for the script to run — the document is displayed for the user to peruse while the script runs in the background.

Controlling Text Breaks in Table Cells

Text in table cells is a fickle beast, especially when filled with long numbers. True text will typically allow for sane breaks, even in a narrow column. However, not much can be done with numeric data or other data that cannot be arbitrarily broken when a column changes size.

For example, consider the following number:

```
1,234,567,890.34
```

How would you break such a number in a column that supports only a four-character width? It's a tough decision, but probably one you would prefer to make yourself, rather than leave it up to HTML and CSS.

Tip

When deciding where to break numbers, keep in mind that the best places to break are around punctuation (e.g., commas or periods) or currency symbols. Doing so will preserve the readability of your numbers as much as possible. ■

You have two essential tools to use when controlling line breaks: the *nonbreaking space* and the *zero-width space*.

The nonbreaking space is best known by its HTML entity code . This resulting character looks and acts like a space but it doesn't allow the browser to break the line at this space. Although it is commonly used to space-fill elements, the nonbreaking space does have its textual uses. For example, you typically would not want the following line broken at the embedded spaces:

```
12344 Mediterranean Circle
```

To prevent the line from breaking, you would replace the spaces with nonbreaking space entities, similar to the following:

```
12344&nbsp;Mediterranean&nbsp;Circle
```

Unlike the nonbreaking space, the zero-width space is not visible but allows the browser to break at the character. The zero-width space can be inserted by using its HTML entity code, `&#8203;`.

Returning to our earlier example number, you can choose where you would like it to be divided if it must break (the original number is on the first line, the doctored number on the second):

```
1,234,567,890.34
1,&#8203;234,&#8203;567,&#8203;890.&#8203;34
```

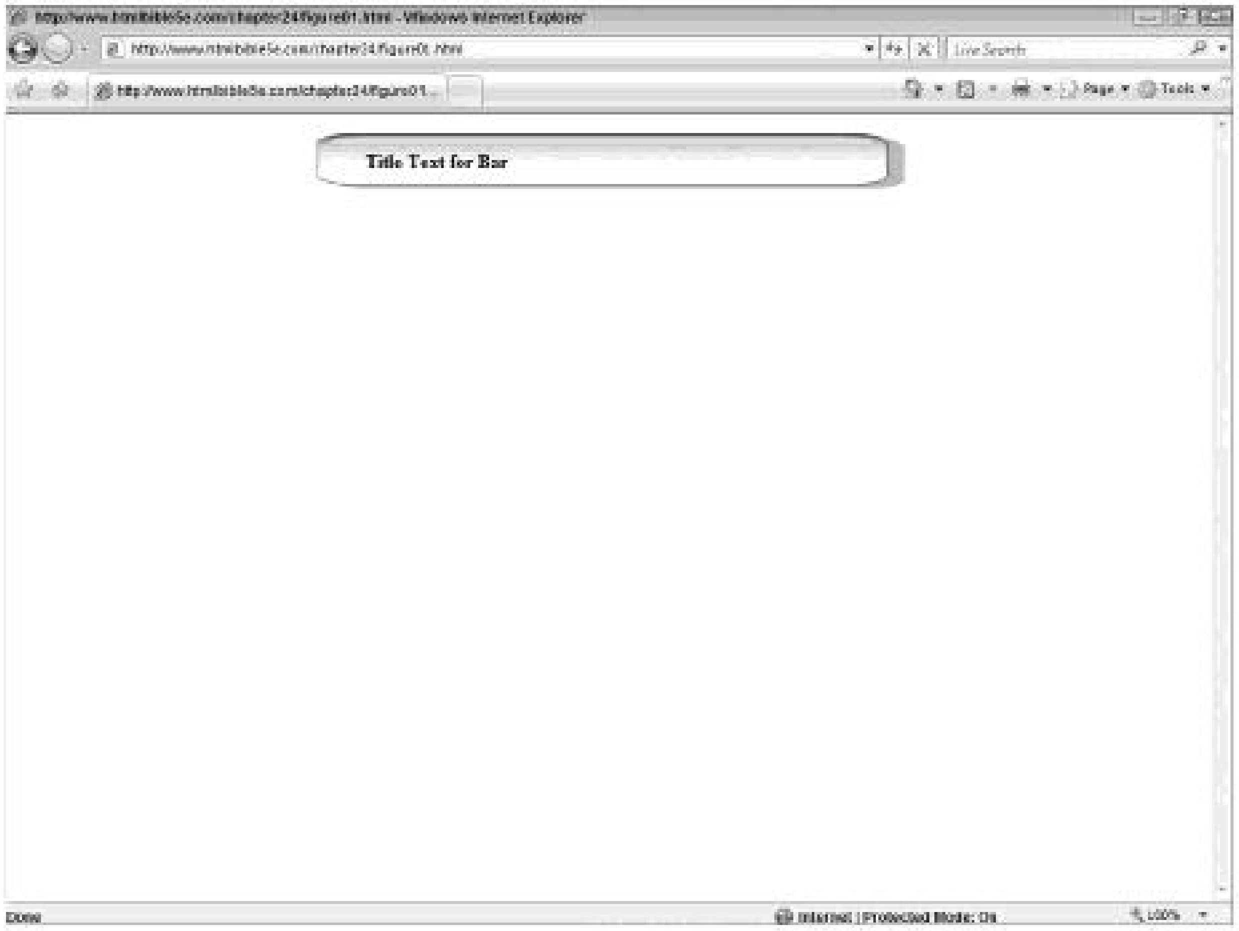
Now, if the number needs to be broken, it will be broken after a comma or after the decimal point.

Stretching Title Bars

In Chapter 41 you will see how to use CSS to create elaborate expandable buttons. However, you can achieve similar results in HTML. This section demonstrates how to create an expandable header bar like that shown in Figure 24-1.

FIGURE 24-1

Using background graphic layering in a table, you can create expandable title bars.


http://www.htmlbible5e.com/chapter24/figure01.html - Windows Internet Explorer
Live Search
Page
Tools
Title Text for Bar
Done
Internet | Protected Mode: On
100%

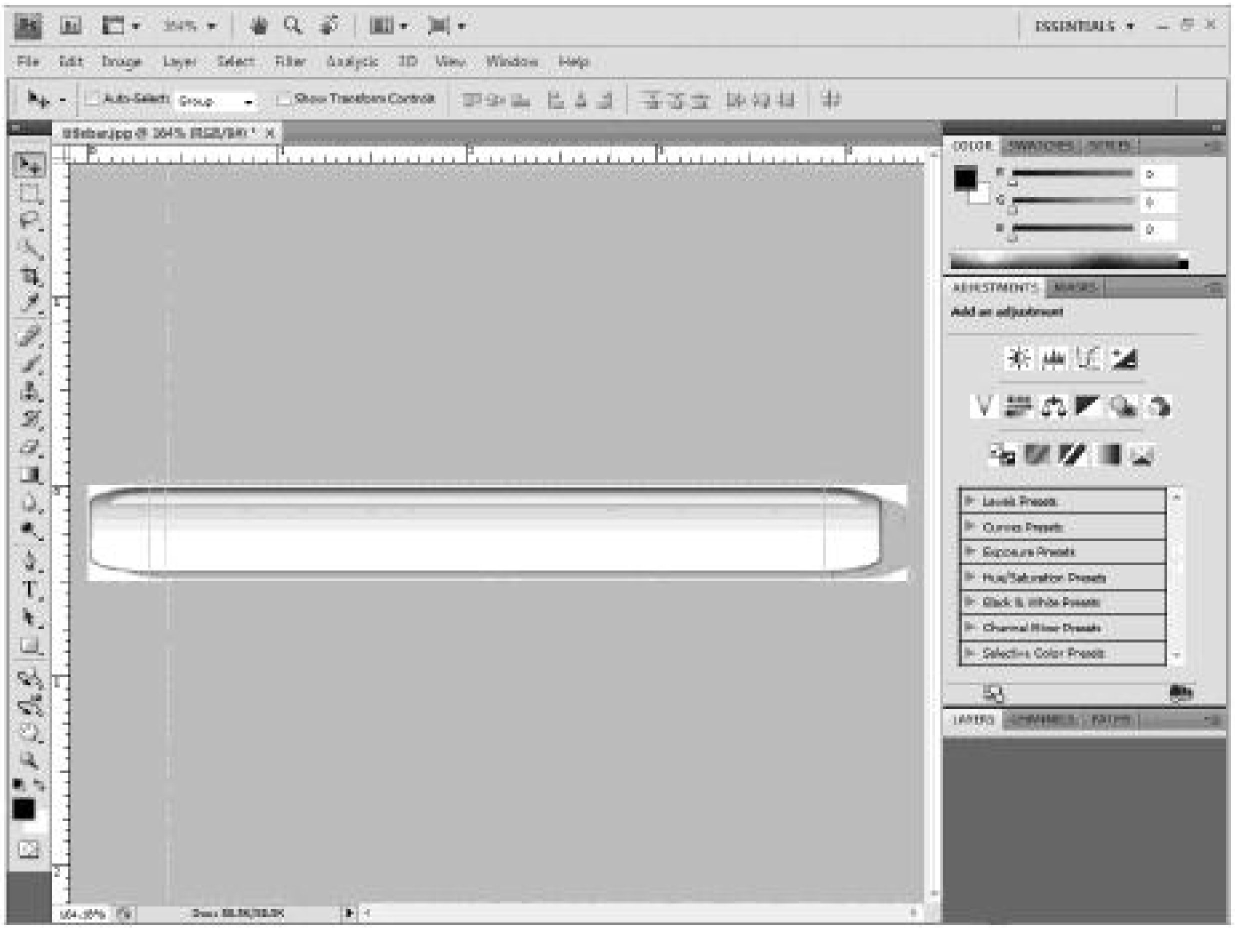
This technique is very simple but employs a method that isn't thought of very often — the use of background images.

The goal is to create a bar that can be stretched to accommodate any page width, design implementation, or text length. At first blush, you might be tempted to simply place the bar as a graphic, increasing its width via a custom value for the image's `width` property. However, doing so will alter the image's aspect ratio and distort it accordingly.

The better choice is to use a sectioned image consisting of the bar's end caps and a slice of its midsection. Figure 24-2 shows a basic title bar created and sliced in a basic graphic editing program. The bar is sliced into four pieces: the two end caps, a slice of midsection, and the remainder of the midsection.

FIGURE 24-2

A title bar can be easily created and sliced into desired pieces with most graphic editing programs.



Note

The large section of the midsection is an unnecessary piece; you only need the small center slice. ■

