chapter 8

Creating Active Web Pages

"Their behavior, like the behavior of anything created by a computer program, is limited only by the programmer's imagination."

—Sherry Turkle, author of Second Self: Computers and the Human Spirit

In this chapter, you will learn how to:

- List the primary scripting languages, understand the role of JavaScript, and write a script that uses variables to display dynamic messages onscreen.
- Understand the purpose of the document object model (DOM), describe the most popular JavaScript DOM objects, and use dot notation to access those objects in a script.
- Understand the concept of a cookie and write a script to maintain the user's state by setting and reading the values of cookies as the user interacts with your site.
- List the three kinds of cascading style sheets and explain when it is appropriate to use them on a Web page.
- Define how Dynamic HTML enables you to create animated Web pages by combining HTML with style sheets and scripts.
- Define and understand the purposes of and relationships among XML, XSL, XSLT, XHTML, and SMIL.

WEB developers make a distinction between static and active Web pages. A **static Web page** is a document in which the content is fixed in HTML codes that make the document always read the same when viewed in a browser. On the Internet are billions of static Web pages, which are appropriate for presenting documents whose contents do not change. The Declaration of Independence, for example, is a historical document whose content will never change. It is therefore appropriate for the National Archives to present this document in the static Web page at www.archives.gov/national_archives_experience/declaration_transcript.html.

Static Web pages, however, do not unleash the potential of the Web to provide users with custom screens generated just in time according to the specific needs of the user. Enter the **active Web page**, which uses the browser's window as a display surface through which the user can interact with dynamic objects onscreen. These objects can be powered by either client-side or server-side processes.

This chapter teaches client-side techniques you can use to create active Web pages that run in the browser window without requiring any server-side processing. First, you learn how to use the JavaScript language to write dynamic messages onscreen. Second, you study the document object model (DOM), which defines many kinds of objects you can manipulate onscreen. Third, you learn how cookies enable scripts to remember things from screen to screen. Fourth, you use cascading style sheets (CSS) to stylize the appearance of objects onscreen. Fifth, you use Dynamic HTML to bring these objects to life onscreen. Last, but certainly not least, you learn how to use an XML module called SMIL to caption a video and create other kinds of multimedia effects onscreen.

Everything you learn in this chapter executes on the client side, meaning that you do not need access to a Web server to do these things. Chapter 12 shows how server-side scripts use databases to enable users to interact with data-driven Web applications onscreen.

ntroduction to Scripting

Scripting is the act of writing little computer programs that can enhance the appearance and functionality of a Web page. Browsers render Web pages by placing objects onscreen. Scripts let you grab hold of those objects to make them do things. You can grab hold of the browser's status

bar, for example, and write a message into it. You can make text on your Web pages display active content, such as the current date and time. You can create rollover effects that highlight a graphic, make sounds, and pop out explanatory messages when the user mouses over an object onscreen. When the user clicks a button to submit something typed into a text field, you can grab hold of the text field and validate what the user entered. You can also write scripts that use cookies to remember things as the user navigates from screen to screen. Later in this book, Chapter 12 provides you with scripts that can store, query, and update information in server-side databases.

What Scripting Languages Are There?

There are many brands of scripting languages. Most well known is **JavaScript**, the language that runs client-side in the browser without requiring any server-side processing. That is why this chapter teaches you how to use JavaScript.

In Chapter 12, which teaches server-side scripting, you have your choice of programming in VBScript or JScript, Microsoft's Active Server Page (ASP) languages. All of these languages are similar in that they can store values in variables, manipulate values programmatically, use IF-THEN statements to make decisions, receive information from HTML forms, and print text or graphics to communicate with end users onscreen. JavaScript and JScript share much of the same syntax. You will find, therefore, that the JavaScript programming you learn in this chapter helps you learn JScript more quickly when you get to Chapter 12. Due to the popularity of Visual Basic (VB), Chapter 12 also provides VBScript versions of each tutorial example, thereby allowing you to choose to program in either JScript or VBScript.

Other popular server-side languages include C#, Java, and J#, Microsoft's version of Java. The C# and Java programming languages are for more advanced application development that is beyond the scope of this book. If you do well in the JavaScript and JScript sections of this book, however, you may well have the potential to become a successful application developer in one of the more advanced languages.

Where Do Scripts Go?

You can put JavaScript in the head or in the body section of a Web page. Scripts can also reside in a separate file called an **include file**, which gets included in the page at runtime. If you do a lot of scripting, an include file can save you time. If you need to revise a script, for example, you can make the change once in the include file instead of changing it on each page that uses the script.

If a script is brief and is not used a lot, you can simply type it into the body of the page. If you find yourself typing the same code often, however, it is better to put it inside a reusable function that goes into the head section of the Web page. A **function** is a named procedure you can call upon by name any time you need to execute the code in the function. When you call the function, you can pass to it one or more parameters that preset the values of variables the function manipulates. When the function finishes executing, it can return values to the script that called it.

Functions you find yourself using on many pages should be put into an include file. If you copy the code of a frequently used function into the heads of many Web pages, on the other hand, you will have many Web pages to modify if you need to update something in that code.

Where Do Scripts Run?

Scripts run either on the client (i.e., in the browser) or on the server that hosts the Web site. JavaScript is an example of a scripting language that runs on the client. When a browser encounters JavaScript on a Web page, the browser processes the script and renders the result onscreen. ASP scripts, on the other hand, run on the server. When a browser asks a server to display an ASP page, the server executes any JScript or VBScript on the page before sending the response to the browser.

When Should You Use JavaScript?

You should use JavaScript whenever the process you are handling can be accomplished by the browser without requiring any programming on the server side. Handling these processes on the client side reduces the number of round trips between the client and the server, thereby saving valuable bandwidth and speeding response time.

Certain effects can be accomplished only through the use of client-side scripting. Animated image effects, for example, happen on the client side. In the Dynamic HTML section of this chapter, therefore, you will use JavaScript to create animated effects onscreen.

Some situations use both client-side and server-side scripting. When a user enters information into a form, for example, you can use JavaScript to check whether the user has entered valid information before submitting the form data to the server. This eliminates the round-trip that would otherwise be required for the server to tell the user the form was incomplete.

