of_field" cols="number_of_columns"
rows="number_of_rows"></textarea>
```

Note that the `<textarea>` tag is one of the few form tags that requires both an open tag and a close tag. If you want the field to have default content, the content should be placed between the tags. For example, the following code results in the initial form shown in Figure 11-3:

```
<textarea cols="50" rows="6">
John Doe
123 Main Street
Anywhere, USA
</textarea>
```

Tip

Whatever is placed between the `<textarea>` tags appears verbatim in the text box when the form is first displayed. Therefore, it is important to carefully watch the formatting of your HTML code. For example, if you want the field to be initially blank, you cannot place the open and close tags on separate lines in the code:

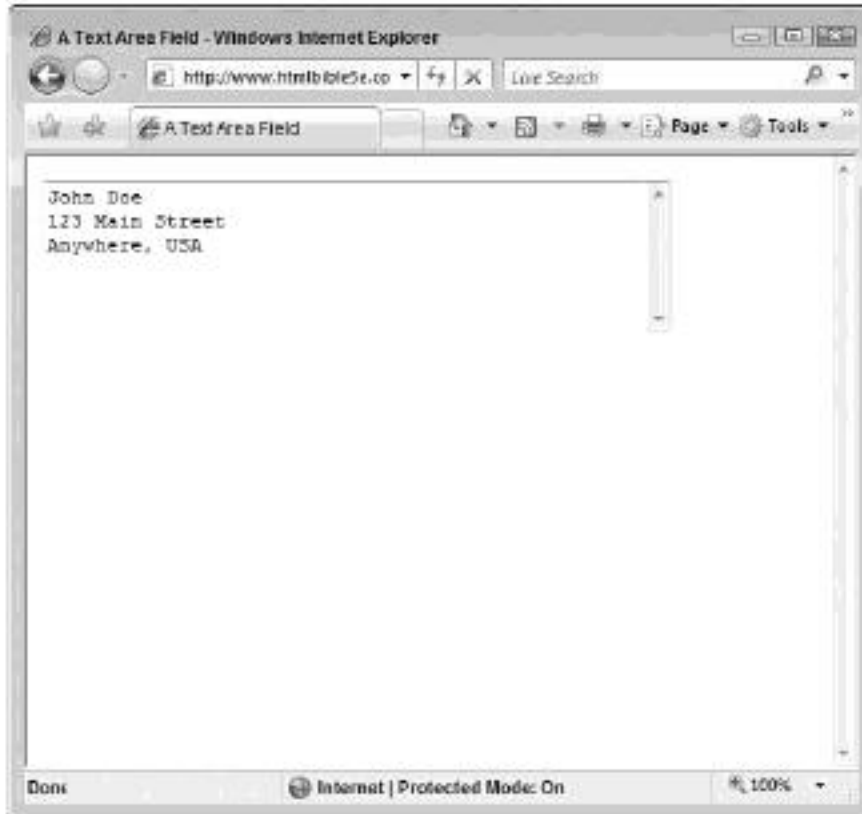
```
<textarea>
</textarea>
```

This would result in the field containing a newline character — it would not be blank. ■

Part I: Creating Content with HTML

FIGURE 11-3

You can set a default value for the `<textarea>` tag by placing content between the open and close tags.



Note that the text entered into the `textarea` field wraps within the width of the box, but the text is sent verbatim to the handler. If the user enters line breaks, then those breaks are also sent to the handler. However, the wrapped text (without hard line breaks) is sent without breaks of any kind.

Note

Previous versions of HTML supported a `wrap` attribute for the `<textarea>` tag. This attribute could be used to control how text wrapped in the text box as well as how it was sent to the handler. Unfortunately, user agent support for this attribute was inconsistent — you could not rely on a browser to follow the intent of the attribute. As such, the attribute has been deprecated and should not be used. ■

Hidden Fields

Hidden fields are used to add data to your form without displaying it to the user. The hidden field has the following format:

```
<input type="hidden" id="id_of_field" name="name_of_field"
value="value_of_field" />
```

Hidden fields are used mostly for tracking data, and other than not being visibly displayed, are like any other field. For example, in a multipage form, a `userid` field can be hidden in the form to ensure that subsequent forms, when submitted, are tied to the same user data.

Keep in mind that hidden fields do not display on the user agent but are still visible in the document's code. As such, hidden fields should never be used for sensitive data.

Buttons

Occasionally, you might need additional, custom buttons on your form. For those cases, you can use the button field, which has the following format:

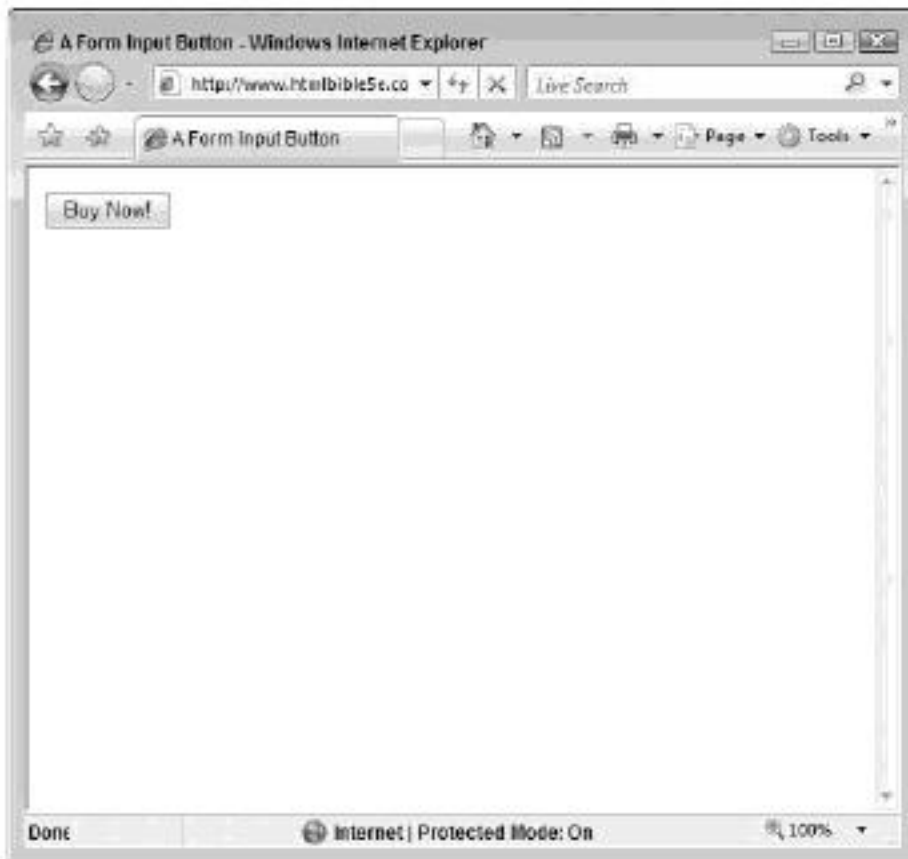
```
<input type="button" id="id_of_field" name="name_of_field"
value="text_for_button" />
```

This tag results in a standard graphical button being displayed on the form. The following code example results in the button shown in Figure 11-4:

```
<input type="button" id="BuyNow" name="buy_button" value="Buy Now!" />
```

FIGURE 11-4

You can use the button field to add custom buttons to your form.



Part I: Creating Content with HTML

Buttons by themselves, however, are fairly useless on a form. To have the button actually do something, you must link it to a script via `onclick` or other event attributes. For example, the following code results in a button that, when clicked, runs the script "buynow":

```
<input type="button" id="BuyNow" name="buy_button" value="Buy Now!"
onClick="buynow();" />
```

Cross-Ref

For more information on `onclick` and other form field event handlers, see the section "Using Events with Forms" later in this chapter. You can also refer to Chapters 16 and 17. ■

Images

Images provide a graphical means to convey a message. Using the image type of the `<input>` tag, you can add images to your form, images that can be used along with other form elements to gather data. The image field has the following format:

```
<input type="image" id="id_of_field" name="name_of_field"
src="url_to_image_file" />
```

The image type of the `<input>` tag also serves as a submit button, giving you the option of easily providing a graphical button. Simply put, if you include an image type `<input>` tag in your form and click the resulting image, it will behave like a Submit button (and likely submit the form).

Note

Submit buttons are covered later in this chapter. ■

However, like the button field, image fields by themselves do not provide any actual form controls. To use the image for input purposes beyond submitting the form, it must be linked to a script. The following example causes the image `buynow.jpg` to be displayed on a form. When the image is clicked, the script `buynow` is run.

```
<input type="image" id="BuyNow" name="BuyNow_graphic" src="buynow.jpg"
onClick="buynow();" />
```

File Fields

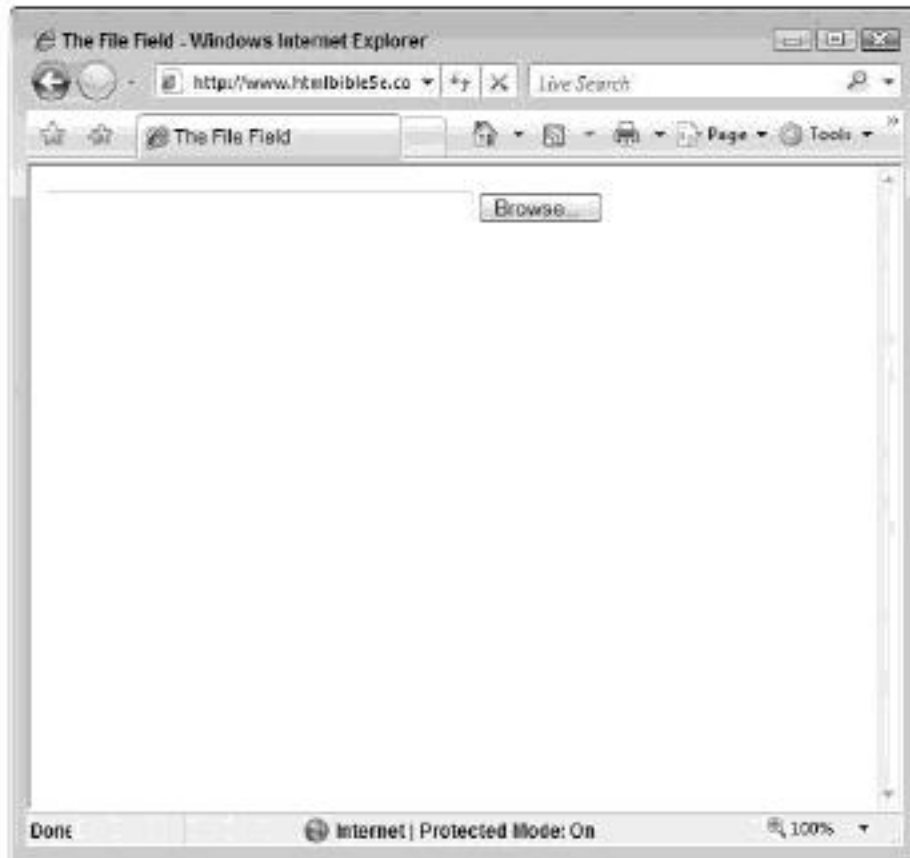
File fields enable users to browse for a local file and send it as an attachment to the form data. The file field has the following format:

```
<input type="file" id="id_of_field" name="name_of_field"
size="display_size_of_field" />
```


The file field results in a text box with a button that enables users to browse for a file using their platform's file browser. Alternately, users can simply type the path and name of the file in the text box. Figure 11-5 shows an example of a file field in Internet Explorer.

FIGURE 11-5

The file field enables users to send a local file.



However, in order to use this control in your forms you must do the following:

- Specify your form as multipart, which allows the file to be attached to the rest of the data.
- Use the POST, not the GET, method of form delivery.

This means your `<form>` tag should resemble the following:

```
<form action="formhandler.cgi" method="post"
  enctype="form/multipart">
```

The form handler you send your form's data to must also be multipart-aware to be able to handle the data sent to it.

Submit and Reset Buttons

Submit and Reset buttons provide control mechanisms for users to submit the data entered to a handler and reset the form to its default state. These buttons have the following format:

Submit button

```
<input type="submit" id="id_of_field" name="name_of_field"  
[value="text_for_button"] />
```

Reset button

```
<input type="reset" id="id_of_field" name="name_of_field"  
[value="text_for_button"] />
```

The `value` attribute for both tags is optional — if this attribute is omitted, the buttons will display default text (usually `Submit` and `Reset`, but ultimately determined by the user agent). Note that some user agents use fairly inappropriate text, such as “Submit Query.” It is a good idea to include the `value` attribute and appropriate text of your own.

The `Submit` button, when clicked, causes the form to be submitted to the handler specified in the `<form>` tag’s `action` attribute. Alternately, you can use the `onclick` attribute to call a script to preprocess the form data before it is passed on to the handler.

The `Reset` button, when clicked, causes the form to be reloaded and its fields reset to their default values. You can also use the `onclick` attribute to change the button’s behavior, calling a script instead of reloading the form.

Tip

Use of `onclick` to change the `Reset` button’s behavior is not recommended. Using `onclick` to cause the `Submit` button to run a script for preprocessing is an expected process, but the `Reset` button should always reset the form. If you need a button to perform some other function, use a custom button that is appropriately labeled. ■

Tab Order and Keyboard Shortcuts

Two additional attributes, `tabindex` and `accesskey`, should be used with your form fields to increase their accessibility.

The `tabindex` attribute defines what order the fields are selected in when the user presses the `Tab` key. This attribute takes a numeric argument that specifies the field’s order on the form.

The `accesskey` attribute defines a key the user can press to directly access the field. This attribute takes a single letter as an argument — that letter becomes the key the user can press to directly access the field.

Note

Keys specified in `accesskey` attributes usually require an additional key to be pressed simultaneously with the chosen key. For example, user agents running on Windows require the Alt key to be pressed along with the letter specified by `accesskey`. Other platforms require similar key combinations, which typically follow the GUI interface conventions of the platform. ■

The following example defines a text box that can be accessed by pressing Alt+F on Windows platforms, and is third in the tab order:

```
<p><label for="FirstName"><u>F</u>first Name: </label>
<input type="text" id="FirstName" name="FirstName" value="" tabindex="3"
accesskey="F" size="30" maxlength="40" /></p>
```

Notice the visual cue given to users, clueing them in to the available shortcut key, via underlining the F in the field's label. Although it is not always possible to provide such cues, doing so will greatly improve the usability of your forms.

Preventing Changes

There are two ways to display information in common form fields but not allow users to change the data: by setting the field to read-only or disabled.

You can add the `readonly` attribute to text fields to prevent users from being able to edit the data contained therein.

The `disabled` attribute effectively disables a control (usually graying out the control, consistent with the user agent's platform method of showing disabled controls) so the user cannot use it.

The following code shows examples of both a read-only and a disabled control. The output of this code is shown in Figure 11-6:

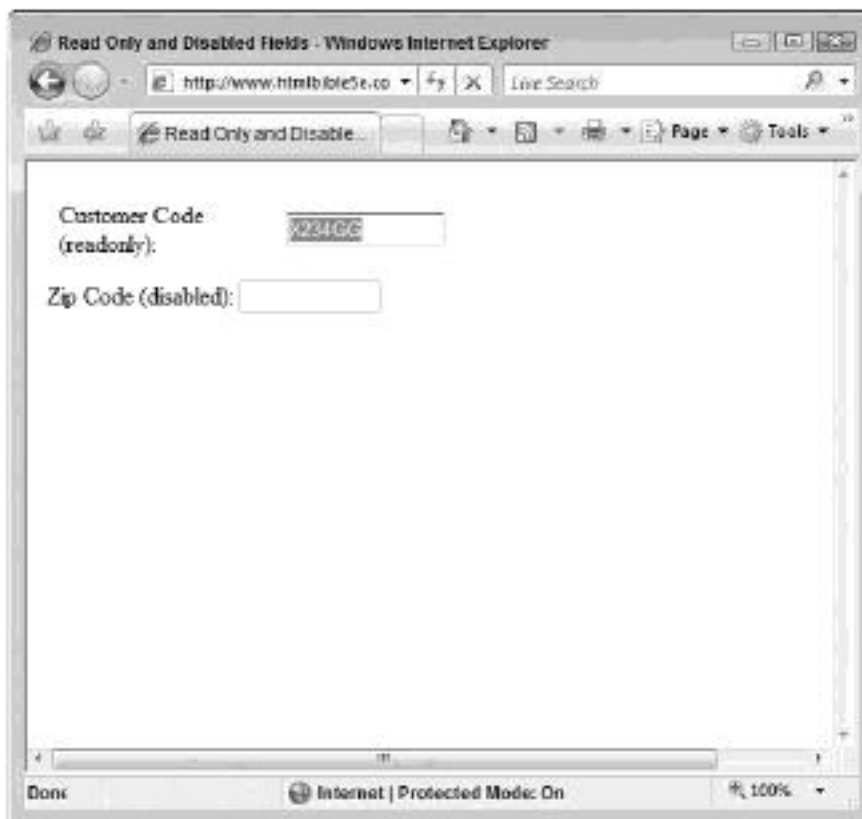
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Read Only and Disabled Fields</title>
</head>
<body>
<form name="sample" id="sample" action="formhandler.cgi"
method="post">
<table cellpadding="10" width="600">
<tr><td width="25%">
<p>Customer Code (readonly):</p>
</td><td>
<input type="text" size="12" value="X234GG"
```