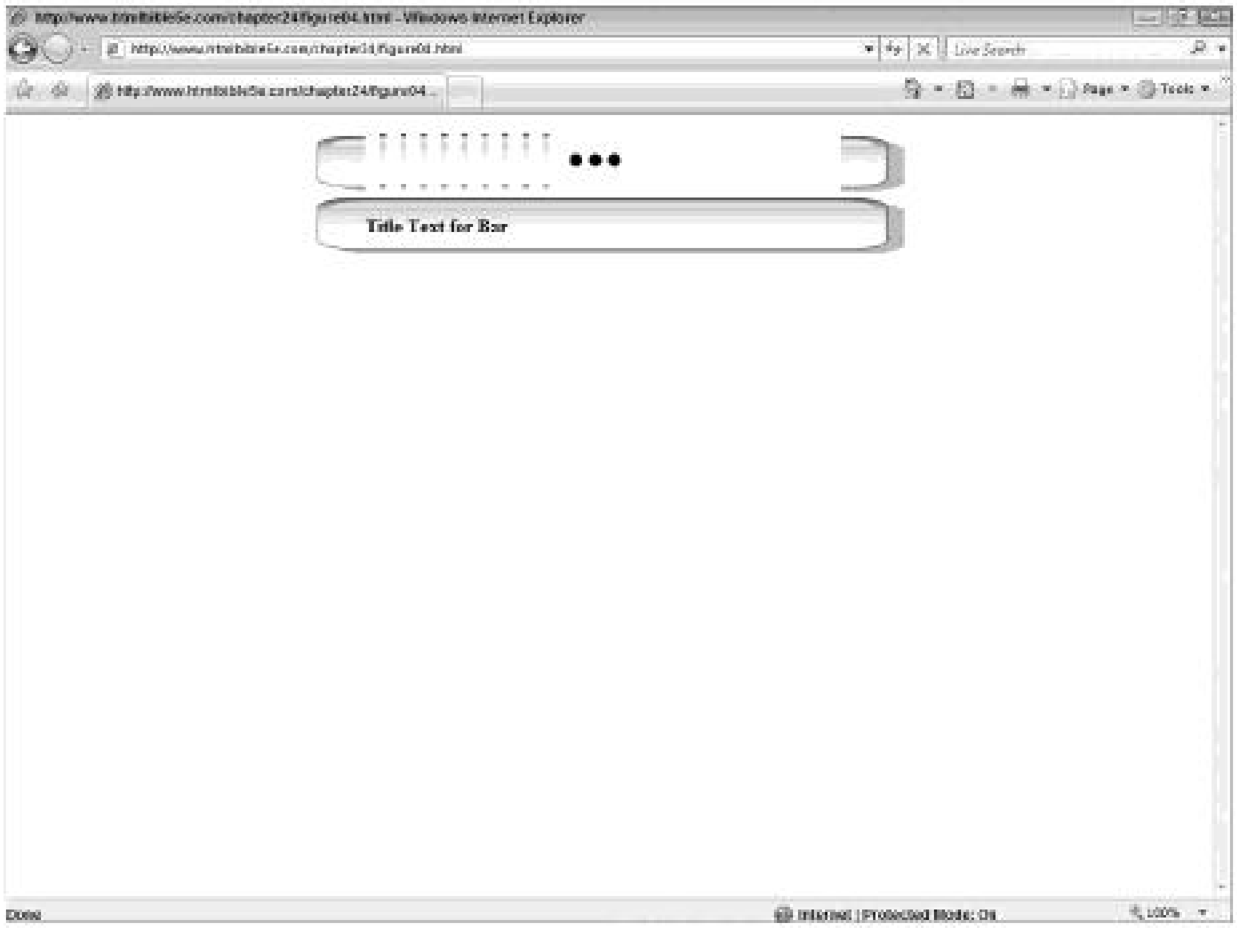
Once you have the three pieces, you place them in a three-celled table, putting the end caps in the first and last cells and using the slice as the background for the middle cell. Use of the `background-repeat` property causes the slice to repeat through the width of the cell without any distortion. The text for the bar becomes the content of the middle cell. The following document shows how the bar is constructed:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
<body>
<table width="50%"  border="0" align="center"
        cellpadding="0" cellspacing="0">
  <tr>
  <td width="2%"><img src="BarLeft.jpg" width="43"
        height="50"></td>
  <td width="96%" style="background-image:
        url('BarSliver.jpg');
        background-repeat: repeat-x;">
    <strong>Title Text for Bar</strong></td>
  <td width="2%"><img src="BarRight.jpg" width="53"
        height="50"></td>
  </tr>
</table>
</body>
</html>
```

Figure 24-3 shows how the method achieves the desired effect.

FIGURE 24-3

The top image shows how the bottom image is achieved.


http://www.htmlbible5e.com/chapter24/figure04.html - Windows Internet Explorer
Live Search
Title Text for Bar
Done
Internet | Protected Mode: On
100%

The bar can easily be extended by increasing the width of the middle cell — the slice will be tiled to fill the extra space. If the table is set to dynamically expand, then the bar will also expand dynamically with its table.

Simulating Newspaper Columns

Although CSS advocates are known to proclaim that table formatting is dead, you can still achieve many impressive layouts with tables. One in particular, newspaper columns, is often used.

Newspaper columns are narrow, parallel, vertical columns of text. This layout incorporates an optional heading that straddles the top of the columns just like a newspaper headline.

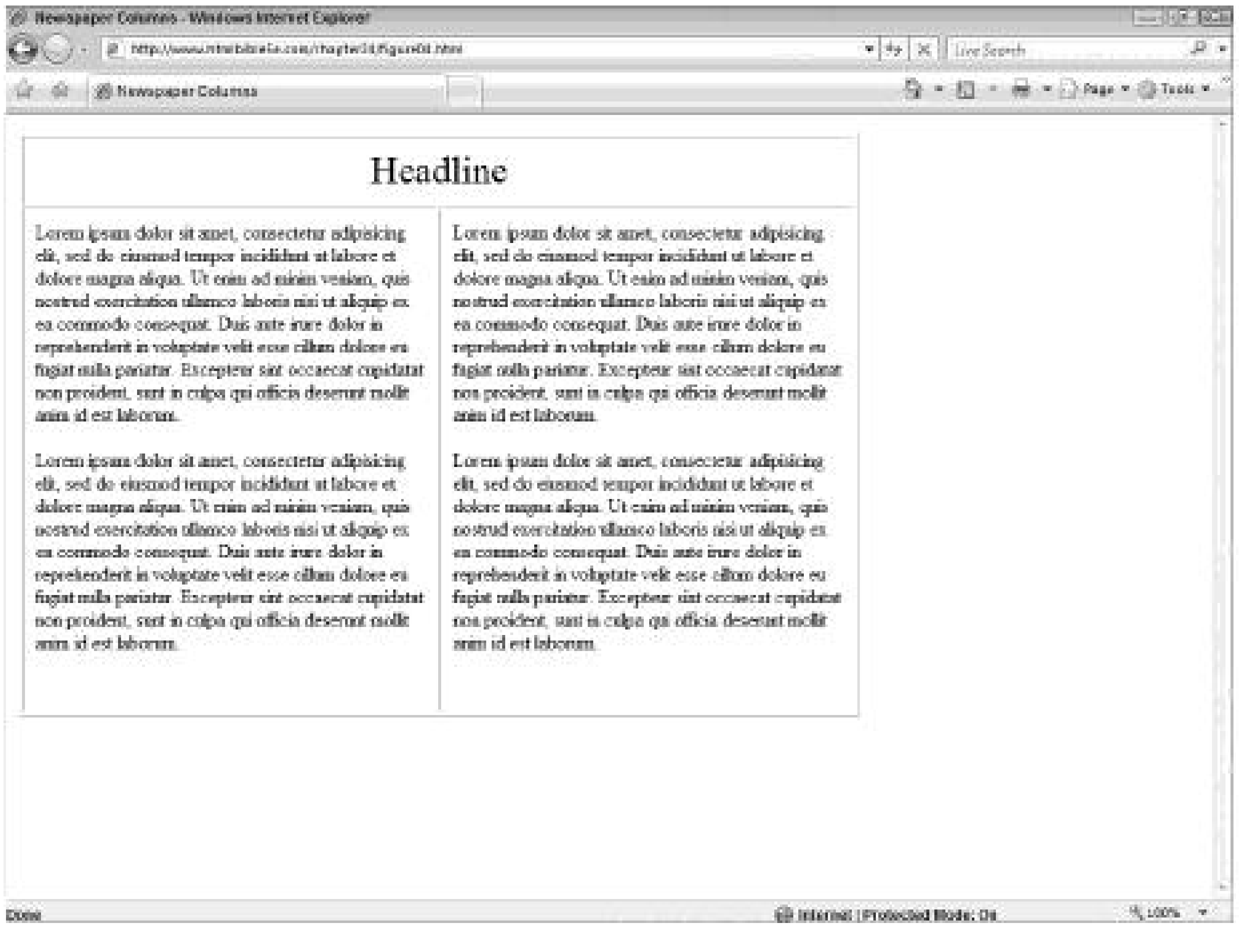
The key is to use the `colspan` attribute with the first table cell, causing it to span multiple columns. The text columns follow on the next row, occupying one table column each. For example, the following code sets up a table for two newspaper columns with a heading above them, as shown in Figure 24-4:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
  <title>Newspaper Columns</title>
  <style type="text/css">
    #newspaper { height: 100px;
                 width: 700px; }
    #headline { font-size: xx-large;
                text-align: center; }
    #column { height: 400px; }
  </style>
</head>
<body>
<table id="newspaper" border="1" cellpadding="10px">
  <tr>
    <td id="headline" colspan="2">Headline</td>
  </tr>
  <tr>
    <td id="column" valign="top" width="50%"><p>Lorem ipsum
dolor sit amet, consectetur adipisicing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad
minim veniam, quis nostrud exercitation ullamco laboris nisi ut
aliquip ex ea commodo consequat. Duis aute irure dolor in
reprehenderit in voluptate velit esse cillum dolore eu fugiat
nulla pariatur. Excepteur sint occaecat cupidatat non proident,
sunt in culpa qui officia deserunt mollit anim id est
laborum.</p><p>Lorem ipsum dolor sit amet, consectetur
adipisicing elit, sed do eiusmod tempor incididunt ut labore et
dolore magna aliqua. Ut enim ad minim veniam, quis nostrud
exercitation ullamco laboris nisi ut aliquip ex ea commodo
consequat. Duis aute irure dolor in reprehenderit in voluptate
velit esse cillum dolore eu fugiat nulla pariatur. Excepteur
sint occaecat cupidatat non proident, sunt in culpa qui officia
deserunt mollit anim id est laborum.</p</td>
    <td id="column" valign="top" width="50%"><p>Lorem ipsum
```

```
dolor sit amet, consectetur adipisicing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad
minim veniam, quis nostrud exercitation ullamco laboris nisi ut
aliquip ex ea commodo consequat. Duis aute irure dolor in
reprehenderit in voluptate velit esse cillum dolore eu fugiat
nulla pariatur. Excepteur sint occaecat cupidatat non proident,
sunt in culpa qui officia deserunt mollit anim id est
laborum.</p><p>Lorem ipsum dolor sit amet, consectetur
adipisicing elit, sed do eiusmod tempor incididunt ut labore et
dolore magna aliqua. Ut enim ad minim veniam, quis nostrud
exercitation ullamco laboris nisi ut aliquip ex ea commodo
consequat. Duis aute irure dolor in reprehenderit in voluptate
velit esse cillum dolore eu fugiat nulla pariatur. Excepteur
sint occaecat cupidatat non proident, sunt in culpa qui officia
deserunt mollit anim id est laborum.</p></td>
  </tr>
</table>
</body>
</html>
```

FIGURE 24-4

A two-column newspaper layout


Newspaper Columns - Windows Internet Explorer
http://www.htmlbible.com/chapter24/figure04.html
Live Search
Newspaper Columns
Page
Tools
Done
Internet | Protected Mode: On
100%

Headline

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

For a production layout you will probably want to set the table's `border` attribute to `0`.

The keys to a good design are to either fix the table width or set it to 100 percent, and ensure that the columns occupy only their share of the table width (text oddities can cause column widths to shift) by explicitly setting their width, as well. Setting the column height isn't always necessary, but sometimes you will want to fix column lengths with a `height` property or set a minimum length with a `min-height` property.

Tip

You can use the `<td>` tag's `colspan` and `rowspan` attributes to make some very creative table designs. ■

Including Image Size for Fast Display

The importance of always specifying an image's dimensions was brought up in Chapter 12, but it bears repeating here. When a browser loads an image to display, it needs one of two things in order to ascertain the size of the image:

- The completion of the image load
- The availability of `width` and `height` properties in the `<img>` tag

Either option will result in a page displaying the image with its correct size. However, the first option causes a delay before the image displays correctly, which can lead to a few unwelcome side effects for the end user. One possible side effect is that the user agent will stop loading all content until the image is loaded, sized, and can be displayed properly. Another possible side effect is that the user agent will reserve a portion of the document for the image, and reformat the document to fit the final size after the image's size is known.

If the size is known, the user agent will reserve the proper amount of space for the image and continue to load the rest of the document. When the image is fully loaded it is dropped into the reserved area with no ill effects for the end user.

Protecting E-mail Addresses

Placing an e-mail address on a Web page is a dangerous prospect nowadays. If the document on which the address appears generates even a medium amount of traffic, it is a given that a robot or other harvester will pick up the e-mail address and add it to dozens of spam lists.

These bots and harvesters collect the e-mail address by simply accessing the document and examining the document's source. For example, to insert a link to e-mail Jill at The Oasis of Tranquility, the following code can be inserted into a document:

```
<a href="mailto:jill@oasisoftranquility.com">Email Jill</a>
```

Although this is displayed as simply "Email Jill" on a user agent's screen, the harvester is able to look at the code to find `mailto:jill@oasisoftranquility.com`. The `mailto` protocol confirms that an e-mail address is within the anchor tag.

The key to protecting your e-mail address is to avoid adding it to documents in an unencoded format. Instead, obfuscate it using one of several methods, including the following:

- Break it into pieces that are reassembled by a script, which can't be easily discerned by the harvesters.
- Encode it using a method that can preserve its functionality.

Tip

One low-security method for obscuring an e-mail address is to replace the at sign (@) with its entity equivalent, `&#64;`. This method relies on the assumption that most harvesters search documents for the literal "@" in their quest for e-mail addresses. By removing the literal at sign, you impede the harvester's ability to recognize e-mail addresses. Using the equivalent entity ensures that compliant browsers will still render the at sign properly. Unfortunately, most harvesters are now aware of this trick and recognize the entity as well as the literal at sign. ■

The first method is fairly straightforward and uses a script similar to the following:

```
<script type="text/JavaScript">
  document.write('<a href="');
  t1 = "mai";
  t2 = "lto";
  t3 = ":";
  t4 = "jill";
  t5 = "&#64;";
  t6 = "oasi";
  t7 = "softra";
  t8 = "nquil";
  t9 = "ity";
  t10 = ".";
  t11 = "com";
  text = t1+t2+t3+t4+t5+t6+t7+t8+t9+t10+t11;
  document.write(text);
  document.write('">Mail Jill</a>');
</script>
```

The preceding script breaks the e-mail portion into small chunks, assigns each chunk to a variable, concatenates the chunks into one variable, and then outputs the entire anchor tag. The key to this method is that the pieces of the e-mail address never appear together in the file. For additional security, you could scramble the order of the chunks — for example, placing number 6 before number 3, and so on.

The other method, encoding the address, is a little more complicated. It requires that you first run a program to encode the address and then use those results in your document. The encoding can be done in a variety of ways, one of which is shown in the following listing, an HTML document with form entry and JavaScript for the encoding:

```
<html>
<head>
  <title>Email Encoder</title>
  <script type="text/JavaScript">
    function encode (email) {
      var encoded = "";
      for (i = 0; i < email.length; i++) {
        encoded += "&#" + email.charCodeAt(i) + ";";
      };
      return (encoded);
    };
  </script>
</head>
<body>
<form action="" name="encoder"
  onsubmit="encoded.value = encode(email.value);
  return false;">
<table border="0" cellpadding="3px">
  <tr>
    <td>Enter your<br/>email address:</td>
    <td><input type="text" name="email" size="30" /></td>
    <td><input type="submit" value="Encode" /></td>
  </tr>
  <tr>
    <td>Encoded email:</td>
    <td colspan="2"><input type="text" name="encoded"
        size="60" /></td>
  </tr>
</table>
</form>
</body>
</html>
```

This document displays a form in which you can enter your e-mail address. When you click the `Encode` button, the e-mail address you entered is converted, character by character, into entity equivalents and placed in the `Encoded email` field where you can copy it to the clipboard for use in your documents. Note that you can encode the e-mail address only or, optionally, the `mailto:` protocol string or even the entire anchor tag. Just be sure to replace the same amount of text in your document as you encoded.

Note

Although the encoded method might seem to be the most secure method of protecting e-mail, it has a fatal flaw. Any bot that can decode entities — which, at their root, are ASCII codes — can read the encoded address very easily. ■

Automating Forms

A traditional use of JavaScript is the automation of forms. Using JavaScript, you can easily manipulate the status of form controls, validate form content, and more.