JavaScript "Hello, World!" Project

The simplest script is the quintessential Hello, World! example that beginners typically create as their first programming project. The purpose of the Hello World! script is to print the words Hello, World! onscreen. Working through this example will teach you how to use the `<script>` start and `</script>` stop tags and the JavaScript `document.write()` method for printing information onscreen. Follow these steps:

1. Pull down the Notepad's File menu and choose New to create a new file. Into this new file, type the following code:

```
<html>
<head>
      <title>Hello, World! Project</title>
</head>
```

```
<body>
<script language="JavaScript">
document.write("Hello, World!");
</script>
</body>
</html>
```

2. Pull down the Notepad's File menu and choose Save As. When the Save dialog appears, save this file under the filename *Hello.html*. Then open the file with your browser. Figure 8-1 shows the completed Notepad file, and Figure 8-2 shows the browser displaying the Hello World! greeting onscreen. If you do not see this greeting onscreen, compare your code to the listing in Figure 8-1, correct any typographical errors, save the file, and click your browser's Refresh button to view the corrected code.
3. Reflect on the Hello, World! script you just created. Notice that the `<script>` tag uses the `language="JavaScript"` attribute to tell the browser you are programming in JavaScript. Observe that the `document.write()` command prints to the screen the string of characters between the quotation marks, which delimit the characters in a string.
4. Besides writing predetermined strings of characters onscreen, the `document.write()` command can also display dynamic content onscreen. The next part of this tutorial teaches you how to do that. Before proceeding, however, study the following code analysis to make sure you understand the concepts in the Hello, World! script.

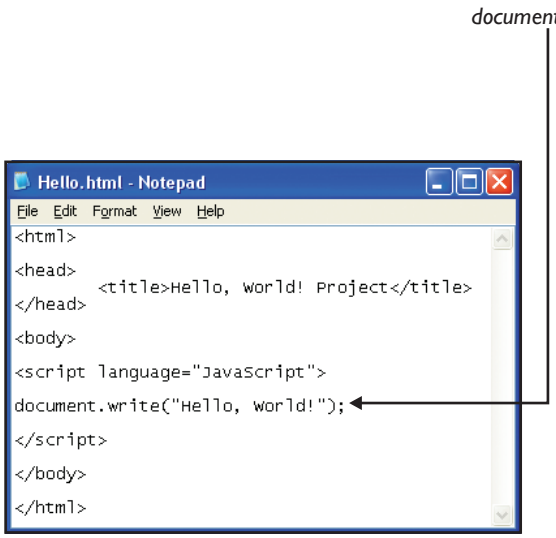


FIGURE 8-1 *In the Notepad view of the Hello.html file, the document.write() command programs the script to write "Hello World!" onscreen.* ■

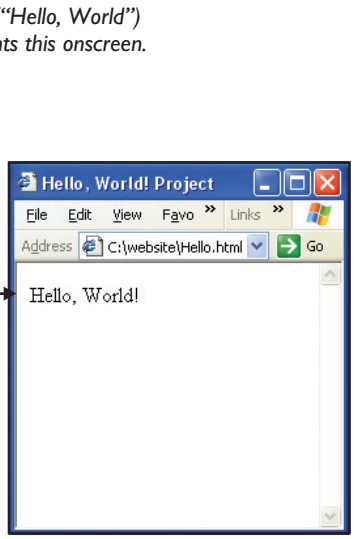
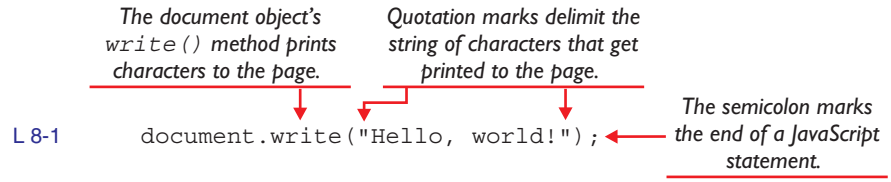


FIGURE 8-2 *In the browser view of the Hello.html file, the browser executes the script in Figure 8-1, and you see "Hello, World!" onscreen.* ■

Hello, World! Code Analysis

The Hello, World! script begins with the `<script>` start tag that you use to mark the beginning of a script on a Web page. The start tag identifies JavaScript as the scripting language via the *language* attribute: `<script language="JavaScript">`. You type the script between the start tag and the `</script>` stop tag. The script consists of one instruction, which writes Hello, world! onscreen. The script writes this message onscreen by invoking the *write* method of the *document* object:



The `write()` method causes a string of characters to be written onscreen. The string to be written onscreen appears inside the parentheses and must be delimited with quote signs, which demarcate the characters in a string. The command concludes with a semicolon, the termination character that ends a JavaScript statement.

The *document* object is one of dozens of JavaScript objects available for you to use in your scripts. Most objects contain methods that can do things for you and properties that expose the current settings of the object. You learn a lot more about objects, methods, and properties as this tutorial continues.

Variables and Assignments

We live in a world full of things that change. Variables enable scripts to process things that change. In a script that balances a checkbook, for example, the bottom line changes every time the user writes a check. Because the balance varies, it cannot be hard-coded into the script. Instead, the script must compute the balance and store the result somewhere. This tutorial teaches you how to store things in variables. After learning how to assign values to variables, you find out how to print their values onscreen. Through a process called concatenation, you learn how to combine variables into strings. By including style tags in the strings, you can display the values of your variables onscreen with style. To put variables and assignments to practical use, you learn how to display the current date and time onscreen by concatenating variables that contain the current year, month, day, hour, minute, and second.

What Is a Variable?

In computing, a **variable** is a place in the computer's RAM that remembers, or stores, the value of something changeable. It is called a variable because its value is subject to change.

What Is a String Variable?

A **string** is a sequence of one or more alphanumeric characters. A **string variable** is a place in computer memory that remembers, or stores, the alphanumeric characters in a string.

What Is a Numeric Variable?

A **numeric variable** is a place in computer memory that remembers, or stores, a number. In a script, the numbers can be integers or floating point. An **integer** is a whole number with no decimal point. A **floating point** number has a decimal point with one or more numbers after the decimal point.

What Is a Variable Name?

A **variable name** is the identifier used to refer to, or call upon, a place in computer memory that stores the value of a variable. When you write scripts that use variables, you make up your own names for the variables. When you name variables, I recommend that you follow a naming convention whereby integers begin with the letter *i*, strings begin with the letter *s*, and floating point numbers begin with the letter *f*. This book follows these naming conventions. Thus, a string that holds the user's first name might be named *sFirstName*, and an integer that holds the user's ID number might be named *iUserID*. A floating point number containing the user's account balance might be called *fAccountBalance*.

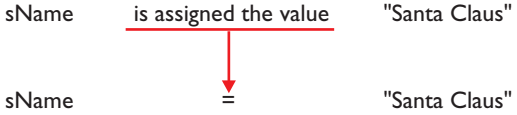
The reason you prefix the variable names with an *i*, *s*, or *f* is to help you keep track of which variables are integers, strings, or floating point numbers. If you mix them up, you can cause data-type conflict errors that make the browser display an error message when you run the script.

What Is an Operator?