Part I: Creating Content with HTML

```
        readonly="readonly" />
    </td></tr>
</table>
<table>
<tr><td>
<p>Zip Code (disabled):</p>
</td><td>
<input type="text" size="10" value=""
        disabled="disabled" />
</td></tr>
</table>
</form>
</body>
</html>
```

FIGURE 11-6

Disabled and read-only fields can be used to show data without the data being editable.



Although the two attributes make the fields look similar when displayed, the `readonly` field can be selected, but not edited. The `disabled` field cannot be selected at all.

Tip

Disabling a control that is not applicable in certain instances is common practice. For example, international addresses do not have a U.S. zip code. When users indicate that they have an international address, you might decide to disable the zip code field so they do not enter data in it.

You can use client-side scripts to dynamically disable controls. Use the `onblur` or `onchange` actions to call a script from fields that could change the enabled status of other fields — those scripts check the data entered and enable or disable other fields by changing the value of that field's `disabled` attribute. For more information on `onclick` and other form field actions, see the section “Using Events with Forms” later in this chapter, and Chapter 16. ■

Fieldsets and Legends

Sometimes it is advantageous to visually group certain controls on your form. This is a standard practice for graphical user agents, as shown in Figure 11-7, where the dialog controls are separated into distinct sections: Home page, Browsing history, Search, Tabs, and Appearance.

FIGURE 11-7

Grouping controls enables users to better understand a form's organization.



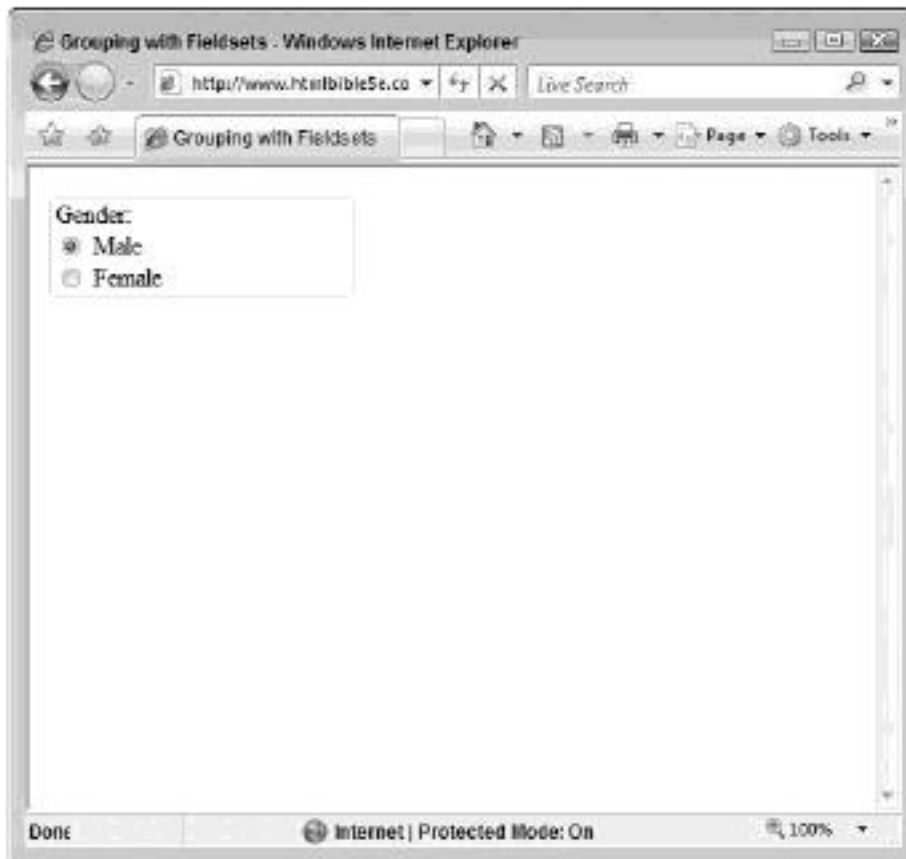
Part I: Creating Content with HTML

The `<fieldset>` tag is used as a container for form elements, and results in a thin border being displayed around the elements it surrounds. For example, the following code results in the output shown in Figure 11-8:

```
<fieldset>
<p>Gender: <br />
<input type="radio" id="gender" value="male" /> Male <br>
<input type="radio" id="gender" value="female" /> Female</p>
</fieldset>
```

FIGURE 11-8

The `<fieldset>` tag can help add organization to your forms.



The `<legend>` tag allows the surrounding `fieldset` box to be captioned. The following code adds a caption to the previous example, the output of which is shown in Figure 11-9:

```
<fieldset>
<p><legend> Gender </legend></p>
<input type="radio" name="gender" value="male" /> Male <br />
<input type="radio" name="gender" value="female" /> Female</p>
</fieldset>
```

FIGURE 11-9

The `<legend>` tag can add captions to your fieldsets.



Using Events with Forms

Another important enhancement to HTML 4 is the addition of events. Events are user or user agent actions that can be captured and acted upon by HTML code. The action is captured via special event attributes added to key tags in your HTML code. These attributes specify the event to watch for and the script to run if the event is encountered.