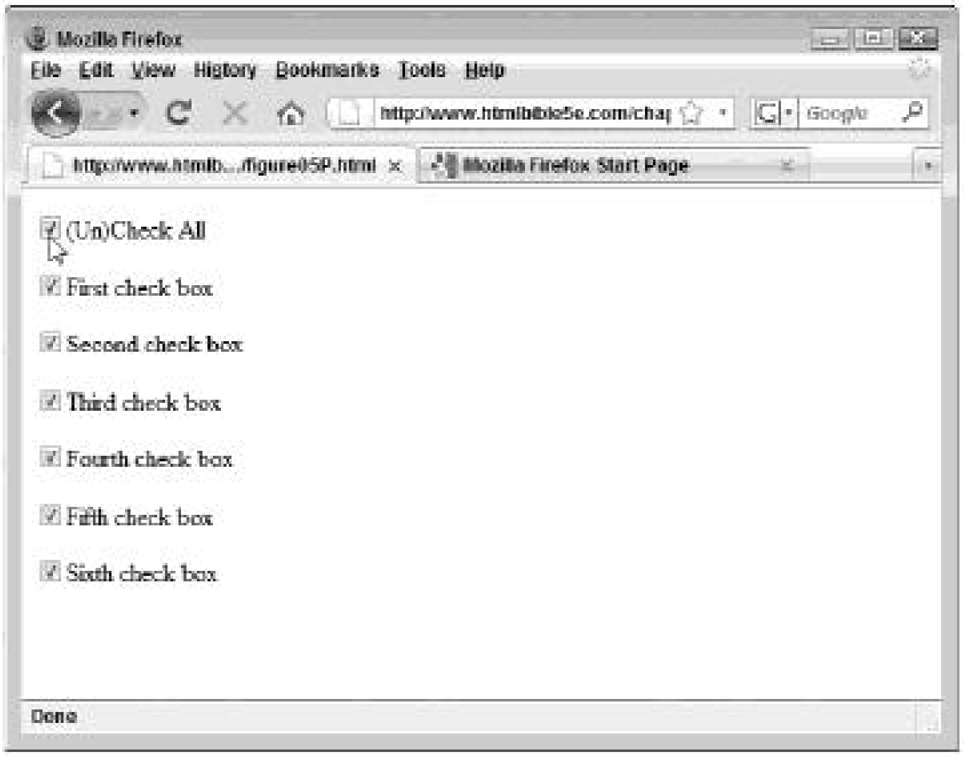
Manipulating form objects

One popular use of JavaScript and forms is to provide a special check box to check or uncheck the rest of the check boxes in the group. A bare-bones document with such a purpose resembles the following code, whose output is shown in Figure 24-5:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
   "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html>
<head>
<script type="text/JavaScript">

function checkall() {

  // Get the state of the first checkbox (0)
  var chk = document.form1.checks[0].checked;

  // Set the rest of the checkbox group to that value
  for (i = 1; i < document.form1.checks.length; i++) {
    document.form1.checks[i].checked = chk;
  }

}

</script>
</head>
<body>

<form name="form1">
<p><input id="allboxes" type="checkbox" name="checks"
      onClick="checkall();">(Un)Check All</p>
<p><input type="checkbox" name="checks">First check box</p>
<p><input type="checkbox" name="checks">Second check box</p>
<p><input type="checkbox" name="checks">Third check box</p>
<p><input type="checkbox" name="checks">Fourth check box</p>
<p><input type="checkbox" name="checks">Fifth check box</p>
<p><input type="checkbox" name="checks">Sixth check box</p>
</form>

</body>
</html>
```

FIGURE 24-5

JavaScript can help automate forms, making them easier to access, like the (Un)Check All check box shown here.


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://www.htmlbible5e.com/chap
http://www.htmlb.../figure05P.html
Mozilla Firefox Start Page
Done

☑ (Un)Check All

☑ First check box

☑ Second check box

☑ Third check box

☑ Fourth check box

☑ Fifth check box

☑ Sixth check box

The automation works thanks to a few, basic HTML and JavaScript constructs:

1. The group of check boxes, including the special check box, have the same `name` attribute, creating a collection of check boxes that can be accessed by a single JavaScript object.
2. The first check box, appropriately labeled "(Un)Check All," includes an `onClick` event handler that calls the JavaScript function `checkall()` when it is clicked (and changes states).
3. The `checkall()` JavaScript function reads the state of the special check box and iterates through the remaining check boxes setting their state to match the special check box's state. This causes the rest of the check boxes to be checked when the special check box is checked, and unchecked when the special check box is unchecked.

In this case the JavaScript function is hard-coded to work on the `checks` collection of check boxes, accessed via the JavaScript object `document.form1.checks`.

Such routines can be quite complex, identifying and modifying elements in the whole document, and even across forms. For example, the following JavaScript function will cause every check box with similar `value` attributes to be checked when one of its kin, anywhere in the document, is checked:

```
// Called from a checkbox's "onClick" event,
//   check all boxes with the same value attribute
function checkboxes(field) {

  // Get settings of the checkbox that called us
       checked = field.checked;
       value = field.value;

  // Retrieve all <input> elements into the MyElements array
       myBody = document.getElementsByTagName("body")[0];
       myElements = myBody.getElementsByTagName("input");

       // Iterate through the retrieved <input> elements,
       //   looking for checkboxes
       for( var x = 0; x < myElements.length; x++ ) {
              if (myElements[x].getAttribute('type') == "checkbox") {

      // If this checkbox (myElements[x]) has the same value
      //   as the checkbox that called us, set this box to
      //   the caller's value
      if (myElements[x].value == value) {
        myElements[x].checked = checked;
      }

              } // if myElements

       } // for x

} // End function
```

All that's left is to ensure that related check boxes have the same value and to place the following code in each check box tag so that when it is clicked it executes the function:

```
onClick = "checkboxes(this);"
```

Note

The preceding example relies upon the `value` attribute in check boxes. However, by modifying the code you can easily match other attributes to trigger the mass checking. ■

Validating form input

Another popular use of JavaScript is to validate form contents before submission to the server-side handler. Using Dynamic HTML, JavaScript can easily pick up form field values, evaluate them, and make decisions based on the values.

For example, the following code checks whether the field `x`, in the form named `form1`, is empty:

```
if (document.form1.x.value == "")
```

Similar JavaScript code can determine the length of content in a field, compare two dates in two fields, test for a valid e-mail address, and more. How does this code execute? Each decision-based piece of code could be tied to an `onChange` event in each field on the form, but that forces you to build several functions, one for each field. It's simpler to tie the decision-based code together in one function and call it when the form is submitted. At that point the code can check the whole form to determine whether it should be submitted or rejected.

To tie a function to a form's submission action, you use the `onSubmit` event handler in the `<form>` tag, similar to the following:

```
<form action="handler.cgi" method="POST" onSubmit="return validation()">
```

The `onSubmit` handler causes the JavaScript function `validation()` to execute when the form's `submit` button is pressed (or any other action results in the form being submitted). The return value of the function determines whether the form is actually submitted — a return value of `true` allows the form to be submitted, whereas a return value of `false` prevents the form from being submitted.

The JavaScript function is constructed like the following pseudocode:

```
function validate() {

  test condition1
  if test fails:
    display message
    return false

  test condition1
  if test fails:
    display message
    return false

  test condition1
  if test fails:
    display message
    return false

  test condition1
  if test fails:
    display message
    return false

  ...

  return true
```

Note that in each case that the function returns `false`, the function's execution stops and the form is not submitted. However, the function also displays a message to users indicating the error encountered, giving them the chance to correct the error. This method of validating forms is serial — that is, each field is validated in order, and a single failure stops the validation. In such a case the user must deal with each error before validation can progress to the next test.

The following code shows a validation function that checks a handful of fields in a form. The comments in the code are self-explanatory as to what is checked and how:

```
function validate() {

  // Check to see if the "y" field's value exceeds 80
  if (parseFloat(document.form1.y.value) >= 80) {
    alert("The value of Y cannot exceed 80.
           Please enter an appropriate value.");
    document.form1.field.focus();
    return false;
  }

  // Check to see if the "x" field is blank
  if (document.form1.x.value == "") {
    alert("X is a required field.");
    document.form1.field.focus();
    return false;
  }

  // Check to see if the "ending_date field" is greater
  //   than the "starting_date field" (end after beginning)
  // Only check if we have both values
  if(document.form1.ending_date.value!="" &&
     document.form1.starting_date.value!="" )
  {
    // Split the dates by their dashes
    var from_date = document.form1.starting_date.value.split("-");
    var to_date = document.form1.ending_date.value.split("-");

    // Re-assemble dates in sortable method
    str_from_date = from_date[2]+from_date[0]+from_date[1];
    str_to_date = to_date[2]+to_date[0]+to_date[1];

    // Check date order
    if(str_from_date > str_to_date){
      alert("Ending Date should be greater then starting Date.");
      document.form1.ending_date.focus();
      return false;
    }
  }


  // Test for a valid email address in email field
  // Use a regular expression to match against the field's contents
  var regx = new
      RegExp(/^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/);
      if (!regx.test(document.form1.email.value)) {
        alert("The email address is invalid.
               Please enter a valid address.");
        return false;
      }

      return true;

    } // End validate function
```

Note
The preceding example uses simple field and form names. In an actual environment you would use more descriptive names that match your usage. In addition, the preceding examples use some fairly complex methods such as regular expression matching. If you intend to do a lot of form work with JavaScript, you are encouraged to pick up a comprehensive JavaScript book, like Wiley's JavaScript Bible, 7th Edition. ■

Modifying the User Agent Environment

Not only can JavaScript modify CSS settings and HTML, it can also modify the user agent. For example, you can move and resize the user agent's window, manipulate its scroll bars, and more. To demonstrate this, we will create JavaScript to scroll the current document vertically, in both directions (up and down).

The concept

The concept for this exercise is simple — provide a control that when moused over will scroll the current document. When the end of the document is reached, the scrolling will scroll upward to the top of the document where it will again scroll downward. When the mouse is moved off of the control, the scrolling will stop.

Note
Although this example has limited usage — thanks to the scroll wheel on most mice — it is a good example of using a controlling element to orchestrate several events. ■

For this example we will use a `<div>` with a fixed position as the control.

The implementation

The code for this exercise appears in the next two listings. Listing 24-1 contains the JavaScript for the document (in an external file); Listing 24-2 contains the HTML document itself, which is shown in Figure 24-6.

LISTING 24-1

JavaScript to control the scroll bar

```
//////////////////////////////////////////////////
// Retrieve and return the current offsets
//   for both scrollbars (x = horizontal,
//   y = vertical)
//////////////////////////////////////////////////
function getScrollXY() {
  // Set offsets to zero
  var scrOfX = 0, scrOfY = 0;

  // Determine browser mode and set
  //  offsets to current scroll position
  if (typeof(window.pageYOffset) == 'number') {
    //Netscape compliant
    scrOfY = window.pageYOffset;
    scrOfX = window.pageXOffset;

  } else if(document.body &&
           (document.body.scrollLeft ||
            document.body.scrollTop)) {
    //DOM compliant
    scrOfY = document.body.scrollTop;
    scrOfX = document.body.scrollLeft;

  } else if(document.documentElement &&
           (document.documentElement.scrollLeft ||
            document.documentElement.scrollTop)) {
    //IE6 standards compliant mode
    scrOfY = document.documentElement.scrollTop;
    scrOfX = document.documentElement.scrollLeft;
  }

  // Return array with offsets
  return [ scrOfX, scrOfY ];
}  // Function getScrollXY


//////////////////////////////////////////////////
// Change the direction (dir) of the scroll and
//   the background image of the scroller (to match
//   direction)
//////////////////////////////////////////////////
function chdirscroll () {
  dir = dir * -1;
  obj = document.getElementById("scroller");
  if (dir == 1) {
    obj.style.backgroundImage = "url(images/dn_arrow.gif)";
  } else {
    obj.style.backgroundImage = "url(images/up_arrow.gif)";
  }
}

//////////////////////////////////////////////////
// Scroll the document in the y direction (vertical) by the
//   currently set amount (increment * direction)
//////////////////////////////////////////////////
function doscroll() {

  // Initialize values
  var y = -1;
  var yy = 0;

  // Get current scroll position
  xy = getScrollXY();
  y = xy[1];

  // Scroll the document
  scrollBy(0,inc*dir);

  // Check new position, if it is the same as the old
  //   position, the scrollbar did not move -- it is
  //   at the top or bottom of the document
  xy = getScrollXY();
  yy = xy[1];
  // If scrollbar is at the top or bottom, reverse the
  //   direction and bounce, if bounce option set. If,
  //   bounce is not set, stop the scrollbar's movement
  if (y == yy) {
    if (bounce == 1) {
      chdirscroll();
    } else {
      clearInterval(scrollIt);
    }
  }
} // End function doscroll()


//////////////////////////////////////////////////
// Begin scrolling according to previously set speedmod
//////////////////////////////////////////////////
function setscroll () {
  inc = speedmod * 20;  // Set the increment to move, in
                        //   multiples of 20 (by speedmod)
  // Start scrollbar movement (by inc) every 25 milliseconds
  scrollIt = setInterval("doscroll(inc);",25);
}
//////////////////////////////////////////////////
```

continued

LISTING 24-1 *(continued)*

```
// Stop the scrolling
//////////////////////////////////////////////////
function stopscroll () {
  clearInterval(scrollIt);
}

//////////////////////////////////////////////////
// Initialize the scroll values and settings
//////////////////////////////////////////////////
function initscroll () {
  // Init the scroll variables
  dir = 1;       // The direction to move (positive = down)
  bounce = 1;    // Whether to bounce (bounce = 1) or not (bounce = 0)
  speedmod = 1; // The speed modifier (higher = faster)
  // Assign the down arrow to the scroller
  obj = document.getElementById("scroller");
  obj.style.backgroundImage = "url(images/dn_arrow.gif)";
}
```

LISTING 24-2

The document with a scrollbar control

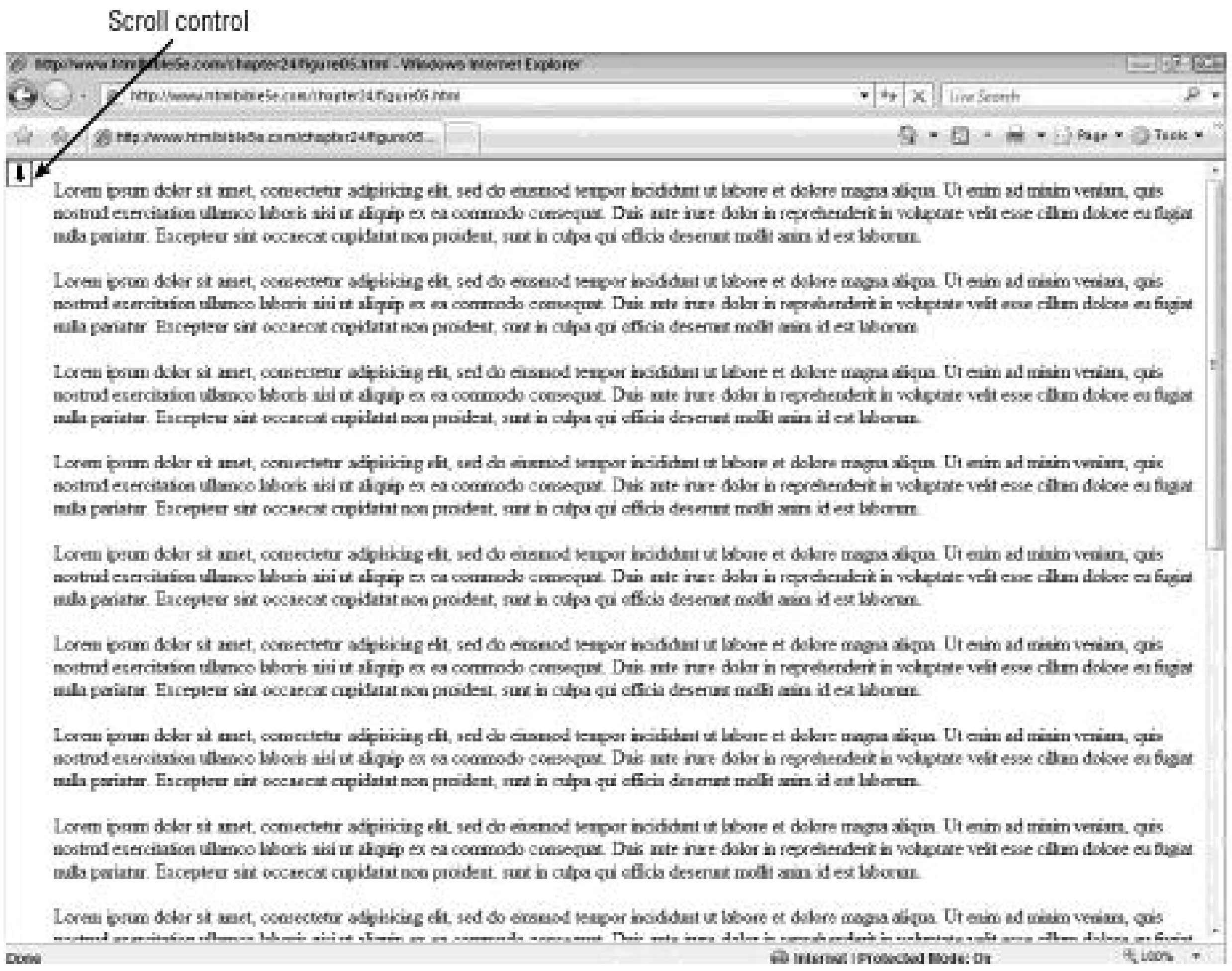
```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">

<html>
<head>

  <script type="text/JavaScript" src="scroll.js"></script>

  <style type="text/css">

    #scroller {
      width: 20px;
      height: 20px;
      background-color: none;
      background-repeat: no-repeat;
      background-position: center center;
      border: 1px solid black;
      position: fixed;
      top: 0px;
      left: 0px;
      }

    #content {
      margin-left: 30px;
    }
  </style>

</head>

<body onLoad="initscroll();">

<div id="scroller" onMouseOver="setscroll();" onMouseOut="stopscroll();"
     onClick="chdirscroll();">&nbsp;</div>

<div id="content">
      <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed
do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim
ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut
aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit
in voluptate velit esse cillum dolore eu fugiat nulla pariatur.
Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia
deserunt mollit anim id est laborum.</p>

       ...