An **operator** is a symbol that causes a script to perform some kind of action on a variable or a value. The most fundamental operator is the **assignment operator**, which assigns values to variables. The assignment operator uses the = symbol. You may be used to thinking of the = symbol as an equal sign. In scripts, however, the symbol = means to assign. To understand how this works, consider the following assignment statement:

L 8-2
```
sName = "Santa Claus";
```

This statement assigns the value `"Santa Claus"` to the variable called `sName`. Notice that the assignment goes from right to left. To understand this, think of the = operator as meaning *is assigned the value*. Thus, the statement `sName = "Santa Claus"` means:



Assigning Values to String Variables

The following example teaches you how to use the = operator to assign values to string variables. In this example, you create variables that hold a person's first and last name. Follow these steps:

1. Pull down the Notepad's File menu and choose New to create a new file. Into this new file, type the following code:

```
<html>
<head>
      <title>Using Variables</title>
</head>
<body>
<script language="JavaScript">
//Assign values to string variables
sFirstName = "Santa";
sLastName = "Claus";
//Write values to the screen
document.write("First Name: ");
document.write(sFirstName);
document.write("<br>Last Name: ");
document.write(sLastName);
</script>
</body>
</html>
```

2. Pull down the Notepad's File menu, choose Save, and when the Save As dialog appears, save this file under the filename *variables.html*. Then open the file with your browser. Figure 8-3 shows the completed Notepad file, and Figure 8-4 shows how the browser displays the values of the variables onscreen. If you do not see these values, compare your code to the listing in Figure 8-3, correct any typographical errors, save the file, and click your browser's Refresh button to view the corrected code.

Using Variables Code Analysis

There are two assignment statements in the *variables.html* script. First, the statement `sFirstName = "Santa"` assigns the value `"Santa"` to a string variable called `sFirstName`. Second, the statement `sLastName = "Claus"` assigns the value `"Claus"` to a string called `sLastName`. Notice that the variable names begin with *s* to remind you that they will contain strings. Also notice that the names of the variables indicate what they will hold: `sFirstName` will hold the person's first name, and `sLastName` will hold the person's last name. The reason you create variable names that suggest what content the variables will hold is to make the code self-documenting. If you choose esoteric variable names such as `sVariable1` and `sVariable2`, on the other hand, your code is harder to read and understand.

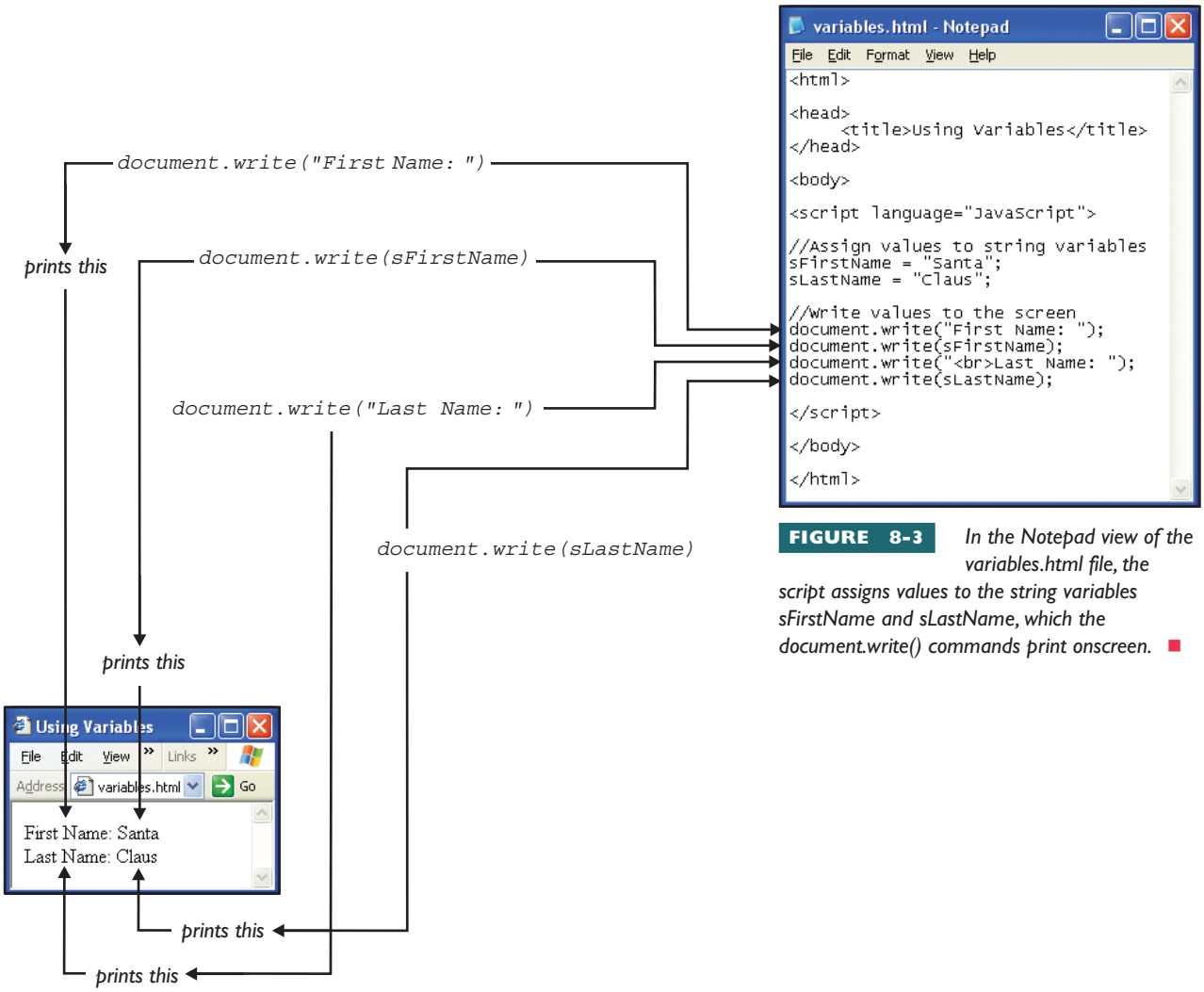


FIGURE 8-3 *In the Notepad view of the variables.html file, the script assigns values to the string variables sFirstName and sLastName, which the document.write() commands print onscreen.* ■

FIGURE 8-4 *In the browser view of the variables.html file, the browser executes the script in Figure 8-3.* ■

To make your code even easier to understand, you can include comment statements that document what your code does. Comment statements begin with the special symbol //. Two of the lines inside the *variables.html* script, for example, begin with the symbol //. Throughout this book, notice that the examples are full of comment statements. When you write scripts, you should form the habit of inserting your own comment statements to document what the code does. Later on, when you need to modify or troubleshoot problems in the code, the comment statements will save you time by making it easier for you to remember what the code is doing.

Concatenating String Variables

To **concatenate** means to join strings together via the concatenation operator. In JavaScript, the **concatenation operator** is the + sign. To learn how

to use the concatenation operator, you add some code to the *variables* script you created earlier in this chapter. Follow these steps:

1. Use the Notepad to open the *variables.html* page you created in the previous exercise.
2. Position your cursor at the bottom of the script. Add the following lines of code to the end of the script by inserting them just above the `</script>` stop tag:

```
document.write(sLastName);
//Concatenation example
sFullName = sFirstName + " " + sLastName;
document.write("<br>Full Name: ");
document.write(sFullName);
</script>
```

3. Save the file and then open it with your browser. Figure 8-5 shows the completed Notepad file, and Figure 8-6 shows how the browser displays the concatenation onscreen. If you do not see these values, compare your code to the listing in Figure 8-5, correct any typographical errors, save the file, and click your browser's Refresh button to view the corrected code.