For example, the `onclick` event can be used to run a script if an element is clicked. This event is particularly handy for button fields in a form, which otherwise perform no action when clicked. For example, the following HTML code specifies that the JavaScript function `addCoupon` should be run when the coupon button is clicked:

```
<input type="button" id="coupon" name="coupon" value="Add Coupon"
      onClick="addCoupon()" />
```

The `addCoupon` script is defined elsewhere — in the document or external script file — and can do any number of things when called. Typically, events and scripts are used to manipulate form data — for example, add shipping costs, validate information entered, dynamically change the form depending on information entered, and more.

Note

Event attributes can be added to any HTML entity. However, they are most useful in conjunction with forms or other dynamic elements. For more information on scripting, see Chapter 16. For more information on dynamic HTML, see Chapter 17. ■

A full list of available event attributes is shown in Table 11-3.

TABLE 11-3

Event Attributes

| Event | Trigger |
|-------------|---|
| onAbort | An element's loading is interrupted. |
| onBlur | An element loses focus. |
| onChange | An element's content is changed. |
| onClick | An element is clicked. |
| onDblclick | An element is double-clicked. |
| onError | An error occurs while loading an element or document. |
| onFocus | An element receives focus. |
| onKeyDown | A key is pressed. |
| onKeyPress | A key is pressed or held down. |
| onKeyUp | A key is released. |
| onLoad | An element or document finishes loading. |
| onMouseDown | A mouse button is pressed. |
| onMouseMove | The mouse pointer is moved. |
| onMouseout | The mouse pointer is moved away from an element. |
| onMouseover | The mouse pointer is moved over an element. |
| onMouseup | A mouse button is released. |
| onReset | The Reset button is clicked. |
| onResize | A window or frame is resized. |
| onSelect | Text in an element is selected. |
| onSubmit | The Submit button is clicked. |
| onunload | The current document is exited or closed. |

The following code shows a sample use of the `onClick` event. When the user clicks the Evaluate button, a script takes the equation in the equation field, evaluates it, and places the value in the results field. A sample run of the process is shown in Figure 11-10.

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
  <title>A Simple Calculator</title>
  <script type="text/JavaScript">
    function eval_eq() {
      // Get the value of the equation field
      var x=document.getElementById("equation").value;
      // Evaluate it (eval) and put the results in
      // the results field
      document.getElementById("results").value=eval(x);
    }
  </script>
</head>
<body>
  <form name="sample" id="sample" action="formhandler.cgi"
    method="post">
    <p>Type an equation into the field below and click the
    Evaluate button to evaluate it. Click Reset to clear
    the form and start over.<br />
    <table border="0" cellpadding="10px">
      <tr style="padding-bottom: 10px;">
        <td><label for="equation">Type your equation here: </label></td>
        <td><input type="text" id="equation" size="40" value="" /></td>
      </tr>
      <tr>
        <td><label for="results">Your results: </label></td>
        <td><input type="text" id="results" size="40" value=""
          disabled="disabled" /></td>
      </tr>
      <tr>
        <td><input type="button" id="evaluate" value="Evaluate"
          onClick="eval_eq();" /></td>
        <td><input type="Reset" id="Reset" /></td>
      </tr>
    </table>
    </p>
  </form>
</body>
</html>

```

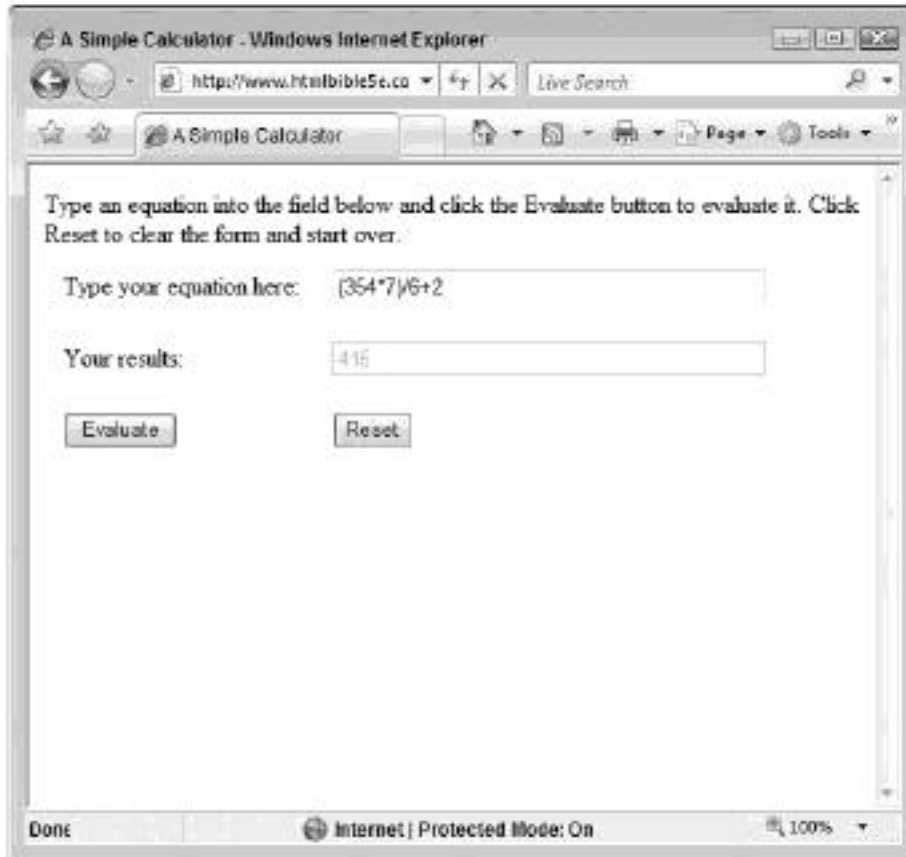
Don't worry if this isn't clear to you at this point. Adding scripting to the HTML mix is a fairly advanced concept. It is introduced here for completeness, but more information on scripting can be found in Chapters 16 and 17.

Tip

If you plan to do a lot of scripting in your HTML documents, you will be best served by picking up a dedicated JavaScript book. Although you may learn the basics here, more advanced script usage is outside the scope of this book. ■

FIGURE 11-10

The `onClick` event enables the button object to perform tasks on other form elements via JavaScript.



Form Scripts and Script Services

As previously mentioned in the section "Understanding Forms," form data is typically passed to a data handler, a script or program that does something useful with the data.

Form handlers typically do one or more of the following actions with the form data:

- Manipulate or verify the data
- E-mail the data
- Store the data in a file or database
- Process the data and return some result

There are many ways to construct a form handler, but the usual method is by using a server-side programming language to create a script that does what you need to the data. Common form handlers are created in Perl, Python, PHP, or another server-side programming language.

Security is an important issue you should consider when creating form handlers. One of the earliest, most popular form handlers, `formmail.cgi`, was found to have a vulnerability that allowed anyone to send data to the script and have it e-mail the data to whomever the sender

wanted. This functionality was an instant hit with e-mail spammers who still use unsecured `formmail` scripts to send anonymous spam.

Because form-handling scripts can be so diverse (performing different functions, written in different languages), it is hard to give tangible examples here. Use a server-side language you are comfortable with to create a form handler that does exactly what you want.

If you want a generic form handler to simply store or e-mail the data, you can choose from a few routes.

Download a handler

Several sites on the Internet offer generic form handlers. One of my favorites is the CGI Resource Index, <http://cgi.resourceindex.com/>. This site has several dozen scripts that you can download and use for your form handling. Keep in mind that most scripts require a processing language such as Perl or PHP — ensure your server has such prerequisites before downloading scripts.

Use a script service

Also available are several services that enable you to process your form data through their server and scripts. You may need such a service if you cannot run scripts on your server or you want a generic, no-hassle solution.

A partial list of script services is available at the CGI Resource Index, <http://cgi.resourceindex.com/>. From the main page, select Remotely Hosted and browse for a service that meets your needs.

Summary

This chapter showed you the particulars of HTML forms. It demonstrated how to include them in your documents and what each form tag can accomplish, and the methods and handlers you might employ to get the most of the data your forms supply.

The next few chapters cover colors, images, and multimedia, and then delve into HTML niche formatting and encoding topics. Chapters 16 and 17 provide basic and then advanced coverage of JavaScript, which can be used for tasks such as automating your forms and documents.

Colors and Images

The Web is not a black-and-white place. In fact, it never has been — the Web and HTML language was born with 16 named colors and blossomed quickly into more than 200 other supported colors. So, although it had its share of growing spurts, unlike most of the other information mediums, the Web didn't have to grow out of a colorless beginning.

This chapter shows how to use colors with fonts, borders, backgrounds, and more. It also covers the image tag, which can be used to insert graphical images into your documents.

Web Color Basics

When the Web was first conceived, most computers were not capable of displaying the multitude of colors possible today. Most computers in that era supported a maximum of 16 colors (via Enhanced Graphics Adapter, or EGA), or a few years later, 256 colors (via Video Graphics Array, or VGA).

To create an initial, standard color palette, the W3C created a color palette of 16 named colors: aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, purple, red, silver, teal, white, and yellow. These color names are still the only color names that will properly validate against HTML 4.

To accommodate colors in elements, several `color` and `bgcolor` attributes were added to the element tags that support color. For example, the

IN THIS CHAPTER

Web Color Basics

Other Means to Specify Colors

The Evolution of Color on the Web

Using Proper Means to Specify Colors

Image Formats for the Web

Creating Graphics

Inserting an Image

Image Alignment

Specifying Text to Display for Nongraphical Browsers

Sizing an Image

Image Borders

Image Maps

following two tags would produce a red background in a document, and text in a white font, respectively:

```
<body bgcolor="red">
<font color="white">
```

Note

It bears mentioning at this point that most of the `color` and `bgcolor` attributes have been deprecated in favor of using styles. As such, you should not use them — they are presented here for completeness and historical context. ■

Other Means to Specify Colors

Besides using assigned names to specify colors, there are several other ways to choose specific colors in your documents, mostly by specifying an exact mix of red, green, and blue that make up the target color.

You can code the mix of colors to create the color you want to use in two main ways: by denoting a hexadecimal number containing the color values to mix, or by using an `rgb()` function, also containing the color values to mix but in decimal format. Either way allows for color values between 0 (no color) and 255 (full color).

The hexadecimal format typically looks as follows:

```
#RRGGBB;
```

The format begins with a pound sign (`#`), has two hexadecimal digits for each color value, and ends with a semicolon (`;`). For the values, the smaller the number, the less the presence of the color in the mix. To create purple, for example (which is equal parts red and blue), you could use code similar to the following:

```
#FF00FF;
```

Lighter shades of purple can be accomplished by lowering the values. Conversely, you can create deeper purple shades by increasing the values.

The second function/decimal value method resembles the following:

```
rgb(rrr,ggg,bbb);
```

In this case, the function begins with `rgb` and encapsulates the color values in parentheses in a comma-separated list. This method also ends with a semicolon. In terms of the preceding example, you would use the following code with `rgb` to define purple:

```
rgb(255,0,255);
```


The Evolution of Color on the Web

Within a few years of the Web's creation, it was clear that more colors, not fewer, were in its future. To answer the call for a standard palette with more colors, the W3C created a so-called "Web-safe" palette of 216 colors, shown in Table 12-1.

TABLE 12-1

The Web-Safe Palette

| | | | | | |
|--------|---------|---------|---------|---------|---------|
| 000000 | #000033 | #000066 | #000099 | #0000CC | #0000FF |
| 003300 | #003333 | #003366 | #003399 | #0033CC | #0033FF |
| 006600 | #006633 | #006666 | #006699 | #0066CC | #0066FF |
| 009900 | #009933 | #009966 | #009999 | #0099CC | #0099FF |
| 00CC00 | #00CC33 | #00CC66 | #00CC99 | #00CCCC | #00CCFF |
| 00FF00 | #00FF33 | #00FF66 | #00FF99 | #00FFCC | #00FFFF |
| 330000 | #330033 | #330066 | #330099 | #3300CC | #3300FF |
| 333300 | #333333 | #333366 | #333399 | #3333CC | #3333FF |
| 336600 | #336633 | #336666 | #336699 | #3366CC | #3366FF |
| 339900 | #339933 | #339966 | #339999 | #3399CC | #3399FF |
| 33CC00 | #33CC33 | #33CC66 | #33CC99 | #33CCCC | #33CCFF |
| 33FF00 | #33FF33 | #33FF66 | #33FF99 | #33FFCC | #33FFFF |
| 660000 | #660033 | #660066 | #660099 | #6600CC | #6600FF |
| 663300 | #663333 | #663366 | #663399 | #6633CC | #6633FF |
| 666600 | #666633 | #666666 | #666699 | #6666CC | #6666FF |
| 669900 | #669933 | #669966 | #669999 | #6699CC | #6699FF |
| 66CC00 | #66CC33 | #66CC66 | #66CC99 | #66CCCC | #66CCFF |
| 66FF00 | #66FF33 | #66FF66 | #66FF99 | #66FFCC | #66FFFF |
| 990000 | #990033 | #990066 | #990099 | #9900CC | #9900FF |
| 993300 | #993333 | #993366 | #993399 | #9933CC | #9933FF |
| 996600 | #996633 | #996666 | #996699 | #9966CC | #9966FF |
| 999900 | #999933 | #999966 | #999999 | #9999CC | #9999FF |

continued

TABLE 12-1 *(continued)*

| | | | | | |
|--------|---------|---------|---------|---------|---------|
| 99CC00 | #99CC33 | #99CC66 | #99CC99 | #99CCCC | #99CCFF |
| 99FF00 | #99FF33 | #99FF66 | #99FF99 | #99FFCC | #99FFFF |
| CC0000 | #CC0033 | #CC0066 | #CC0099 | #CC00CC | CC00FF |
| CC3300 | #CC3333 | #CC3366 | #CC3399 | #CC33CC | #CC33FF |
| CC6600 | #CC6633 | #CC6666 | #CC6699 | #CC66CC | #CC66FF |
| CC9900 | #CC9933 | #CC9966 | #CC9999 | #CC99CC | #CC99FF |
| CCCC00 | #CCCC33 | #CCCC66 | #CCCC99 | #CCCCCC | #CCCCFF |
| CCFF00 | #CCFF33 | #CCFF66 | #CCFF99 | #CCFFCC | #CCFFFF |
| FF0000 | #FF0033 | #FF0066 | #FF0099 | #FF00CC | #FF00FF |
| FF3300 | #FF3333 | #FF3366 | #FF3399 | #FF33CC | #FF33FF |
| FF6600 | #FF6633 | #FF6666 | #FF6699 | #FF66CC | #FF66FF |
| FF9900 | #FF9933 | #FF9966 | #FF9999 | #FF99CC | #FF99FF |
| FFCC00 | #FFCC33 | #FFCC66 | #FFCC99 | #FFCCCC | #FFCCFF |
| FFFF00 | #FFFF33 | #FFFF66 | #FFFF99 | #FFFFCC | #FFFFFF |

This palette was chosen because it represented the 216 colors that were common between the PC and MAC platforms — each having its own set of 40 or so reserved colors different from the other platform.

In today's computing environments, most user agents on most platforms are capable of producing or approximating several million colors. However, you should still be aware of your audience and their ability to see a particular color before designing your page to use that color prominently.

In addition to the original 16 named colors, a palette of 150 additional named colors should be recognized by most browsers. Keep in mind, however, that using the names of the colors in this palette will cause your documents not to validate against the HTML 4 standard. If you need your document to validate, stick with the names in the 16-color palette or use the hex codes shown next to the names in Table 12-2.

Tip

Most image-editing programs include a hexadecimal value field in their color picker or palette dialog. Using that feature you can easily sample colors from graphics, obtain their hexadecimal value, and include that value in your styles and other color codes within your HTML documents. ■

TABLE 12-2

Extended Named Color Palette

| Name | Hex Value | Name | Hex Value |
|----------------|-----------|----------------|-----------|
| AliceBlue | #F0F8FF | DarkMagenta | #8B008B |
| AntiqueWhite | #FAEBD7 | DarkOliveGreen | #556B2F |
| Aqua | #00FFFF | DarkOrange | #FF8C00 |
| Aquamarine | #7FFFD4 | DarkOrchid | #9932CC |
| Azure | #F0FFFF | DarkRed | #8B0000 |
| Beige | #F5F5DC | DarkSalmon | #E9967A |
| Bisque | #FFE4C4 | DarkSeaGreen | #8FBC8F |
| Black | #000000 | DarkSlateBlue | #483D8B |
| BlanchedAlmond | #FFEBCD | DarkSlateGray | #2F4F4F |
| Blue | #0000FF | DarkSlateGrey | #2F4F4F |
| BlueViolet | #8A2BE2 | DarkTurquoise | #00CED1 |
| Brown | #A52A2A | DarkViolet | #9400D3 |
| BurlyWood | #DEB887 | DeepPink | #FF1493 |
| CadetBlue | #5F9EA0 | DeepSkyBlue | #00BFFF |
| Chartreuse | #7FFF00 | DimGray | #696969 |