      <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed
do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim
ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut
aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit
in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint
occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit
anim id est laborum.</p>

</div>  <!-- /Content -->

</body>
</html>
```

The desired implementation is straightforward:

- An `onLoad` event trigger is attached to the `<body>` tag so that the scrolling options are set (globally) in preparation for scrolling when the document is first loaded.
- A fixed `<div>`, positioned at the upper-left corner of the user agent window, is used to control the scrolling.
- Two event triggers, `onMouseOver` and `onMouseOut`, are attached to the controlling `<div>` and are used to start and stop the scrolling. When the mouse is placed over the `<div>`, the scrolling starts. When the mouse is moved off of the `<div>`, the scrolling stops.
- A third event trigger, `onClick`, is used to change the direction of the scroll. When the controlling `<div>` is clicked, the scrolling changes direction.
- By default, the scrolling "bounces" — that is, when it reaches the bottom of the document it begins traveling upward to the top of the document where it bounces back down.

FIGURE 24-6

The scroll control division appears and stays in the upper-left corner of the window.


Scroll control
http://www.htmlbible5e.com/chapter24/figure06.html - Windows Internet Explorer
http://www.htmlbible5e.com/chapter24/figure06.html
Live Search
Page
Tools
Done
Internet | Protected Mode: On
100%

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis

The document layout is accomplished with two divisions. The "scroller" `<div>` is a small square with an arrow pointing in the direction the document will scroll. It is used for turning on and off the scroll. The `<div>` position is set to fixed so it will stay where it was placed when the document scrolls.

A second `<div>` is used to contain the actual content of the document. Note how the "content" `<div>` has a larger left margin specified so as to not conflict with the scroll `<div>`. For this example, the document simply contains several paragraphs of the Lorem Ipsum placeholder text to provide a simple document long enough to scroll.

The JavaScript functions

The scrolling control is accomplished by six JavaScript functions:

- `initscroll()`: Called by the `<body>` tag's `onLoad` event, this function sets the initial values, preparing for the scrolling action — the initial direction (`dir`) of the scroll (1 = down, -1 = up), whether the scroll should bounce (1 = yes, 0 = no), and the speed modification (speedmod, faster values = faster movement). The settings in this function should be used to tailor your scrolling actions.

- `getScrollXY()`: This function is called to retrieve the current position of the scroll bars. It performs basic browser detection to determine how to retrieve the horizontal (x) and vertical (y) coordinates of the scroll bars. The coordinates are returned in an array, with x being index 0, and y being index 1.
- `setscroll()`: This function begins the scrolling action by setting the requisite variables and setting an Interval timer to move the scroll bar by the specified increment (`speedmod * 20`) every 25 milliseconds.
- `doscroll()`: This function moves the scroll bar every so often as defined by the Interval timer. Every 25 milliseconds the function is called by the timer and the scroll bar moves via the `scrollBy()`JavaScript function. The function then uses the `getScrollXY()`function to check whether the before position (y) is the same as the after position (yy). If the two values are the same, then the scroll bar didn't move and must be at the beginning or the end of the document. If that's the case, then the function `chdirscroll()`is called to reverse the scroll bar's direction (providing `bounce` is set to 1).
- `chdirscroll()`: This function reverses the direction of the scroll bar's movement by changing the value of `dir` to 1 (move down) or −1 (move up). The actual movement is handled by the `doscroll()`function.
- `stopscroll()`: This function is used to remove the Interval timer, effectively stopping the scroll. It is tied to the control `<div>`'s `onMouseOut` event and can also be called by other events or functions that require the scroll bar's movement to stop.

Tip

To perform other operations on the user agent's environment, use Google to search for applicable JavaScript functions. For example, to see how to move the user agent window, search for JavaScript move window. ■

Summary

This chapter presented a handful of HTML tips and tricks you can employ in your documents, including those related to presenting content, as well as a few on optimizing or speeding up your document's delivery. This is the last chapter in the HTML section (Parts I and II). The next part starts the coverage of CSS.

Part III

Controlling Presentation with CSS

IN THIS PART

CHAPTER 25

CSS Basics

The Web was founded on HTML and plain-text documents. Over the last few years the Web has become a household and industrial mainstay, maturing into a viable publishing platform thanks in no small part to Cascading Style Sheets (CSS).

CSS enables Web authors and programmers to finely tune elements for publishing both online and across several different types of media, including print format. This chapter serves as the introduction to CSS. Subsequent chapters in this section will show you how to use styles with specific elements.

IN THIS CHAPTER

The Purpose of Styles

Styles are an electronic publishing invention for dynamically coding text and other document elements with formatting. For example, a style called "Heading" would be attached to every heading in the document. The style definition would contain information about how headings should be formatted. In this book, for example, headings (such as "The Purpose of Styles," above) use a larger, bold font.

Note

Anyone who has spent an appreciable amount of time in and around a word-processing program has no doubt encountered styles. The concept of styles used by word processors does not differ appreciably from that of CSS and the Web — if you understand the former, you should have a good grasp on usage of the latter. ■

The advantage of styles is that you can change a definition once and that change affects every element using that style. Coding each element individually, by contrast, would require that each element be recoded individually whenever you wanted them all to change. Thus, styles provide an easy means to update document formatting and maintain consistency across multiple documents.

Coding individual elements is best done while the document is being created. This means that the document formatting is usually done by the author — not always the best choice. Instead, the elements can be tagged with appropriate styles (such as `heading`) while the document is created, and the final formatting can be left up to another individual who defines the styles.

Styles can be grouped into purpose-driven style sheets. *Style sheets* are like blueprints, holding groups of styles relating to a common purpose. Style sheets enable multiple styles to be attached to a document all at once and for all the style formatting in a document to be changed at once. Therefore, documents can be quickly formatted for different purposes — one style sheet can be used for documents meant for online consumption, another style sheet can be used (on the same documents) for brochures, and so on.

Styles and HTML

For a tangible example that uses HTML, consider the following code:

```
<p><b><u>Rabbit Run Racing</u></b></p>
<p>Rabbit Run Racing is similar to many of the "cart" racing games
on the market. You pick a character from one of six available and
race around a small track, picking up power ups and trying to beat
your opponent(s) to the finish line. Rabbit Run Racing supports 1-4
players.</p>
<p><b><u>Driving Range III</u></b></p>
<p>Driving Range III is one of the first games to take advantage of
the momentum joystick, using its pendulum-weighted motion to
simulate a golf driving iron. Unfortunately, the game is too simple
to hold much replay value--you drive balls on three different
ranges, two with a handful of trick-shot areas. Driving Range III is
a single-player only game, though you can compare high-scores with
your buddies.</p>
<p><b><u>Run, Gun, Gore</u></b></p>
<p>Capitalizing on the revitalized run and gun genre, RGG bring
plenty of everything in its title to the table. The graphics are
surprisingly crisp and the levels are designed well for
deathmatch-style play. Although there are surprisingly few levels in
the current release (10), soon to be released add on packs promise
more theme-driven levels. RGG is a single-player game, but supports
up to 100 players when linked across the NGame network.</p>
```

Note

For the purpose of this example, ignore the fact that most of the text formatting tags (underline, center, and so on) have been deprecated. ■

All three heading elements are coded bold and underlined. Now suppose that you wanted the heading elements to be larger and italicized. Each heading would have to be individually recoded, similar to the following:

```
<p><font size="+2"><i>Rabbit Run Racing</i></font></p>
```

Although using a decent text editor with global search and replace makes this change pretty easy, consider managing an entire site, with several — if not tens or hundreds of — documents, each with numerous headings. Each document makes the change exponentially harder.

Now, let's look at how styles would change the example. With styles, the example could be coded similarly to the following:

```
<p class="heading">Rabbit Run Racing</p>
<p>Rabbit Run Racing is similar to many of the "cart" racing games
on the market. You pick a character from one of six available and
race around a small track, picking up power ups and trying to beat
your opponent(s) to the finish line. Rabbit Run Racing supports 1-4
players.</p>
<p class="heading">Driving Range III</p>
<p>Driving Range III is one of the first games to take advantage of
the momentum joystick, using its pendulum-weighted motion to
simulate a golf driving iron. Unfortunately, the game is too simple
to hold much replay value--you drive balls on three different
ranges, two with a handful of trick-shot areas. Driving Range III is
a single-player only game, though you can compare high-scores with
your buddies.</p>
<p class="heading">Run, Gun, Gore</u></b></p>
<p>Capitalizing on the revitalized run and gun genre, RGG bring
plenty of everything in its title to the table. The graphics are
surprisingly crisp and the levels are designed well for
deathmatch-style play. Although there are surprisingly few levels in
the current release (10), soon to be released add on packs promise
more theme-driven levels. RGG is a single-player game, but supports
up to 100 players when linked across the NGame network.</p>
>
```

Cross-Ref

There are several methods for applying styles to document elements. Chapter 26 covers ways to define and use styles. ■

The style is defined in the head section of the document, similar to the following:

```
<head>
  <style type="text/css">
    p.heading { font-weight: bold; text-decoration: underline; }
  </style>
</head>
```

This definition defines a heading class that formats text appearing in a paragraph with a heading as bold and underlined.

Cross-Ref

Style definitions and selectors are covered in Chapter 26. Style property values and units are covered in Chapter 27. Individual CSS properties are covered in appropriate chapters later in this part of the book. ■

To change all the headings in the document to a larger, italic font, the *one* definition can be recoded:

```
<head>
  <style type="text/css">
    p.heading { font-size: larger; font-style: italic; }
  </style>
</head>
```

CSS Levels 1, 2, and 3

There are three levels of CSS — two levels are actual specifications, whereas the third level is in recommendation status. The main differences between the three levels are as follows:

- CSS1 defines basic style functionality, with limited font and limited positioning support.
- CSS2 adds aural properties, paged media, and better font and positioning support. Many other properties have also been refined.
- CSS3 will add presentation-style properties, enabling you to effectively build presentations from Web documents (similar to Microsoft PowerPoint presentations).

You don't have to specify the level of CSS you are using, but you should be aware of what user agents will be accessing your site. Most modern browsers support CSS, but the level of support varies dramatically between user agents. It's always best to test your implementation on target user agents before widely deploying your documents.

Note

When using styles, keep in mind that not all style properties are well supported by all user agents. This book attempts to point out major inconsistencies and differences in the most popular user agents, but the playing field is always changing. ■

Defining Styles

Styles can be defined in several different ways and attached to a document. The most popular method for defining styles is to add a style block to the head of a document:

```
<head>
  <style type="text/css">
    ...Style definitions...
  </style>
</head>
```

If you use this method, all style definitions are placed within a style element, delimited by `<style>` tags. This tag has the following syntax:

```
<style type="MIME_type" media="destination_media">
```

In most cases, the MIME type is "text/css," as used throughout this chapter. The `media` attribute is typically not used unless the destination media is nontextual. The media attribute supports the following values:

- `all`
- `aural`
- `braille`
- `embossed`
- `handheld`
- `print`
- `projection`
- `screen`
- `tty`
- `tv`

Note

Multiple style definitions, each defining a style for a different medium and encased in its own `<style>` tags, can appear in the same document. This powerful feature enables you to easily define document styles for a variety of uses and deployment. ■

Alternately, the style sheet can be contained in a separate document and linked to documents using the link (`<link>`) tag:

```
<head>
  <link rel="stylesheet" type="text/css" href="mystyles.css" />
</head>
```

The style sheet document, `mystyles.css`, contains the necessary styles:

```
...
p.heading { font-size: larger; font-style: italic; }
...
```

This way, when the style definitions in the external style sheet change, *all* documents that link to the external sheet reflect the change. This provides an easy way to modify the format of many documents — whether to affect new formatting for visual reasons, or for another specific purpose.

Attaching external style sheets via the link tag should be your preferred method of applying styles to a document, as it provides the most scalable use of styles — you can change only one external style sheet to affect many documents.

Tip

You can add comments to your style section or style sheet by delimiting the comment with /* and */. For example, the following is a style comment:

```
/*  Define a heading style with a border */
```
■

Cascading Styles

So where does the "cascading" in Cascading Style Sheets come from? It comes from the fact that styles can stack, or override, each other. For example, suppose that an internal corporate website's appearance varies according to the department that owns the various documents. All the documents need to follow the corporate look and feel, but the Human Resources department might use people-shaped bullets or apply other small changes unique to that department. The HR department doesn't need a separate, complete style sheet for its documents — it needs only a sheet containing the differences from the corporate sheet. For example, consider the following two style sheet fragments:

```
/*  corporate.css */
body {
  font-family:verdana, palatino,  georgia,  arial, sans-serif;
  font-size:10pt;
}
p {
  font-family:verdana, palatino,  georgia,  arial, sans-serif;
  font-size:10pt;
}
p.quote {
  font-family:verdana, palatino,  georgia,  arial, sans-serif;
  font-size:10pt;
  border: solid thin black;
  background: #5A637B;
  padding: .75em;
}
h1, h2, h3 {
  margin: 0px;
  padding: 0px;
}
ul {
  list-style-image: url("images/corporate-bullet.png")
}
...
/* humanresources.css */
ul {
  list-style-image: url("images/people-bullet.png")
}
```

The humanresources.css sheet contains only the style definitions that differ from the corporate.css sheet; in this case, only a definition for ul elements (using the different bullet). The two sheets are linked to the HR documents using the following <link> tags:

```
<head>
  <link rel="stylesheet" type="text/css" href="corporate.css" />
  <link rel="stylesheet" type="text/css" href="humanresources.css" />
</head>
```

Note

When a user agent encounters multiple styles that could be applied to the same element, it uses CSS rules of precedence, covered at the end of this section. ■

Likewise, other departments would have their own style sheets and their documents would link to the corporate and individual department sheets. As another example, members of the Engineering department might use their own style sheet and declare it in the head of their documents:

```
<head>
  <link rel="stylesheet" type="text/css" href="corporate.css" />
  <link rel="stylesheet" type="text/css" href="engineering.css" />
</head>
```

Furthermore, individual HTML elements can contain styles themselves:

```
<ul style="list-style-image: url("images/small-bullet.png");" >
```

Note

Styles embedded in elements take precedence over all previously declared styles. ■

CSS refers to the location of declarations as follows:

- **Author origin** — The author of a document includes styles in a style section or linked sheets (via <link>).
- **User origin** — The user (viewer of document) specifies a local style sheet.
- **User Agent origin** — The user agent specifies a default style sheet (when no other exists).

Tip

Styles that are critical to the document's presentation should be coded as important by placing the text `!important` at the end of the declaration. For example, the following style is marked as important:

```
.highlighted { color: blue  !important; }
```

Such styles are treated differently from normal styles when the style to use is determined from the cascade — styles coded as important override the next level of precedence when being evaluated. ■

The CSS standard uses the following rules to determine which style to use when multiple styles exist for an element:

1. Find all style declarations from all origins that apply to the element.
2. For normal declarations, author style sheets override user style sheets, which override the default style sheet. For `!important` style declarations, user style sheets override author style sheets, which override the default style sheet.
3. More specific declarations take precedence over less specific declarations.
4. Styles specified last have precedence over otherwise equal styles.

Summary

This chapter covered the basics of CSS — how styles are attached to a document, how they are best used, the different levels of CSS, and how the "cascade" in Cascading Style Sheets works. You learned the various ways to embed and define styles, and more about the separation between content and formatting that CSS can provide. The next chapter delves into the ins and outs of style definitions. Subsequent chapters in this part of the book will show you how styles are best used with various elements.

CHAPTER 26

Style Definitions

By this point in the book, you should recognize the power and versatility that styles can bring to your documents. You have seen how styles can make format changes easier and how they adhere to the content versus formatting separation. Now it's time to learn how to create styles — the syntax and methods used to define styles for your documents.

The Style Definition Format

CSS style definitions all follow the same basic format. A definition starts with a selector expression used to match elements within the documents(s), and is followed by one or more style properties and value sets. Roughly, this format approximates the following structure:

```
selector {
  property: value(s);
  property: value(s);
  ...
}
```

The `selector` is an expression that can be used to match specific elements within HTML documents. Its simplest form is an element's name, such as `h1`, which would match all `h1` elements. At its most complex, the selector expression can be used to match individual sub-elements of particular elements or to specify text to include before or after matched elements.

Cross-Ref

Selectors are covered in depth within the next section of this chapter. Acceptable property values are covered in Chapter 27. Individual CSS properties are covered in topical chapters, Chapter 29 through Chapter 37. ■

The `property` component of a style rule specifies which properties of the element the definition will affect. For example, to change the color of an element you would use the `color` property. Note that some properties affect only one aspect of an element, whereas others combine several properties into one declaration. For example, the `border` property can be used to define the width, style, and color of an element's border — and each of the properties (width, style, color) has its own property declarations as well (`border-width`, `border-style`, and `border-color`).

The `values(s)`component of a style rule contains one or more values that should be assigned to the properties. For example, to specify an element's color as red, you would use `red` as the value for the `color` property.

Now, let's look at all these elements of a style declaration in a real example. The following style definition can be used to change all the first-level headings (`h1` elements) in a document to red text:

```
h1 {
  color: red;
}
```

The actual formatting of the style declarations can vary, but must follow these rules:

- The selector or selector expression must be first.
- Braces ({ and }) must follow the selector and enclose the style property-value pairs.
- Each style property-value pair must end with a semicolon (`;`).

It is also suggested, but not absolutely necessary, that style definitions be formatted with liberal white space — spaces between elements of the definition, line breaks between property-value pairs, sub-elements indented, and so on. Feel free to add as many spaces, line breaks, and tabs as you like, as the amount of white space does not matter. What is important is that the definitions are legible.

For example, both of the following definitions produce identical results, but they are formatted quite differently:

```
h1 { color: red; border: thin dotted black; font-family: helvetica,
sans-serif; text-align: right;}

h1 {
  color: red;
  border: thin dotted black;
  font-family: helvetica, sans-serif;
  text-align: right;
}
```

Understanding Selectors

Selectors are patterns that enable a user agent to identify what elements should get what styles. For example, the following style says, in effect, "If it is a paragraph element (p), then give it this style":

```
p { text-indent: 2em;}
```

The rest of this section shows you how to construct selectors of different types to best match styles to the elements within your documents.

Matching elements by type

The easiest selector to understand is the plain element selector, as in the following example:

```
h1 { color: red;}
```

Using the actual element type (h1) as the selector causes all objects within h1 elements to be formatted with the property-value section of the definition (color: red). You can also specify multiple selectors by listing them all in the selector area, separated by commas. For example, the following definition will affect all levels of heading elements (1 through 6) in the document:

```
h1, h2, h3, 4h, h5, h6 { color: red;}
```

Matching using the universal selector

The universal selector, designated by an asterisk (*), can be used to match any element in the document. As an extreme example, you can use the universal selector to match *every* tag in a document:

```
* { color: red;}
```

Using this rule, every tag will have the color:red property-value applied to it. You would rarely want a definition to apply to all elements of a document, of course, although you can also use the universal selector to match other elements than the selector specifically defines. The following selector matches any ol element that is a descendant of a td element, which is a descendant of a tr element (an ordered list in a cell in a row of a table):

```
tr td ol  { color: red;}
```

Cross-Ref
More information on child/descendant selectors can be found in the section "Matching child, descendant, and adjacent sibling elements" later in this chapter. ■

This selector rule is very strict, requiring all three elements. If you also wanted to include descendant ol elements of th elements or ol elements occurring within p elements, you would need to specify additional selectors, or use the universal selector to match all elements that might occur between tr and ol, as in the following example:

```
tr * ol  { color: red;}
```

You can use the universal selector within any of the selector forms discussed in this chapter.

Matching elements by class

You can use selectors to match element classes. Suppose you had two areas in your document with different backgrounds, one light and one dark. You would want to use dark-colored text within the light background area and light-colored text within the dark background area. You could use light_bg and dark_bg classes in your style selector and applicable elements to ensure that the appropriate text colors were applied within the areas.

To specify a class to match with a selector, you append a period and the class name to the selector. For example, the following style will match any paragraph element with a class of dark_bg:

```
p.dark_bg  { color: white;}
```

Suppose the following paragraph were in the area of the document with the dark background:

```
<p class="dark_bg">Based on the preview we were given at Rodent Stu-
dios, Gopher Hunt promises to be a great game.</p>
```

The specification of the dark_bg class with the paragraph tag will match the defined style, causing the paragraph's text to be rendered in white.

The universal selector can be used to match multiple elements with a given class. For example, this style definition will apply to *all* elements that specify the dark_bg class:

```
*.dark_bg  { color: white;}
```

You can omit the universal selector, specifying only the class itself (beginning with a period):

```
.dark_bg  { color: white;}
```

Tip

One little-known and infrequently used trick is to give HTML elements more than one class. For example, we can give our sample paragraph both the dark_bg and bold_text classes:

```
<p class="dark_bg bold_text"> ...
```

Using this method, the tag can be influenced by two different styles: one influencing the dark_bg class and one influencing the bold_text class.

You can give HTML elements as many classes as you like, but if two class-based styles conflict, the last one listed will be used. ■

Matching elements by identifier

You can also use selectors to match element identifiers — the `id` attribute of element(s). To match identifiers, use the pound sign (#) as a prefix for the selector, followed by the `id`. For example, the following style will match any tag that has an `id` of `comment`:

```
#comment { background-color: green;}
```

Matching elements by specific attributes

You can use a selector to match any attribute in elements, not just `class` and `id`. To do so, you specify the attribute and the value(s) you want to match at the end of the selector, offset in square brackets. This form of the selector has the following format:

```
element[attribute="value"] { property: value(s);}
```

For example, if you want to match any `table` element with a `border` attribute set to `3`, you can use this selector:

```
table[border="3"]
```

You can also match elements that contain the attribute, no matter what the value of the attribute, by omitting the equal sign and attribute value. To match any `table` element that contains a `border` attribute (of any value), you can use this selector:

```
table[border]
```

Tip

You can combine two or more selector formats for even more specificity. For example, the following selector will match `table` elements with a `class` value of `datalist`, and a `border` value of 3:

```
table.datalist[border="3"] ■
```

Multiple attributes within the same selector can also be specified. Each attribute is specified in its own bracketed expression. For example, if you wanted to match `table` elements with a `border` value of `3` and a `width` value of `100%`, you would use the following selector:

```
table[border="3"][width="100%"]
```

In addition, you can match single values that are contained within a space- or hyphen-separated list value. To match a value in a space-separated list, use tilde equal (`~=`) instead of the usual equal sign (`=`). To match a value in a hyphen-separated list, you use bar equal (`|=`) instead of the usual equal sign. For example, the following selector would match any attribute that has "us" in a space-separated value of the `language` attribute:

```
[language~="us"]
```

Matching child, descendant, and adjacent sibling elements

The most powerful selector methods match elements by their relationships to other elements within a document. For example, you can specify a selector style that matches italic elements only when appearing within a heading, or list items only within ordered (not unordered) lists.

Understanding document hierarchy

The elements in an HTML document are related to each other in a hierarchical manner. The hierarchy follows the same nomenclature as family trees — ancestors, parents, children, descendants, and siblings — with the `<html>` tag as the root of the document tree. For example, consider the following, fairly simplistic document code. Figure 26-1 shows a document and its hierarchy.

```
<html>
<body>
<div class="div1">
  <h1>Heading 1</h1>
  <table>
    <tr><td>Cell 1</td><td>Cell 2</td></tr>
    <tr><td>Cell 3</td><td>Cell 4</td></tr>
  </table>
  <p>Lorem ipsum dolor sit amet, consectetuer adipiscing
  elit, sed diam nonummy nibh euismod tincidunt ut laoreet
  dolore magna aliquam erat volutpat. Ut wisi enim ad minim
  veniam, quis nostrud exerci tation ullamcorper suscipit
  lobortis nisl ut aliquip ex ea commodo consequat.</p>
</div>
<div class="div2">
  <h1>Heading 2</h1>
  <p>Lorem ipsum dolor sit amet, consectetuer adipiscing
  elit, sed diam nonummy nibh euismod tincidunt ut laoreet
  dolore magna aliquam erat volutpat. Ut wisi enim ad minim
  veniam, quis nostrud exerci tation ullamcorper suscipit
  lobortis nisl ut aliquip ex ea commodo consequat.</p>
  <ol>An ordered list
    <li>First element,</li>
    <li>Second element</li>
    <li>Third element</li>
  </ol>
</div>
</body>
</html>
```

Ancestors and descendants

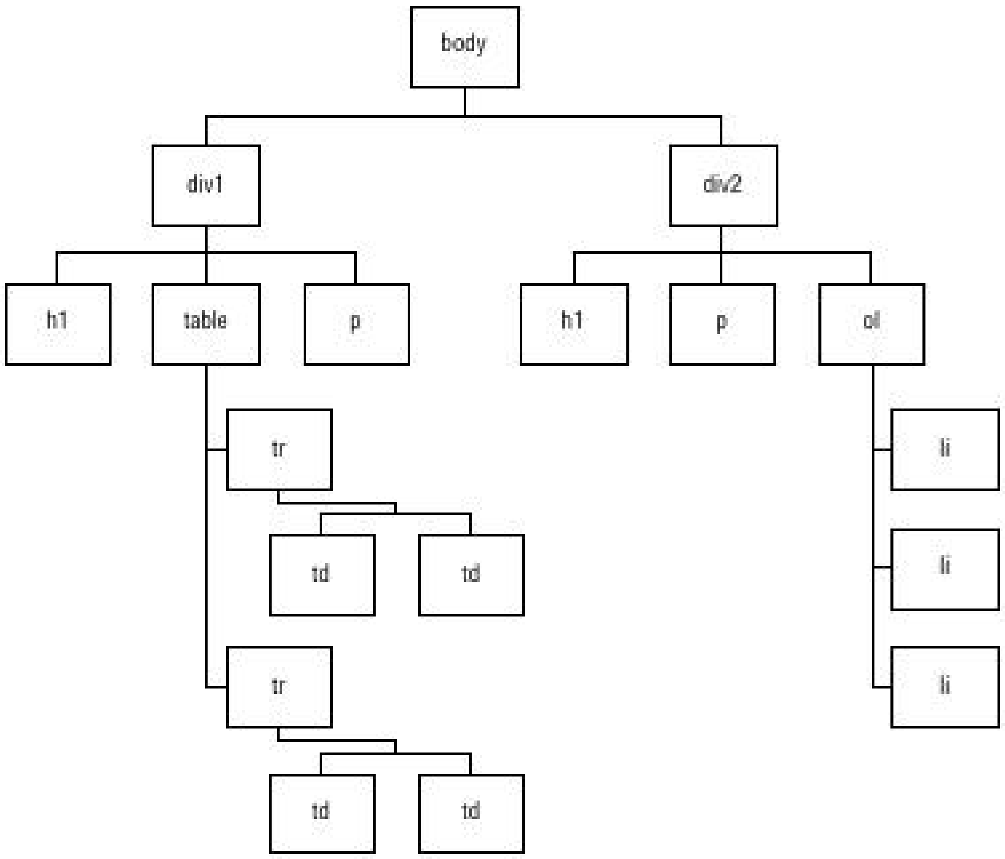
Ancestors and descendants are elements that are linked by lineage, no matter the distance between them. For example, in Figure 26-1, the list elements under `div2` are descendants of the `body` element, and the `body` element is their ancestor, even though multiple elements separate the two.

Parents and children

Parents and children are elements that are directly connected in lineage. For example, in Figure 26-1 the table rows (`tr`) under `div1` are children of the `table` element, which is their parent.

FIGURE 26-1

Diagram of a document's hierarchy



Siblings

Siblings are children that share the same, direct parent. In Figure 26-1, the list elements (li) under `div2` are siblings of each other. The header (`h1`), paragraph (`p`), and table (`table`) elements are also siblings because they share the same, direct parent (`div1`).

Selecting by hierarchy

Several selector mechanisms are available that enable you to match elements by their hierarchy in the document.

To specify ancestor and descendant relationships, list all involved elements, separated by spaces. For example, the following selector matches the `li` elements in Figure 26-1 (`li` elements within a `div` that has a class of `div2`):