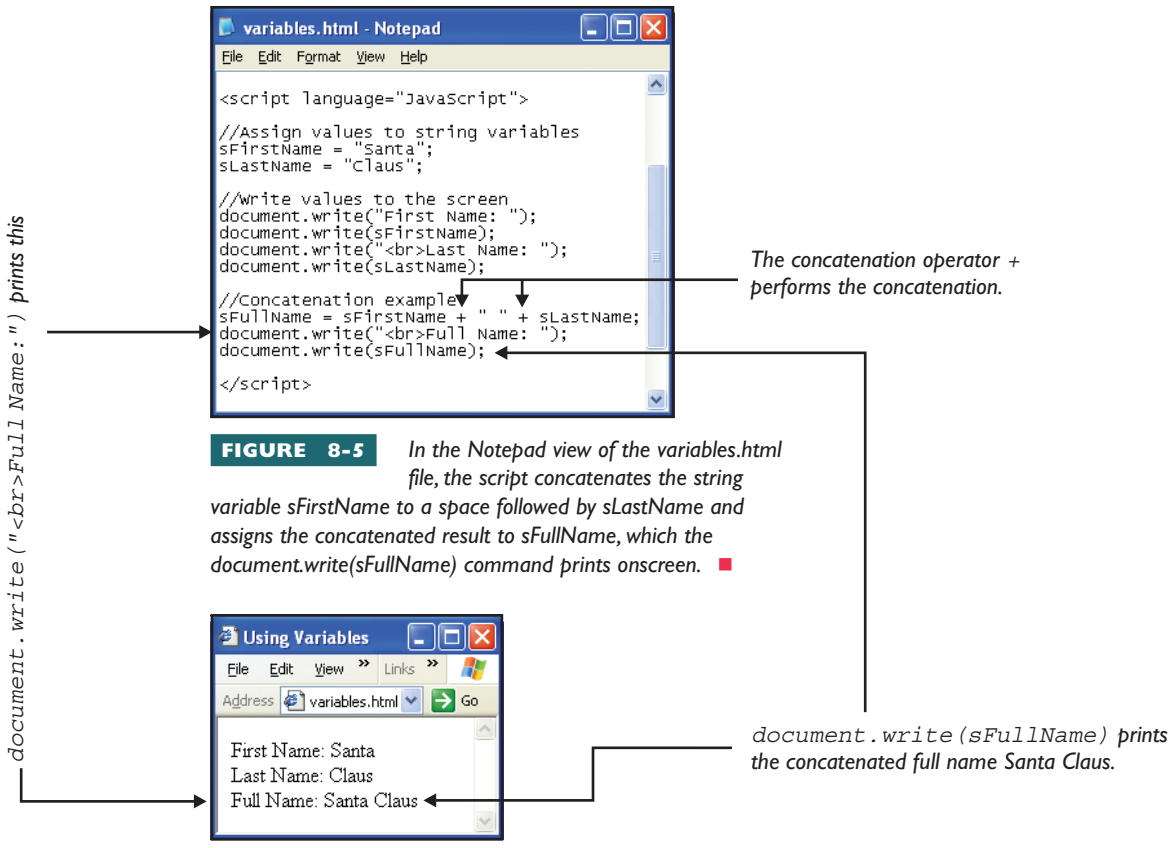
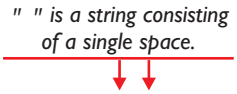


FIGURE 8-5 *In the Notepad view of the variables.html file, the script concatenates the string variable sFirstName to a space followed by sLastName and assigns the concatenated result to sFullName, which the document.write(sFullName) command prints onscreen.* ■

FIGURE 8-6 *In the browser view of the variables.html file, the browser executes the script in Figure 8-5, which concatenates the first and last names into the full name and prints it onscreen.* ■

Concatenation Code Analysis

The string variable *sFullName* is so named because it will hold the person's full name. Notice that the concatenation inserts a space between the person's first and last names:




```
L 8-3    sFullName = sFirstName + " " + sLastName;
```

Assigning Values to Numeric Variables

You use the = assignment operator to assign values to numeric variables. In this example, you create variables that hold a person's age and weight. Follow these steps:

1. Use the Notepad to open the *variables.html* page you created in the previous exercise.
2. Position your cursor at the bottom of the script. Add the following lines of code to the end of the script by inserting them just above the `</script>` stop tag:

```
document.write(sFullName);
//Assign values to numeric variables
iAge = 100;
iWeight = 350;
//Create the print string
sPrint = "<p>At age ";
sPrint += iAge;
sPrint += ", " + sFullName;
sPrint += " weighs ";
sPrint += iWeight;
sPrint += " pounds.</p>";
//Write the print string onscreen
document.write(sPrint);
</script>
```

3. Save the file and then open it with your browser. Figure 8-7 shows the completed Notepad file, and Figure 8-8 shows how the browser displays the age and weight onscreen. If you do not see these values, compare your code to the listing in Figure 8-7, correct any typographical errors, save the file, and click your browser's Refresh button to run the corrected code.

Numeric Variables Code Analysis

The age and weight are stored in variables named *iAge* and *iWeight*. You put an *i* at the beginning of these variables as a reminder that you are using them as integers. The letter *i* stands for integer.