| Chocolate | #D2691E | DimGrey | #696969 |
| Coral | #FF7F50 | DodgerBlue | #1E90FF |
| CornflowerBlue | #6495ED | FireBrick | #B22222 |
| Cornsilk | #FFF8DC | FloralWhite | #FFFAF0 |
| Crimson | #DC143C | ForestGreen | #228B22 |
| Cyan | #00FFFF | Fuchsia | #FF00FF |
| DarkBlue | #00008B | Gainsboro | #DCDCDC |
| DarkCyan | #008B8B | GhostWhite | #F8F8FF |
| DarkGoldenRod | #B8860B | Gold | #FFD700 |
| DarkGray | #A9A9A9 | GoldenRod | #DAA520 |
| DarkGrey | #A9A9A9 | Gray | #808080 |
| DarkGreen | #006400 | Grey | #808080 |
| DarkKhaki | #BDB76B | Green | #008000 |

continued

Part I: Creating Content with HTML

TABLE 12-2 (continued)

| Name | Hex Value | Name | Hex Value |
|----------------------|-----------|-------------------|-----------|
| GreenYellow | #ADFF2F | Maroon | #800000 |
| HoneyDew | #F0FFF0 | MediumAquaMarine | #66CDAA |
| HotPink | #FF69B4 | MediumBlue | #0000CD |
| IndianRed | #CD5C5C | MediumOrchid | #BA55D3 |
| Indigo | #4B0082 | MediumPurple | #9370D8 |
| Ivory | #FFFFFF | MediumSeaGreen | #3CB371 |
| Khaki | #F0E68C | MediumSlateBlue | #7B68EE |
| Lavender | #E6E6FA | MediumSpringGreen | #00FA9A |
| LavenderBlush | #FFF0F5 | MediumTurquoise | #48D1CC |
| LawnGreen | #7CFC00 | MediumVioletRed | #C71585 |
| LemonChiffon | #FFFACD | MidnightBlue | #191970 |
| LightBlue | #ADD8E6 | MintCream | #F5FFFA |
| LightCoral | #F08080 | MistyRose | #FFE4E1 |
| LightCyan | #E0FFFF | Moccasin | #FFE4B5 |
| LightGoldenRodYellow | #FAFAD2 | NavajoWhite | #FFDEAD |
| LightGray | #D3D3D3 | Navy | #000080 |
| LightGrey | #D3D3D3 | OldLace | #FDF5E6 |
| LightGreen | #90EE90 | Olive | #808000 |
| LightPink | #FFB6C1 | OliveDrab | #6B8E23 |
| LightSalmon | #FFA07A | Orange | #FFA500 |
| LightSeaGreen | #20B2AA | OrangeRed | #FF4500 |
| LightSkyBlue | #87CEFA | Orchid | #DA70D6 |
| LightSlateGray | #778899 | PaleGoldenRod | #EEE8AA |
| LightSlateGrey | #778899 | PaleGreen | #98FB98 |
| LightSteelBlue | #B0C4DE | PaleTurquoise | #AFEEEE |
| LightYellow | #FFFFE0 | PaleVioletRed | #D87093 |
| Lime | #00FF00 | PapayaWhip | #FFEFD5 |
| LimeGreen | #32CD32 | PeachPuff | #FFDAB9 |
| Linen | #FAF0E6 | Peru | #CD853F |
| Magenta | #FF00FF | Pink | #FFC0CB |

| Name | Hex Value | Name | Hex Value |
|-------------|-----------|-------------|-----------|
| Plum | #DDA0DD | SlateGrey | #708090 |
| PowderBlue | #B0E0E6 | Snow | #FFFAFA |
| Purple | #800080 | SpringGreen | #00FF7F |
| Red | #FF0000 | SteelBlue | #4682B4 |
| RosyBrown | #BC8F8F | Tan | #D2B48C |
| RoyalBlue | #4169E1 | Teal | #008080 |
| SaddleBrown | #8B4513 | Thistle | #D8BFD8 |
| Salmon | #FA8072 | Tomato | #FF6347 |
| SandyBrown | #F4A460 | Turquoise | #40E0D0 |
| SeaGreen | #2E8B57 | Violet | #EE82EE |
| SeaShell | #FFF5EE | Wheat | #F5DEB3 |
| Sienna | #A0522D | White | #FFFFFF |
| Silver | #C0C0C0 | WhiteSmoke | #F5F5F5 |
| SkyBlue | #87CEEB | Yellow | #FFFF00 |
| SlateBlue | #6A5ACD | YellowGreen | #9ACD32 |
| SlateGray | #6A5ACD | | |

Using Proper Means to Specify Colors

As previously mentioned, the old HTML tag attributes of `color` and `bgcolor` have been deprecated. If you use them, your documents will not validate against the HTML 4 standard, and you run the risk that modern user agents will not interpret the attributes correctly.

Instead, you should use styles.

Many different color style properties can be applied to almost any element in HTML. Of course, not all elements support color changes, but you might be surprised to learn how many do.

The most common style properties to change an element's color are `color` and `background-color`. These two properties do what you would expect: change the foreground color of an element and change the background color of an element, respectively.

As an example, the following tag would cause the text within the paragraph to be rendered in yellow:

```
<p style="color: yellow;">
```

FIGURE 12-1

Changing an element's color or background affects the element and possibly its children, but not its parent.



The following tag would cause the entire document to have a green background:

```
<body style="background-color: green;">
```

Many other properties also have color components — that is, arguments that can be used to affect the color of elements to which the property is applied. These include the following:

- background
- border
- border-color
- border-top, right, bottom, left
- border-top, right, bottom, left-color
- outline
- outline-color
- text-shadow

Each of these properties affects the elements to which it is applied in a slightly different manner. Using the color attribute of the text-shadow property, for example, affects only the color of the text shadow, not the element itself, despite how the color attribute might work elsewhere.