```
div.div2 li
```

To specify parent and child relationships, list all involved elements, separated by a right angle bracket (>). For example, the following selector matches the `table` element in Figure 26-1 (a `table` element that is a direct descendant of a `div` element that has a class of `div1`):

```
div.div1 > table
```

To specify sibling relationships, list all involved elements, separated by plus signs (+). For example, the following selector matches the `p` element under `div1` in Figure 26-1 (a `p` element that has a sibling relationship with a `table` element):

```
table + p
```

Of course, you can mix and match the hierarchy selector mechanisms for even more specificity. For example, the following selector will match only `table` and `p` sibling elements that are also children of the `div` with a class value of `div1`:

```
div.div1 > table + p
```

Understanding Style Inheritance

Style inheritance is an important concept when working with CSS. The term *inheritance* reflects the fact that an element acquires the properties of its ancestors. In CSS, all *foreground* properties are inherited by descendant elements. For example, the following definition would result in all elements being rendered in green because every element in the document descends from the body tag:

```
body { color: green;}
```

Note that this inheritance rule is valid only for foreground properties. Background properties (background color, image, and so on) are not automatically inherited by descendant elements.

You can override inheritance by defining a style for an element with a different value for the otherwise inherited property. For example, the following definitions result in all elements being rendered with a green foreground, *except* for paragraphs with a `nogreen` class, which are rendered with a red foreground:

```
body { color: green;}
p.nogreen { color: red;}
```

Attributes that are not in conflict are cumulatively inherited by descendant elements. For example, the following rules result in paragraphs with an `emphasis` class being rendered in green, bold text:

```
body { color: green;}
p.emphasis { font-weight: bold;}
```

Using Pseudo-Classes

You have at your disposal a handful of pseudo-classes to match attributes of elements in your document. Pseudo-classes are identifiers that are understood by user agents, and they apply to elements of certain types without the elements having to be explicitly styled. Such classes are typically dynamic in nature; as such, they are tracked by means other than the static `class` attribute.

For example, there are pseudo-classes used to modify visited and unvisited anchors in the document. Using the pseudo-classes, you don't have to specify classes in individual anchor tags — the user agent determines which anchors are in which class (visited or not) and applies the style(s) appropriately in real time as the user browses.

The following sections discuss the available pseudo-classes.

Anchor styles

A handful of pseudo-classes can be used with anchor tags (`<a>`). The anchor pseudo-classes are listed in Table 26-1.

TABLE 26-1

Pseudo-Classes for Anchor Tags

Pseudo-Class	Matches
`:link`	Unvisited links
`:visited`	Visited links
`:active`	Active links
`:hover`	The link over which the browser pointer is hovering
`:focus`	The link that currently has the user interface focus

For example, the following definition will cause all unvisited links in the document to be rendered in blue, all visited links in red, and when hovered over, in green:

```
:link  { color: blue;}
:visited { color: red;}
:hover {color: green;}
```

Note
All pseudo-class definitions begin with a colon (:) and the pseudo-class name. ■

The order of the definitions is important; because the link membership in the classes is dynamic, `:hover` must be the last definition. If the order of `:visited` and `:hover` were reversed, then visited links would not turn green when hovered over because the `:visited` color attribute would override the `:hover` color attribute. Ordering is also important when using the `:focus` pseudo-class — it should be placed last in the definitions.

Pseudo-class selectors can also be combined with other selector methods. For example, if you wanted all nonvisited `a` elements with a class attribute of `boldme` to be rendered in a bold font, you could use the following code:

```
/* Add explicit "boldme" class to non-visited pseudo class */
:link.boldme { font-weight: bold;}
...
<!-- The following link is important! -->
<a href="http://something.example.com/important.html"
  class="important">An important message</a>
```

The :first-child pseudo-class

The `:first-child` pseudo-class is used to assign style definitions to the first child element of a specific element. You can use this pseudo-class to add more space or otherwise change the formatting of a first child element. For example, if you need to indent the first paragraph inside specific `div` elements (having an `indent` class), you could use the following definition:

```
div.indent > p:first-child { text-indent: 25px;}
```

This code results in only the first paragraph (`p`) element of all `div` elements having a class of `indent` being indented by 25px (pixels).

The :lang pseudo-class

The `:lang` pseudo-class is used to change elements according to the language being used for the document. For example, the French language uses angle brackets (< and >) to offset quotes, whereas the English language uses quote marks (" and "). If you need to address this difference in a document (seen by both French and English native readers), you could use a definition similar to the following:

```
/* Two levels of quotes for two languages */
.quote:lang(en) { quotes: "" "" "'" "'";}
.quote:lang(fr) { quotes: "{\ll}" "{\gg}" "<" ">";}
/* Add quotes (before and after) to quote class */
.quote:before { content: open-quote;}
.quote:after  { content: close-quote;}
```

This code would cause any element having a class of `quote` to be placed in appropriate quote characters, depending on the current language setting of the document or user agent.

Note

The pseudo-elements `:before` and `:after` are used in the preceding example to automatically place quote characters around elements. These two pseudo-classes are covered in the next section. ■

Pseudo-Elements

Pseudo-elements are another virtual construct to help apply styles dynamically to elements within a document. For example, the `:first-line` pseudo-element applies a style to the first line of an element dynamically — that is, as the first changes size (longer or shorter), the user agent adjusts the style coverage accordingly.

First line

The `:first-line` pseudo-element specifies a different set of property values for the first line of elements. This is illustrated in the following document listing and in Figure 26-2, which shows two browser windows of different widths, highlighting how the underlining of the first sentence is dynamic thanks to the pseudo-element first-line style:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
  <title>First-line formatting</title>
  <style type="text/css">
    p:first-line { text-decoration: underline;}
    p.noline:first-line { text-decoration: none;}
  </style>
</head>
<body>
<h1>The Oasis of Tranquility</h1>
<p class="noline">The Founding and Mission of The Oasis</p>
<p>Founded in 2001, The Oasis of Tranquility strives to be a
different, pleasurable experience for those seeking to get away from
their daily routine. Staffed by individuals each with specific
specialties, The Oasis provides the luxuries of many day spas, but at
salon prices. The main mission of The Oasis is that personal luxury
doesn't have to be expensive.</p>
</body>
</html>
```

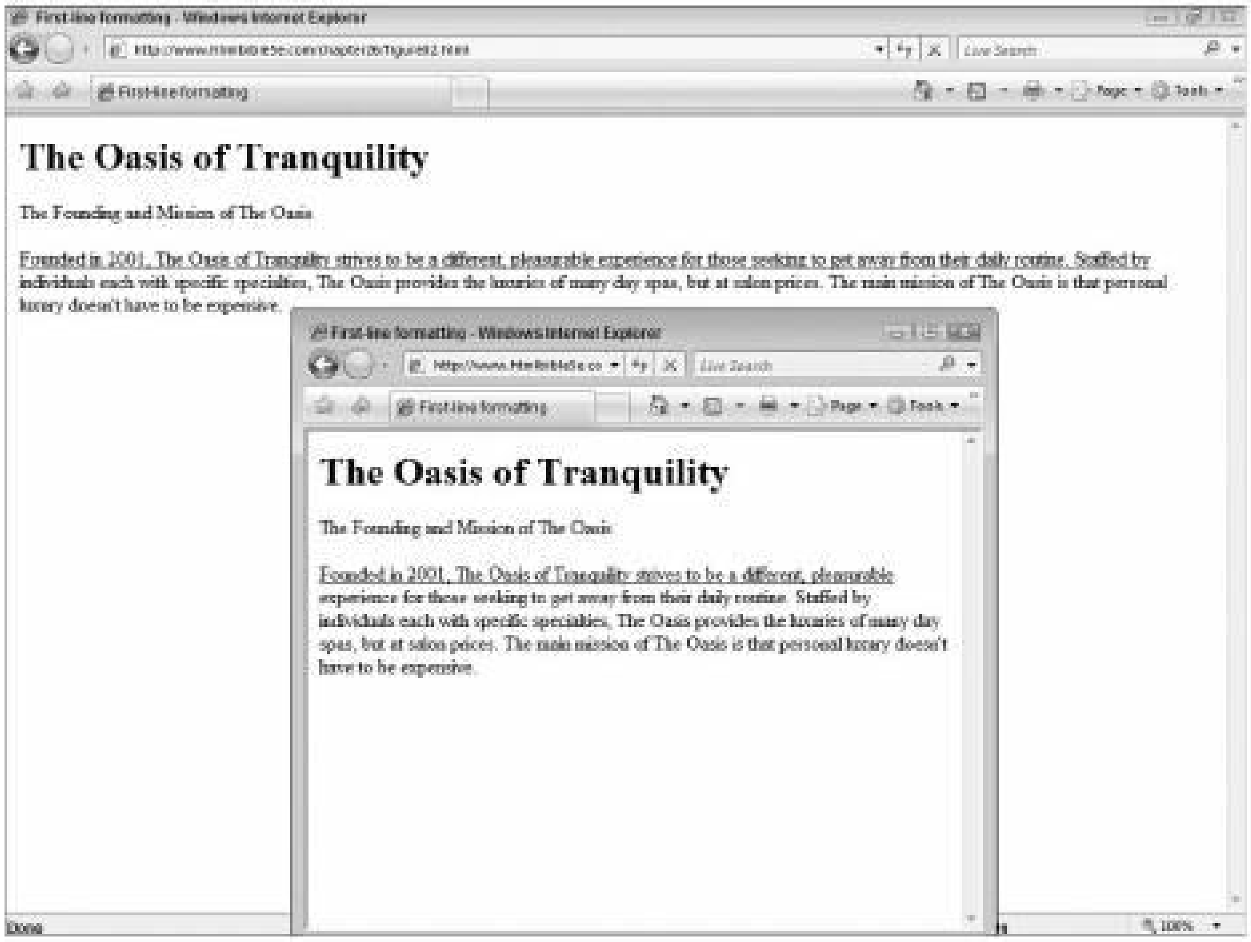
Note

The preceding code example manages element formatting by exception. Most paragraphs in the document should have their first line underlined. A universal selector is used to select all paragraph tags. A different style, using a class selector (`noline`), is defined to select elements that have a class of `noline`. Using this method, you only need to add class attributes to the exceptions (the minority), rather than the rule (the majority). ■

The `:first-line` pseudo-element can affect only a finite range of properties. Only properties in the following groups can be applied using `:first-line` — font properties, color properties, background properties, `word-spacing`, `letter-spacing`, `text-decoration`, `vertical-align`, `text-transform`, `line-height`, `text-shadow`, and `clear`.

FIGURE 26-2

You can use the first-line pseudo-element to dynamically apply properties to a paragraph's first line — when the first line's length changes, so does application of the properties.



The Oasis of Tranquility

The Founding and Mission of The Oasis

Founded in 2001, The Oasis of Tranquility strives to be a different, pleasurable experience for those seeking to get away from their daily routine. Staffed by individuals each with specific specialties, The Oasis provides the luxuries of many day spas, but at salon prices. The main mission of The Oasis is that personal luxury doesn't have to be expensive.

The Oasis of Tranquility

The Founding and Mission of The Oasis

Founded in 2001, The Oasis of Tranquility strives to be a different, pleasurable experience for those seeking to get away from their daily routine. Staffed by individuals each with specific specialties, The Oasis provides the luxuries of many day spas, but at salon prices. The main mission of The Oasis is that personal luxury doesn't have to be expensive.

First letter

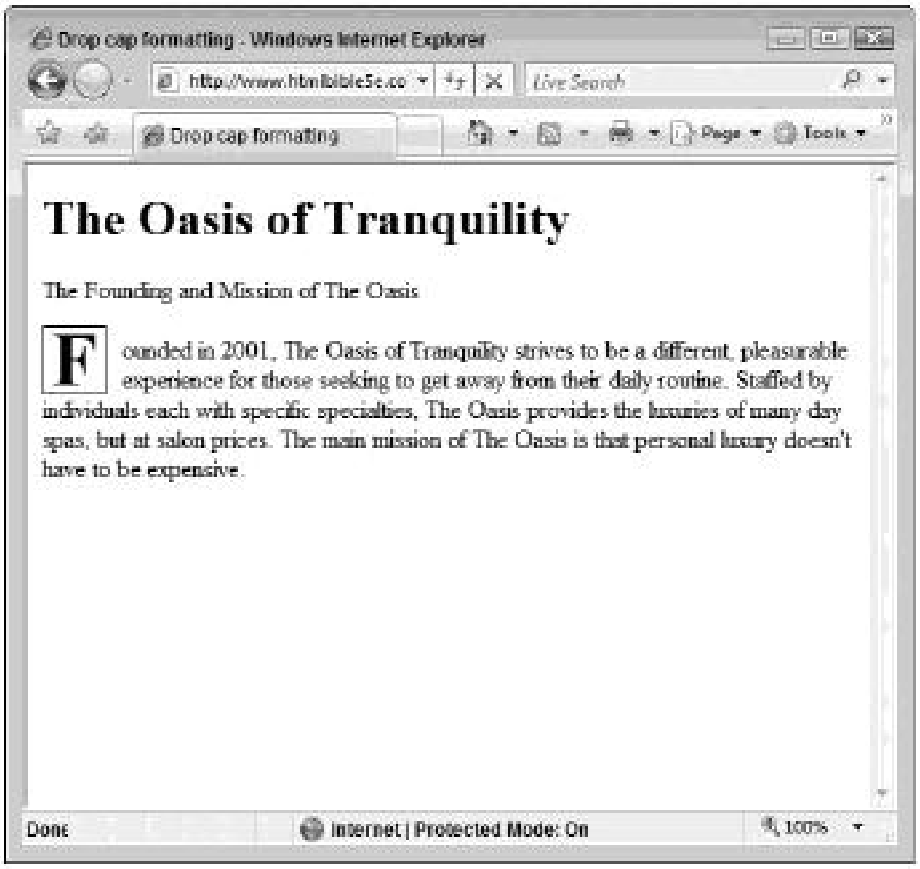
The `:first-letter` pseudo-element is used to affect the properties of an element's first letter. This selector can be used to achieve typographic effects such as drop caps, as illustrated in the following code and Figure 26-3:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
  <title>Drop cap formatting</title>
  <style type="text/css">
    p.dropcap:first-letter { font-size: 3em;
       font-weight: bold; float: left;
       border: solid 1px black; padding: .1em;
       margin: .2em .2em 0 0;}
  </style>
</head>
```