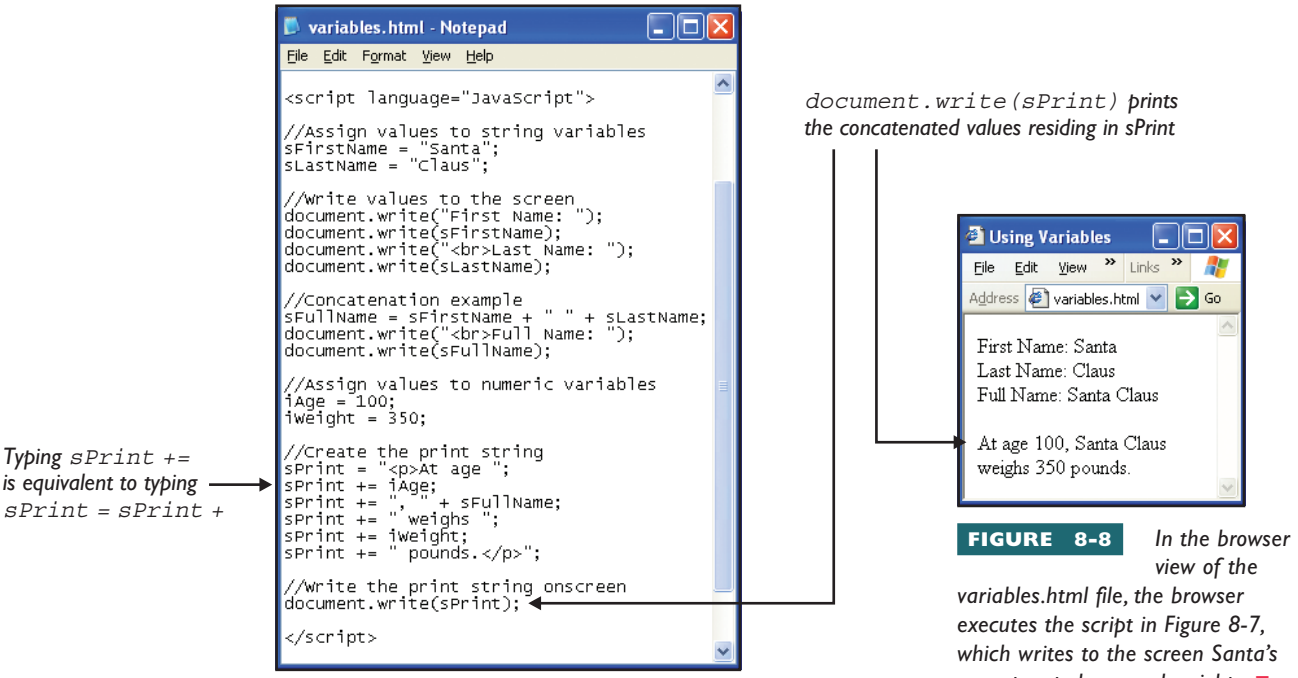


FIGURE 8-7 *In the Notepad view of the variables.html file, the script assigns values to iAge and iWeight variables and concatenates them into a string named sPrint, which the document.write(sPrint) command prints onscreen.* ■

FIGURE 8-8 *In the browser view of the variables.html file, the browser executes the script in Figure 8-7, which writes to the screen Santa's concatenated age and weight.* ■

The string variable *sPrint* holds the string that will be printed onscreen. The print string gets created through a series of statements that keep adding more onto it through the process of concatenation.

Notice the use of the += operator in the concatenation statements. Typing `sPrint +=` is equivalent to typing `sPrint = sPrint +`. Thus, += is a typing shortcut that can save you a lot of time.

Displaying Dynamic Messages Onscreen

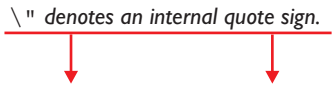
Scripts can include tags that mark up the strings you display onscreen. For example, suppose you wanted the person's age and weight to appear bold. The `<strong>` and `</strong>` tags accomplish these boldings:

L 8-4
```
sPrint = "At age <strong>" + iAge + "</strong>, " + sFullName
+ " weighs <strong>" + iWeight + "</strong> pounds.";
```

Sooner or later, everyone who works with strings gets confronted with the challenge of including quotation marks inside a string. Suppose you need to create a string called *sQuotation*, for example, that contains the following characters:

L 8-5
```
Santa sang "Jingle Bells" as he drove the sleigh.
```

The dilemma is: How do you create a string like this when the quote sign delimits the string? To make it possible to do this, JavaScript uses the special symbol \" to denote a quote sign inside a string. Thus, the way to type this assignment statement is as follows:




```
L 8-6    sQuotation = "Santa sang \"Jingle Bells\" as he drove the sleigh.";
```

Objects, Methods, and Properties

Think about what you see onscreen when you visit a Web page. There are menus, buttons, fields, paragraphs, tables, images, address fields, status bars, and links. Imagine being able to grab hold of these elements and manipulate them dynamically when a user visits the page. Think of how interactive such a Web page would be. If a user mouses over a link, for example, you could pop out a description of what will happen if the user clicks there. On a form that prompts the users to type their e-mail address, you could validate that address and make sure it has the proper format. In a product catalog where users can choose different colors, you could display a color palette and change the product's color dynamically as the user mouses over the color choices. On a Web page that contains a time-consuming process, you could write a message into the status bar informing the user of the progress of the operation.

Would you like to be able to do some of these things? Of course you would. Is doing so difficult? Not if you understand the two underlying principles that this tutorial is about to teach you. First, you need to know where to go to find out what the objects are that you can manipulate dynamically. Second, you need to know when and how you can grab hold of these objects and manipulate them dynamically.

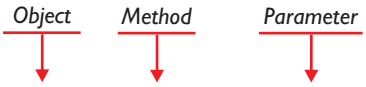
What Is an Object?

In computing, an **object** is a self-contained entity consisting of properties and methods enabling you to do something programmatically. You already have experience using one of the objects that is built into the scripting languages. That is the document object you used to print to the screen in the JavaScript examples you completed earlier in this chapter. You were probably impressed by how easy it was to use the document object to write onscreen. Imagine being able to grab hold of different parts of a Web page and manipulate them just as easily. Soon you will know how to do so. Read on.

What Is a Method?

A **method** is a little computer program built into an object to enable the object to do something. In the Hello, World! script you wrote in the previous part of this chapter, for example, you learned how the `write()`

method enables the document object to print a message onscreen. In your script, you called upon this method with a statement in the form of:



L 8-7
```
document.write("Hello, World!")
```

What Is a Property?

A **property** is an attribute of an object that has a value. You can find out the value of the attribute by grabbing hold of the property and taking a look at its contents. Suppose someone has clicked the Submit button on a form that prompts users to type their e-mail address. Before submitting that address to the server, you would like to grab hold of the e-mail field and see what the user typed. You can easily do this by inspecting the text property of the e-mail field. When the user clicks the Submit button, you can run a script that inspects the text property of the e-mail field and checks to see if it contains the required elements. If the e-mail address does not include the @ sign, for example, you can pop out a box explaining the problem and ask the user to fix the error and try again.

Some objects have properties that you can manipulate to change the appearance of things onscreen. Colors, sizes, bolding, underlining, and positioning are just a few of the properties you can manipulate this way.

What Is an Event?

In computer programming, an **event** is an action that provides an opportunity to trigger a script that can use objects to inspect properties and execute methods that make things happen onscreen or keep records behind the scenes. The most commonly used events are (1) the mouseover, which fires when the user mouses over an object onscreen; (2) the click, which fires when the user clicks the mouse; (3) the double-click, which the user triggers by clicking twice quickly; and (4) the page load, which fires when the user visits a Web site and a page first comes onscreen. Whenever one of these events happens, you have the opportunity to run a script and party with the objects, methods, and properties in the DOM.