```
 <body>
 <h1>The Oasis of Tranquility</h1>
 <p>The Founding and Mission of The Oasis</p>
 <p class="dropcap"> Founded in 2001, The Oasis of Tranquility
strives to be a different, pleasurable experience for those seeking
to get away from their daily routine. Staffed by individuals each
with specific specialties, The Oasis provides the luxuries of many
day spas, but at salon prices. The main mission of The Oasis is that
personal luxury doesn't have to be expensive.</p>
 </body>
 </html>
```

FIGURE 26-3

The first-letter pseudo-element can be used for effects such as drop caps on user agents that support it.


Drop cap formatting - Windows Internet Explorer
http://www.htmlbible5e.co
Live Search
Drop cap formatting
Page
Tools
Done
Internet | Protected Mode: On
100%

The Oasis of Tranquility

The Founding and Mission of The Oasis

Founded in 2001, The Oasis of Tranquility strives to be a different, pleasurable experience for those seeking to get away from their daily routine. Staffed by individuals each with specific specialties, The Oasis provides the luxuries of many day spas, but at salon prices. The main mission of The Oasis is that personal luxury doesn't have to be expensive.

Before and after

You can use the `:before` and `:after` pseudo-elements to add additional text to specific elements. These pseudo-elements were used in the section "The :lang pseudo-class," to add quote marks to the beginning and ending of elements with a `quote` class:

```
.quote:before { content: '"';}
.quote:after  { content: '"';}
```

Notice the use of the `content` property. This property assigns the actual value to content-generating elements such as `:before` and `:after`. In this case, quote marks are assigned as the content to add before and after elements with a `quote` class. The following code and Figure 26-4 illustrate how a user agent that supports these classes generates content from the `:before` and `:after` pseudo-elements:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
  <title>Auto-quote marks</title>
  <style type="text/css">
  .quote:before { content: '"';}
  .quote:after  { content: '"';}
  </style>
</head>
<body>
<p class="quote">I set out to create a spa different from any other.
I was tired of seeing the average, every-day consumer pay top dollar
for a personal luxury service that wasn't even as good as a salon
service. I decided that with the right people and minimized overhead
I could create a day spa like experience for much less.</p>
</body>
</html>
```

Generated content breaks the division of content and presentation. However, adding presentation content is sometimes necessary to enhance the overall appeal of a document. Besides adding elements such as quote marks, you can also create counters for custom numbered lists, and other more powerful features.

Note
Some user agents have poor support for pseudo-elements. ■

Shorthand Expressions

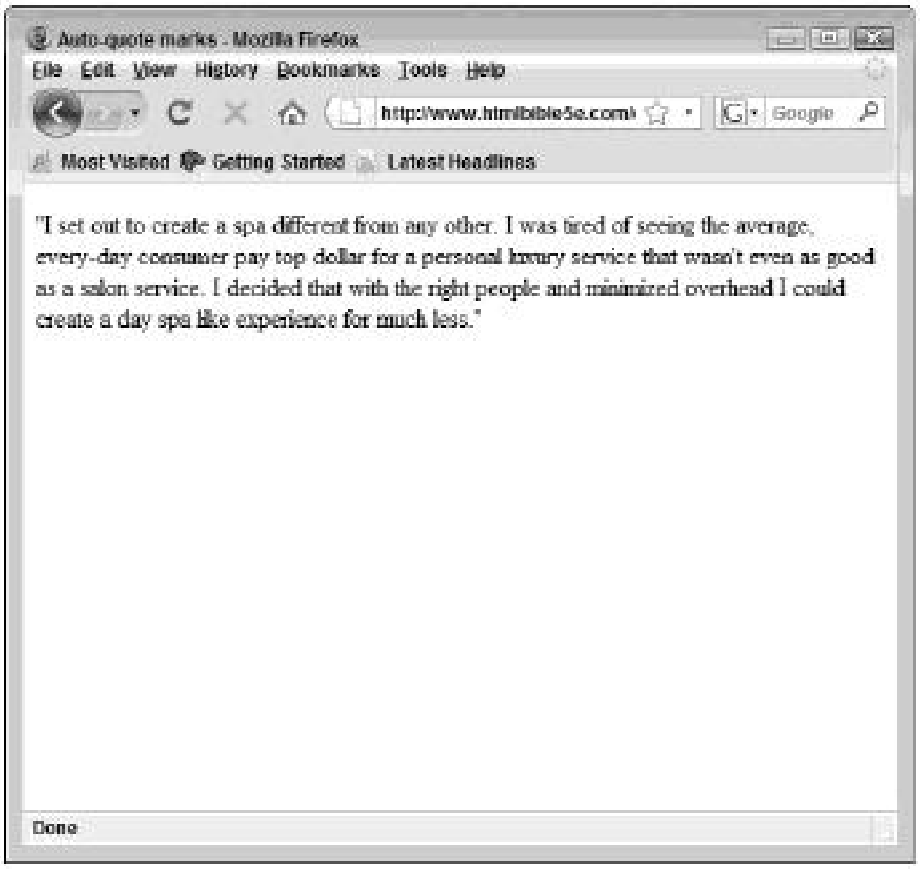
CSS supports many properties for control over elements. Many of the properties overlap or affect only slightly different areas of an element. For example, consider the following properties, all of which apply to element borders:

- `border`
- `border-collapse`
- `border-spacing`
- `border-top`
- `border-right`
- `border-bottom`
- `border-left`
- `border-color`
- `border-top-color`

- `border-right-color`
- `border-bottom-color`
- `border-left-color`
- `border-style`
- `border-top-style`
- `border-right-style`
- `border-bottom-style`
- `border-left-style`
- `border-width`
- `border-top-width`
- `border-right-width`
- `border-bottom-width`
- `border-left-width`

FIGURE 26-4

The :before and :after pseudo-elements can be used to add characters such as quote marks to text.


Auto-quote marks - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://www.htmlbible5e.com/
Google
Most Visited Getting Started Latest Headlines
Done

"I set out to create a spa different from any other. I was tired of seeing the average, every-day consumer pay top dollar for a personal luxury service that wasn't even as good as a salon service. I decided that with the right people and minimized overhead I could create a day spa like experience for much less."

Several of these properties can be used to set multiple properties within the same definition. For example, to set an element's four-sided border you can use code similar to the following:

```
p.bordered {
  border-top-width: 1px;
  border-top-style: solid;
```

```
    border-top-color: black;
    border-right-width: 2px;
    border-right-style: dashed;
    border-right-color: red;
    border-bottom-width: 1px;
    border-bottom-style: solid;
    border-bottom-color: black;
    border-left-width: 2px;
    border-left-style: dashed;
    border-left-color: red;
  }
```

Alternately, you could use the shorthand property border-*side* to shorten this definition considerably:

```
  p.bordered {
    border-top: 1px solid black;
    border-right: 2px dashed red;
    border-bottom: 1px solid black;
    border-left: 2px dashed red;
  }
```

This definition could be further simplified by use of the `border` property, which sets all sides of an element to the same values:

```
  p.bordered {
    border: 1px solid black;
    border-right: 2px dashed red;
    border-left: 2px dashed red;
  }
```

The preceding code shortens the definition by first setting all sides to the same values and then setting the exceptions (right and left borders).

Tip

As with all things code, avoid being overly ingenious when defining your styles. Otherwise, you will dramatically decrease your code's legibility. ■

Summary

This chapter explained the basics of defining styles — from the format and use of the various selector methods to the format of property declarations and setting their values. You also learned about special pseudo-classes and pseudo-elements that can make your definitions more dynamic. The next series of chapters in this book delve into specific style use for text, borders, tables, and more.

CHAPTER

27

CSS Values and Units

CSS is a rich language offering property-based control over many aspects of your HTML documents. Because the many aspects of the document differ from one another, several different types of metrics must be available — scales of units, such as inches, picas, and such — to be able to adequately apply values to their properties. This chapter covers the various types of metrics available in CSS and some reasoning regarding where and why to use each.

Cross-Ref

This chapter summarizes the syntax of values in CSS definitions and the various metric values available in CSS. For specific uses of each metric with each property, see the specific coverage of the individual properties in Chapters 29 through 35. ■

General Property Value Rules

A style definition's property/value section is contained within the braces of the definition. For example, in the following definition, `border-width: 3pt` is the property/value clause:

```
p.bordered  { border-width: 3pt; }
```

In this clause, the property (`border-width`) and value (`3pt`) are separated by a colon and the clause is terminated by a semicolon.

Tip

Technically speaking, the last (or only) property/value clause in a style definition does not need a closing semicolon. However, it is generally good practice to include a semicolon at the end of every property/value clause. ■

The property half of the clause is fairly straightforward; it is a CSS property keyword. The property/value-separating colon comes next, terminating the property half of the clause.

The value half of the clause is a bit more complex, and a myriad of values and units can be used. However, the general structure of this half is uniform, and contains the following elements, in the following order:

1. An optional unary sign (+/-)
2. A keyword, number, function (`url`, `rgb`, and so on), or textual string
3. An optional abbreviation signifying the metric value of the value (`%`, `cm`, `pt`, and so on)

There can be no white space between the items listed; they must be one contiguous string of characters. Also note that the "optional" component of the metric abbreviation is driven by the value, not by the style coder's whim. For example, the percentage metric identifier is always necessary when you want a value specified as a percentage. Omitting the metric would create ambiguity, and most user agents would just treat the value as if it were pixels, which would not create the result you desired. However, properties such as `font-size-adjust` can have only numeric values — as such, no metric abbreviation is necessary.

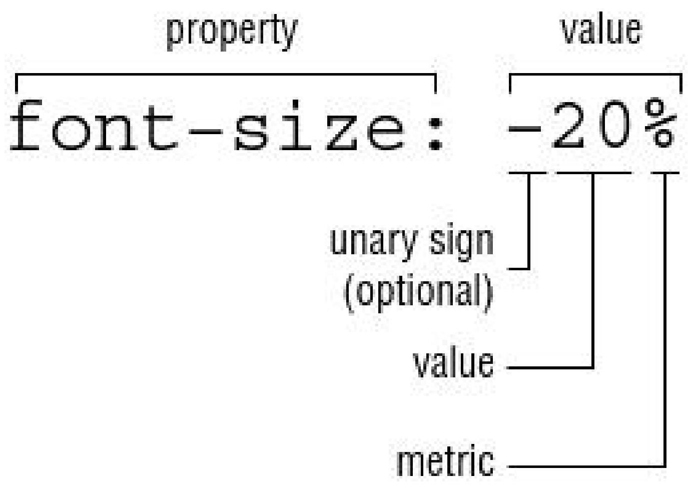
Tip

Although there can be no white space in the value of a property, there can be white space around the value. You can use this space (after the colon and before the semicolon) to format your definitions for legibility. ■

An illustrated example of the value half of a property/value clause is shown in Figure 27-1.

FIGURE 27-1

Illustrated example of a CSS property's value clause



Some properties support more than one value in a property/value statement. For example, the shortcut property `border` can specify an element's border width, style, and color all in one definition. Such a definition would resemble the following:

```
p.bordered  { border: thin solid black; }
```

To specify multiple values within one property/value statement, simply separate each value from the others using white space — one or more spaces, a line break, and so on. The list of values should still begin after the colon and should still end with a semicolon, just like single-valued statements.

Property Value Metrics

Throughout this chapter you have seen how to apply values to properties using CSS. Now let's look at the values themselves. Property values can be expressed in several different metrics according to the individual property and the desired result.

CSS supports the following metrics for property values:

- Keywords such as `thin`, `thick`, `transparent`, `ridge`, and so forth
- Real-world measures
  - inches (in)
  - centimeters (cm)
  - millimeters (mm)
  - points (pt) — The points used by CSS2 are equal to $\frac{1}{72}$ of an inch
  - picas (pc) — 1 pica is equal to 12 points
- Screen measures in pixels (px)
- Relational to font size (font size [em] or x-height size [ex])
- Percentages (%)
- Color codes (keywords, `#rrggbb`, or `rgb(r,g,b)`)
- Angles — Used with aural style sheets
  - degrees (deg)
  - grads (grad)
  - radians (rad)
- Time values (seconds [s] and milliseconds [ms]) — Used with aural style sheets
- Frequencies (hertz [Hz] and kilohertz [kHz]) — Used with aural style sheets
- Textual strings
- URLs — Links to other resources on the Web (via the `url()` function)

Which metric is used depends on the value you are setting and your desired effect. For example, it doesn't make sense to use real-world measures (inches, centimeters, and so on) unless the user agent is calibrated to use such measures or your document is meant to be printed.

The following sections cover the various unit values.

Note
As with other elements in CSS (property names, reserved names, keywords, and so on), metric abbreviations are case sensitive. Be sure to use them as shown in this chapter, mostly lowercase (`pt` for points, and so on). ■

Keyword values

Many keywords have distinct meaning when used as CSS property values. For example, you can define a border property as follows:

```
border: thin solid black;
```

In this case, `thin`, `solid`, and `black` are all keywords applied to one property. The last keyword (`black`) is a specific keyword — a color keyword — that is used in a variety of places in Web coding, including HTML attribute values. The other two keywords, `thin` and `solid`, are used only with border-related properties.

That brings up an important point with keyword values: Most of them are valid only when used with specific properties. For example, the value `solid` is meaningless when used with the `font` property.

Cross-Ref
The various keywords available in CSS, along with the respective properties to which they can apply, are covered within Chapters 29 through 35. The keywords are covered in Appendix C. ■

One specific keyword, `inherit`, can be applied to almost any CSS property. This keyword implies that the property to which it applies should inherit the values of its parent. Of course, omitting a definition for the specific property would generally have the same effect due to the inheritance rules of CSS. However, there are times when intervening style rules might affect an object's inheritance, in which case you will need to explicitly specify that an object should inherit properties.

As an example, the following definition explicitly specifies that paragraph tags with a class of `highlight` should inherit their parent element's border properties but have their text rendered in a bold font:

```
p.highlight  { border: inherit;
               font-weight: bold; }
```

Real-world measures

CSS property values can be specified in the following real-world measures:

- Inches (in)
- Centimeters (cm)
- Millimeters (mm)
- Points (pt)
- Picas (pc)

These metrics can be used with almost any property that defines the length, width, or depth of an element. To use these metrics, you would use an appropriate numeric value with the suffix corresponding to the metric you are using. For example, to specify that an element with a class of `tall` should have a height of 2 inches, you can use the following definition:

```
.tall  { height: 2in; }
```

Note

Points and picas are traditionally font-related measures. As such, they are typically used with font-related CSS properties. However, because they both have absolute measures — a point is equal to $\frac{1}{72}$ of an inch and a pica is equal to 12 points — you can use these metrics with any property that can take a length, width, or depth measure. ■

The border properties are good examples of where real-world metrics are often used. In the following example, the style definition specifies that all `td` elements should have a 2-point top and left border and a 4-point bottom and right border:

```
td  {  border-top:    2pt;
       border-left:   2pt;
       border-bottom: 4pt;
       border-right:  4pt;   }
```

These values can be translated by several means, resulting in a proper border displayed on the user agent's screen. Typically, the user agent uses the same rules it uses for fonts, resulting in the points of a border width being the same scale as the points of a display font. This method helps keep everything in scale if the user agent employs a zoom function or other screen-scaling function — if the screen is magnified, then the fonts *and* border grow larger. Still, if you want a border in a particular size no matter the scaling on the user agent, specify your measures in screen metrics (pixels).

Specifying metrics in real-world measures is especially important in documents that are meant to be printed or that have some other real-world metric connection.

Screen measures

Any property that expresses a length, width, or depth value can be expressed in pixels. Pixels are the dots that make up a computer screen — one pixel equals one dot on screen. Pixels are expressed as a metric in style definitions by suffixing a value with px. For example, the following style definition specifies that the top border of all td elements should be 2 pixels wide:

```
td  { border-top: 2px; }
```

Pixel metrics are often used to ensure that a user agent's screen renders a document to an exact size. Because the pixel metric is not relative and directly influences the screen, it can be a powerful tool. However, it can have undesired side effects if the user agent screen is an odd size or it employs any type of display scaling mechanism. In the latter case, your specified pixel metric elements may not scale appropriately.

Note

Many user agents assume that a property's value is given in pixels if no other metric is specified. This means that the following two definitions will typically produce the same results:

```
p.px_specified       {  border: 2px;  }
p.px_notspecified    {  border: 2;  }
```

This is not always the case, however, and should not be relied upon; some user agents may ignore your styles if they do not include proper metric identifiers. Always specify the proper metrics for your property values to ensure that they are rendered properly on screen. ■

Relational measures

Three metrics can be used to specify that a value should be set in relation to another value. For example, the following definition sets all table widths to half (50 percent) of their parent element's width:

```
table  { width: 50%; }
```

The other two metric specifications — em and ex — refer to font sizes. The em metric refers to the height of the current font, whereas the ex metric refers to the height of the letter *x* in the current font (generally half the font's height). These two metrics specify their values as related to the current element's font-height property value. However, if either metric is used to specify an element's font-height property, the value specified is related to the parent's font-height property. The percentage metric (%) always relates to the parent's properties.

That last statement, "always relates to the parent's properties," is an important one. Many people assume that a table whose width is set to 50 percent will span half the user agent screen because the parent element of the table in question spans the entire screen width. But what happens when that isn't the case, when the parent element is not 100 percent as wide as the user agent?

Consider the following example and Figure 27-2, which shows how a table with a width specified as 50 percent can be a smaller fraction (25 percent) as wide as the user agent if its parent element is set to a smaller percentage width (50 percent):

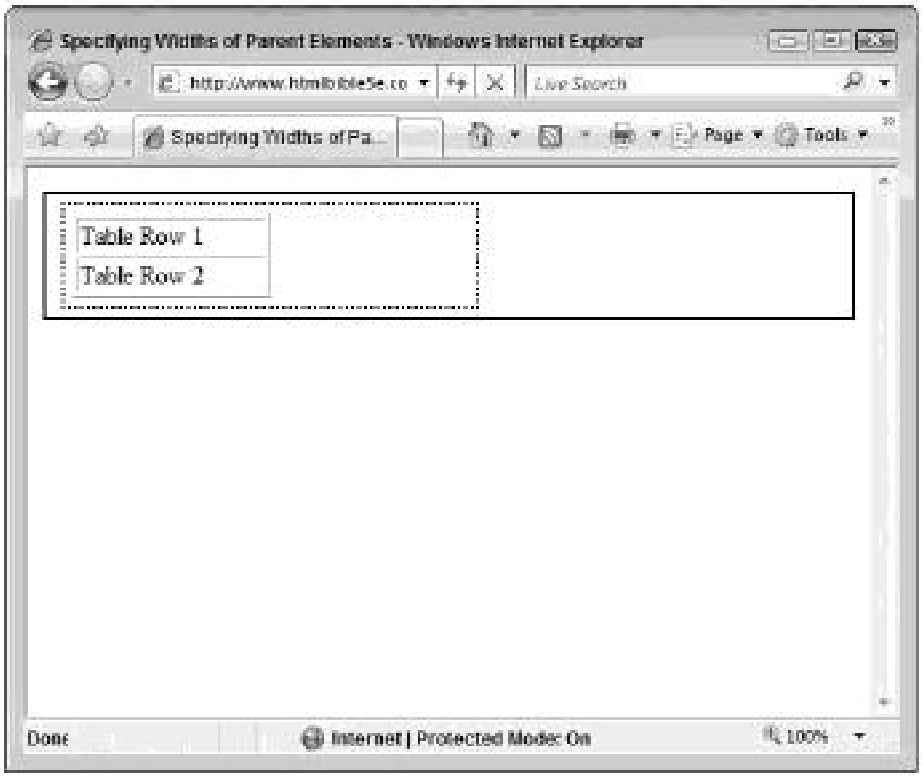
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Specifying Widths of Parent Elements</title>
<style type="text/css">
  body   { border: thin solid black;
           padding: 5px; }
  div    { width: 50%;
           border: thin dotted black;
           margin: 5px;
           padding: 5px; }
  table  { width: 50%; }
</style>
</head>
<body>
<div>
  <table border="1">
    <tr><td>Table Row 1</td></tr>
    <tr><td>Table Row 2</td></tr>
  </table>
</div>
</body>
</html>
```

The elements in the preceding example were given borders and extra padding to make the borders more evident. Using the borders, it is easy to see how the `body` element stretches to 100 percent of the user agent window's width, the `div` element occupies 50 percent of the `body` element's width (and hence 50 percent of the window), and the `table` element occupies only 25 percent of the window (50 percent of the `div`). In short, the `table` element is only 25 percent of the window because its `width` property is relative to its parent element's `width` property, not the user agent's window width.

The em and percent metrics can be quite powerful, specifying a value that changes as the element sizes around it change. The em and percent metrics are best used when you need a relational, not absolute, value.

FIGURE 27-2

The percentage metric always relates to the element's parent properties.


Specifying Widths of Parent Elements - Windows Internet Explorer
http://www.htmlbible5e.co
Live Search
Specifying Widths of Pa...
Page
Tools
Table Row 1
Table Row 2
Done
Internet | Protected Mode: On
100%

Color and URL functions

Two functions provide `color` and `url` values to properties.

The `rgb()` function (red, green, blue) takes three arguments to provide a color to a property via the mix of the values given. The arguments are values of the colors red, green, and blue to be mixed together to create the color desired. The values themselves are decimal numbers between 0 and 255 or percentage values between 0 and 100. In either case, 0 signifies no amount of the color specified, and the highest number (255 or 100 percent) is the most mix of the color specified. For example, consider the following:

```
border-color: rgb(0,0,0);        /* black (absence of color) */
border-color: rgb(255,255,255);  /* white (full mix of all colors) */
border-color: rgb(100%,0%,100%); /* purple (red and blue mix) */
border-color: rgb(100%,0%,0%);   /* red (100% red only) */
border-color: rgb(255,125,0);    /* orange (red and green mix) */
```

The `rgb()` function can only be used where a color value can be provided, such as for a `color`, `background-color`, or `border-color` property. Keep in mind that the HTML-style, hexadecimal shorthand method of specifying color values is also available in CSS. This method has the following syntax:

```
#rrggbb
```

The letters signify hexadecimal digits for the colors red (rr), green (gg), and blue (bb). Using the two digits you can specify values between 0 and 255, as in the rgb() function. For example, the following is an example of the value orange, mixing a value of 255 red with 125 green:

```
border-color: #FF7D00;
```

The url() function is used to specify a URL — the Web or Internet address of a resource — to a property. The syntax of this function is very simple; the URL is encased between the parentheses in optional quote marks. For example, the following code specifies a background image for a table found in the images directory of the current site:

```
background-image:  url("images/star-background.jpg");
```

Keep in mind that the URL specified does not have to be a local path. You can specify a resource located on another server as long as it is accessible to the user agent. For example, this code specifies a similar image but one located on a remote server:

```
background-image:
url("http://www.on-target.com/images/star-background.jpg");
```

The URL can be encapsulated by single or double quotes. Although technically the quotes are optional, it's always a good idea to include them to help the parsing of your URL string. Note that, in any case, parentheses, commas, white space characters, single quotes, and double quotes must be escaped if they appear in the URL. To escape a character, simply prefix it with a backslash. For example, \ would adequately escape a comma.

Aural metrics

Several metrics are used with aural style properties, such as azimuth and pitch. These properties control the position and properties of sounds generated via aural style sheets. Metrics that can be used with such properties include the following:

- Angles — Used with positioning properties
  - degrees (deg)
  - grads (grad)
  - radians (rad)
- Time values (seconds [s] and milliseconds [ms]) — Used with sound properties
- Frequencies (hertz [Hz] and kilohertz [kHz]) — Used with sound properties

Each metric is used by specifying a decimal number and suffixing it with the appropriate suffix for the metric. For example, to specify an azimuth of 120 degrees, you can use the following property/value statement:

```
azimuth: 120deg;
```

Tip

For a good overview of aural style sheets, including a breakdown of browser support, visit `http://lab.dotjay.co.uk/tests/css/aural-speech/`. ■

Summary

There are almost as many metrics in CSS as there are properties. This has to do with the rich nature of the technology and its ability to influence many aspects of a document, because each characteristic of a document can use certain metrics, and the properties of these metrics are related to the characteristic where they're used. This may seem confusing at first, but the next few chapters help connect distinct metrics with distinct properties. In no time you will settle into your favorite metrics, and their use will become habit.

The next chapter covers the very important topics of cascade and inheritance, or how CSS styles and properties affect one another. From there, the other CSS chapters are broken into CSS topic areas — fonts, text, tables, and so on.

CHAPTER 28

CSS Inheritance and Cascade

IN THIS CHAPTER

Inheritance

Cascade

Specificity

The words "inheritance" and "cascade" are bandied about a lot in regard to CSS. They are often used interchangeably. However, they each have a unique style-related meaning. This chapter clarifies these terms and their meanings in CSS.

Inheritance

The word "inheritance" is defined by *Webster's Dictionary* as "a) the act of inheriting property; b) the reception of genetic qualities by transmission from parent to offspring; c) the acquisition of a possession, condition, or trait from past generations." This definition is accurate for the behavior of HTML elements controlled by CSS — child elements inherit the properties of their parents.

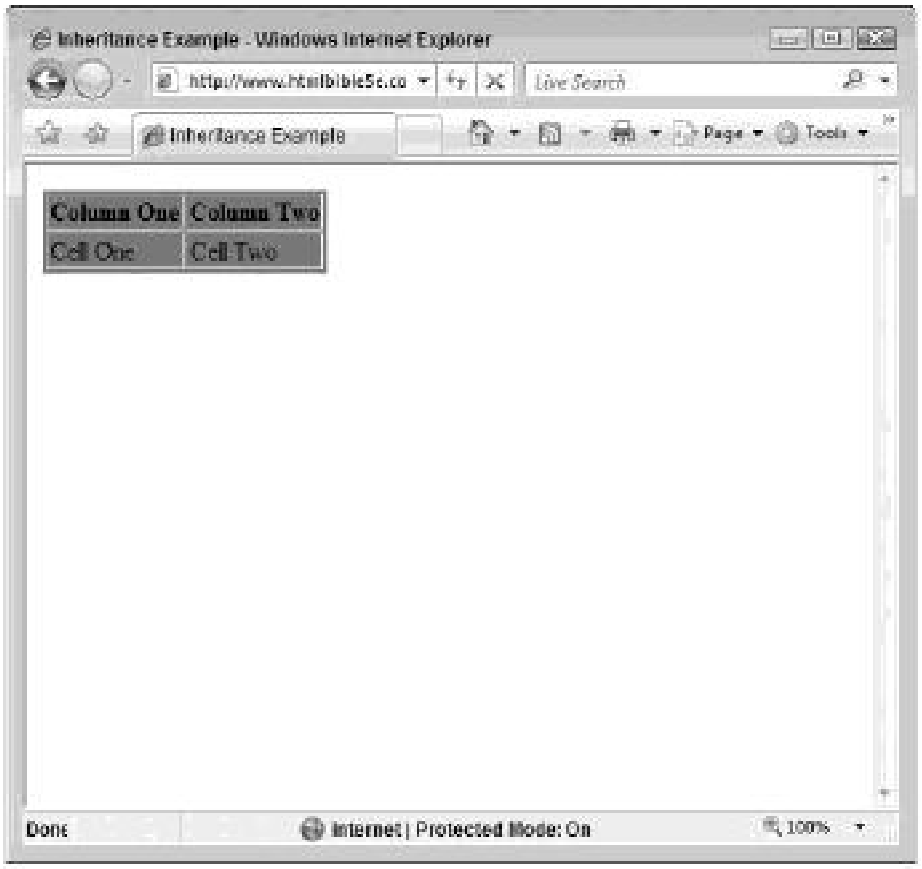
For example, consider the following document, whose output is shown in Figure 28-1:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
  <title>Inheritance Example</title>
  <style type="text/css">
    table { background-color: red; }
  </style>
</head>
<body>
<table border="1">
  <tr>
```

```
        <th>Column One</th>
        <th>Column Two</th>
      </tr>
      <tr>
        <td>Cell One</td>
        <td>Cell Two</td>
      </tr>
    </table>
    </body>
    </html>
```

FIGURE 28-1

The table rows and cells inherit the table's background-color property.


Inheritance Example - Windows Internet Explorer
http://www.htmlbible5e.co
Live Search
Inheritance Example
Page
Tools
Done
Internet | Protected Mode: On
100%

Column One	Column Two
Cell One	Cell Two

The table's `background-color` definition applies to all table elements (`table`), all table row elements (`tr`), and all table cell elements (`th` and `td`) in the document.

The rows and cells are also colored red because they are child elements of the table and inherit the table's background color property. The main body of the document, the parent of the table, does not inherit the background color because it is a parent of the table, not a child.

Inheritance can be more complex. For example, consider the following style definitions:

```
p            { border: thin solid black; }
p.redbottom  { border-bottom: thin dotted red; }
```