JavaScript Clock Project

The purpose of the JavaScript clock project is to give you some experience using the methods and properties of one of the objects in the JavaScript DOM. The clock project uses the JavaScript `Date()` object, which contains methods and properties that enable you to find out the current date and time in a wide range of formats. Combined with the numerical and string processing techniques you learned earlier in this chapter, the `Date()` object enables you to provide dynamic content onscreen in the form of a clock that tells the user what time it is. To create the JavaScript clock project, follow these steps:

1. Pull down the Notepad's File menu and choose New to create a new file. Into this new file, type the following code:

```
<html>
<head>
      <title>Clock Project</title>
</head>
<body>
<h1>The Clock Strikes!</h1>
<script language=javascript>
//What is the date and time now?
dateNow = new Date();
sPrint = "The time is ";
sPrint += dateNow.toString();
//display the print string
document.write(sPrint);
</script>
</body>
</html>
```

2. Pull down the Notepad's File menu and choose Save. When the Save dialog appears, save this file under the filename *clock.html*. Then open the file with your browser. Figure 8-9 shows the completed Notepad file, and Figure 8-10 shows how the script makes the browser display the current date and time onscreen. If you do not see the current date and time, compare your code to the listing in Figure 8-9, correct any typographical errors, save the file, and click your browser's Refresh button to view the corrected code.

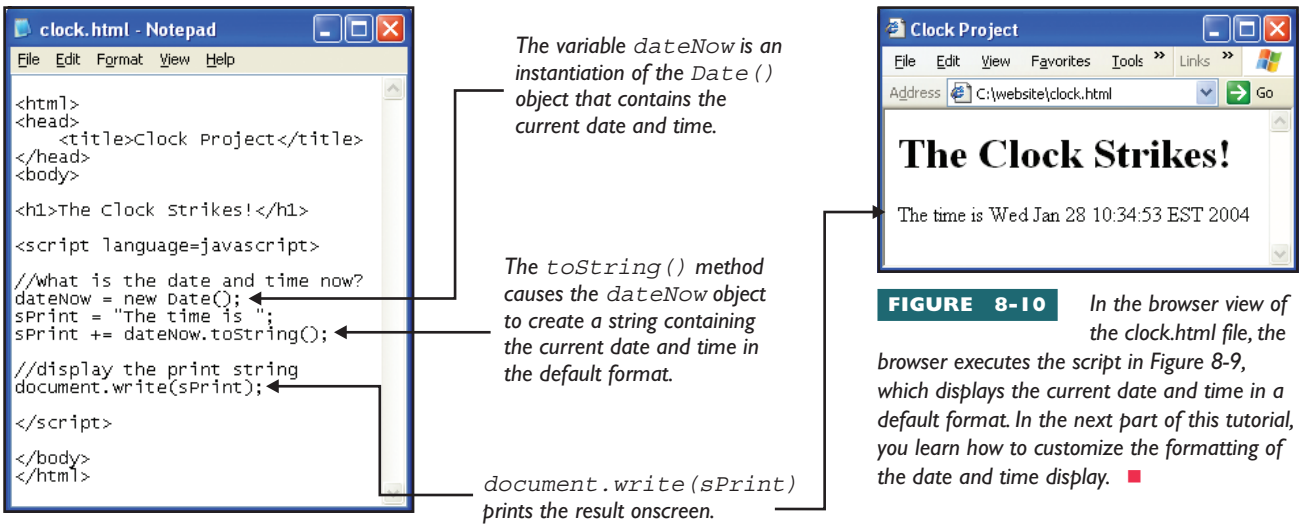
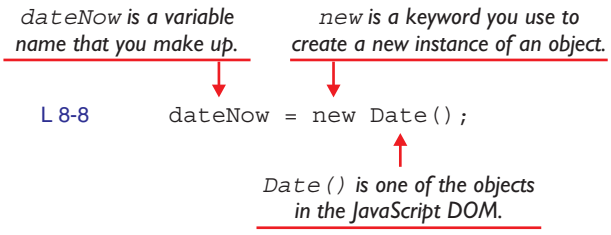


FIGURE 8-9 *In the Notepad view of the clock.html file, the script uses the toString() method of the Date() object to get the current date and time into the sPrint string, which the document.write() command displays onscreen.* ■

FIGURE 8-10 *In the browser view of the clock.html file, the browser executes the script in Figure 8-9, which displays the current date and time in a default format. In the next part of this tutorial, you learn how to customize the formatting of the date and time display.* ■

Clock Code Analysis

The clock script obtains the current date and time by creating a new instance of the `Date()` object. The keyword *new* creates this new instance, which is assigned to the variable named dateNow:



Then the clock script uses the `toString()` method to create the print string that displays the date and time onscreen. The method `toString()` can be used with many objects in the JavaScript DOM. When applied to the `Date()` object, the `toString()` method returns a string containing the date and time, as displayed in Figure 8-10.

Customizing the Date and Time Display

The manner in which the date and time appear onscreen in the clock project is just one of many ways you can format dates and times. Table 8-1 shows the `Date()` object methods and properties that let you grab hold of the individual date and time components and use them to customize the display of the current date and time. If you would like to try some of these customizations, add the following code to the script in your *clock.html* file. The place to put this code is right before the `</script>` end tag.

Method	What It Does
`toString()`	Returns the date and time in the default format
`toUTCString()`	Returns the date and time with Universal Time Code (UTC) formatting
`toLocaleString()`	Returns the date and time in the locale formatting defined on the user's computer
`toDateString()`	Returns the date only, in the default format
`toTimeString()`	Returns the time only, in the default format
`toLocaleDateString()`	Returns the date only, in the locale format
`toLocaleTimeString()`	Returns the time only, in the locale format

TABLE 8-1 *JavaScript Date() Methods for Customizing Date and Time Display Strings* ■

Then save the file and open it with your browser to see the customized print strings onscreen. Here is the code to type:

L 8-9

```
sPrint = "<br><br>The UTC time is ";
sPrint += dateNow.toUTCString();
document.write(sPrint);
sPrint = "<br><br>The locale time is ";
sPrint += dateNow.toLocaleString();
document.write(sPrint);
sPrint = "<br><br>The date is ";
sPrint += dateNow.toDateString();
document.write(sPrint);
sPrint = "<br><br>The time only is ";
sPrint += dateNow.toTimeString();
document.write(sPrint);
sPrint = "<br><br>The locale date is ";
sPrint += dateNow.toLocaleDateString();
document.write(sPrint);
sPrint = "<br><br>The locale time only is ";
sPrint += dateNow.toLocaleTimeString();
document.write(sPrint);
```

note *The `Date()` object contains many more methods for working with dates and times. In the next part of this chapter, you learn how to list all the methods and properties of the `Date()` object as well as all the other objects that are available to you when scripting.*

Document Object Model (DOM)

The **document object model (DOM)** is the official W3C structural definition of the objects, methods, and properties that comprise documents on the World Wide Web. Like many of the W3C standards, the DOM is evolving and taking on more exciting features. You need to be aware that if you use one of the newer features, it is possible that not all browsers support it. Therefore, you should always test your pages with the targeted browsers your users are likely to have. You can find the latest version of the DOM by going to www.w3.org/DOM, where you'll find a tremendous amount of information. Before wading in too deeply, you should work through this chapter's tutorial, which introduces you to things you can do with the DOM without getting overly technical.

Popular JavaScript DOM Objects

In JavaScript, the most commonly used DOM objects provide access to the HTML elements that comprise the document displayed inside the browser window. Of particular importance are the elements in the forms through which the user interacts by selecting things, entering textual information, and clicking to submit the form. Table 8-2 contains an outline of these methods and properties. Remember that this is just a small sampling of the vast array of objects defined in the DOM.

TABLE 8-2 *Popular JavaScript DOM Objects* ■

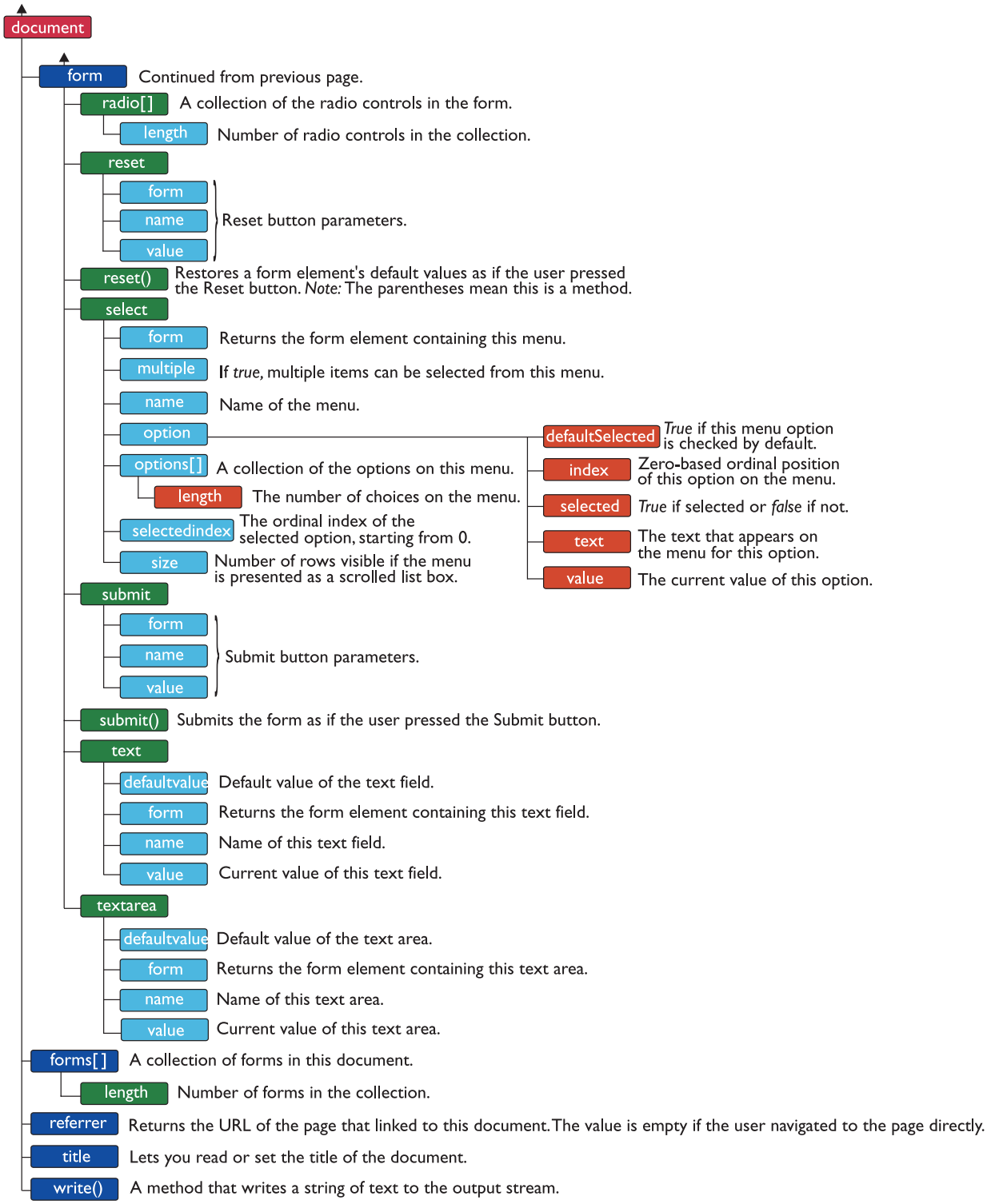


TABLE 8-2 *Popular JavaScript DOM Objects* (continued)

Intrinsic Events

In addition to defining the objects that enable a script to manipulate elements on a Web page, the W3C has defined the intrinsic events that can trigger such a script. Table 8-3 lists and defines these intrinsic events. These definitions are based on the W3C specification at http://www.w3.org/TR/html401/interact/scripts.html#h-18.2.3. In the Dynamic HTML section of this chapter, you learn how to use these events to trigger scripts that access objects in the DOM to bring your Web pages to life and make the user interface more intelligent.

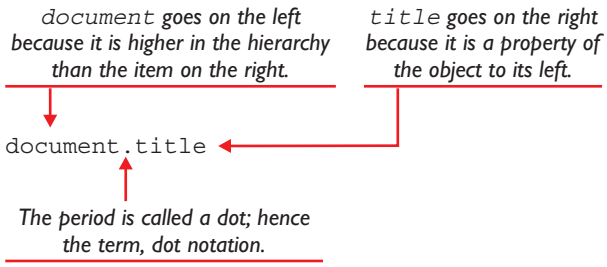
Accessing DOM Objects via Dot Notation

To access DOM objects in a script, you use dot notation to refer to the objects you want to manipulate. Following the document's hierarchical structure, dot notation places to the left of a period elements that are structurally higher than elements further down the tree. Let us work through a very simple example. Suppose you want to write a script that can alter dynamically the title of a Web page. As you saw in the list of popular JavaScript DOM objects presented in Table 8-2, the *document* object has a

Event	When It Occurs	May Be Used With
onclick	Occurs when the user clicks the pointing device button over an element	Most elements
ondblclick	Occurs when the user double-clicks the pointing device button over an element	Most elements
onmousedown	Occurs when the user presses the pointing device button over an element	Most elements
onmouseup	Occurs when the user releases the pointing device button over an element	Most elements
onmouseover	Occurs when the user moves the pointing device button onto an element	Most elements
onmousemove	Occurs when the user moves the pointing device button while it is over an element	Most elements
onmouseout	Occurs when the user moves the pointing device away from an element	Most elements
onfocus	Occurs when an element receives focus either via the pointing device or tabbing navigation	A, AREA, LABEL, INPUT, SELECT, TEXTAREA, and BUTTON
onblur	Occurs when an element loses focus either via the pointing device or tabbing navigation	A, AREA, LABEL, INPUT, SELECT, TEXTAREA, and BUTTON
onkeypress	Occurs when the user presses and releases a key over an element	Most elements
onkeydown	Occurs when the user presses down a key over an element	Most elements
onkeyup	Occurs when the user releases a key over an element	Most elements
onsubmit	Occurs upon the submission of a form	The FORM element
onreset	Occurs when a form is reset	The FORM element
onselect	Occurs when a user selects some text in a text field	INPUT and TEXTAREA
onchange	Occurs when a control loses the input focus and its value has been modified since gaining focus	INPUT, SELECT, and TEXTAREA
onload	Occurs when the browser finishes loading the window or all frames in a frameset	BODY and FRAMESET
onunload	Occurs when the browser removes a document from a window or frame	BODY and FRAMESET

TABLE 8-3 *Intrinsic Events that Can Trigger a Script* ■

title property that can be used to retrieve or set the title of the page. A script can grab hold of the title via the dot notation:



Try This!

Title Bar Clock Script

To experience how easy it is to manipulate an object defined by the DOM, you can write a little script that uses the dot notation *document.title* to set the title of a Web page. In this example, you make the title display the current time. Any time the user wants to know what time it is, the title bar will display it. Follow these steps:

1. Pull down the Notepad's File menu and choose New to start a new page. Type the following code to get the page started:

```
<html>
<head>
</head>
<body>
</body>
</html>
```

2. Pull down the Notepad's File menu and choose Save As; use the Save controls to save this file in your *website* folder under the filename *TitlebarClock.html*.
3. Click to position your cursor before the `</head>` tag that ends the head of the document. Type the following script into the head of the document, or download it from the book's Web site, where the script is called *titlebarclock.txt*:

```
<script language="JavaScript">
function clock()
{
      var dateNow = new Date();
      var hour = dateNow.getHours();
      var minute = dateNow.getMinutes();
      var second = dateNow.getSeconds();
      if (hour > 12)
            hour -= 12;
      if (minute < 10)
            minute = "0" + minute;
      if (second < 10)
            second = "0" + second;
      document.title = "The time is " + hour + ":" + minute + ":" + second;
      setTimeout("clock()", 1000);
}
</script>
```

Try This! *continued*

4. Scroll down to the <body> tag and modify the <body> tag to make it read as follows:

onLoad is an event that triggers when the browser loads the page.

```
<body onLoad="clock()">
```

clock() is the name of the function you typed in the previous step.

5. Save the page and open it with a browser. Notice that the browser's title bar displays the current time. See how the time updates every second. That update is triggered by the last command in the script, which has the format:

setTimeout is a JavaScript command that sets a timer.

This is the number of milliseconds after which the timer will trigger. (1000 milliseconds = 1 second)

```
setTimeout("clock()", 1000)
```

This is the name of the script that will run when the timer triggers.

Accessing DOM Objects by Arrays

An *array* is a named table of memory locations in which values can be stored. Like variables, arrays have names. You can define an array to have a fixed number of memory locations via the following JavaScript command, in which the number specifies the size of the array:

L 8-11
```
dayArray = new Array(6);
```

You use assignment statements to put values into the array, such as:

L 8-12
```
dayArray[0]="Sunday";
dayArray[1]="Monday";
dayArray[2]="Tuesday";
dayArray[3]="Wednesday";
dayArray[4]="Thursday";
dayArray[5]="Friday";
dayArray[6]="Saturday";
```

To access the value in one of the array's slots, you use a subscript. Arrays are zero-based, meaning that the numbering of the slots begins at zero. Thus, to access the value of the first slot in an array called *dayarray*, for example, you would refer to the value as dayArray[0]. To access the next slot, you would refer to the value as dayArray[1].

When a Web page loads, the browser creates arrays for the images, forms, links, anchors, and all the other elements onscreen. As the browser encounters objects on the page, it places them into these arrays. The arrays are indexed sequentially, beginning with zero. The first image on the page, therefore, goes into slot 0 of the images array. You could refer to it in a script as:

L 8-13
```
document.images[0]
```

In like manner, the fourth image on the page could be accessed as:

L 8-14
```
document.images[3]
```

The first form on the page goes into slot 0 of the forms array. You could refer to it in a script as:

L 8-15
```
document.forms[0]
```

Each entry in the forms array has another array inside it called elements, which contains the fields and buttons that comprise the form. If the first form on the screen begins with three radio buttons, for example, the third such button would be referred to as:

L 8-16
```
document.forms[0].elements[2]
```

Referring to the objects in this manner is not very intuitive, however. After you write code like this, if you later insert another radio button ahead of a button you already had onscreen, the insertion changes the indexing. What was formerly `document.forms[0].elements[2]` is now `document.forms[0].elements[3]`, so all the code references to `document.forms[0].elements[2]` would need to be changed. Clearly, you would not want to code this way. You need to know about the arrays to understand how the browser stores the elements on a Web page, but you will not normally be referring to these elements via array indexes in your code.

Accessing DOM Objects by Name

Happily, there is a more direct way for you to access elements on a Web page: you simply name the elements when you put them on the page. At runtime, your scripts can refer to the elements by name. To name a Web page element, you use the *name* and *id* attributes to give the element a unique identifier. The attribute `id` stands for identifier. The reason you use both the `name` and `id` attributes is that older versions of HTML use the `name` attribute, but the latest version uses the `id` attribute. By using both `name` and `id` attributes, you can write code that is compatible with both old and new versions of HTML. Suppose you want to refer to the image on your Web page résumé by the name MyPhoto. To make such a name, you would code the image tag as follows:

L 8-17
```
<img src="photo.gif" name="MyPhoto" id="MyPhoto">
```

So named, the image can appear in a script that modifies the image attributes by referring to the image by name. Following this paragraph is an example that provides users with three buttons onscreen, followed by an image named MyPhoto. The buttons give users the choice of making the image larger or smaller or reloading the page. If the user clicks to make the image larger, the script multiplies MyPhoto.width and MyPhoto.height by 2, thereby doubling the size of the image. If the user clicks to make the image smaller, the script divides these attributes by 2, thereby reducing the image to half its former size. If the user clicks the Reset button, the script calls the `window.location.reload()` function to reload the page, thereby resetting the image to its original size. Here is the code that accomplishes this:

`MyPhoto.height *= 2` is a shorthand way of writing `MyPhoto.height = MyPhoto.height * 2`.

```
L 8-18    <p>
          Click the buttons to change the size of the picture.
          <input type="submit" value="Bigger" onclick="MyPhoto.width *= 2; MyPhoto.height *= 2">
          <input type="submit" value="Smaller" onclick="MyPhoto.width /= 2; MyPhoto.height /= 2">
          <input type="reset" onclick="window.location.reload()">
          </p>
          <img src="photo.gif" name="MyPhoto" id="MyPhoto">
```

`MyPhoto.height /= 2` is a shorthand way of writing `MyPhoto.height = MyPhoto.height / 2`.

Debugging JavaScript via the Alert Box

Writing scripts requires patience because you can run into some problems. When a script is not working, you need to take it from the top and work through it until you find what is causing the problem. Beginners can stumble by forgetting to click the browser's Reload button to refresh the page after modifying the script. Remember that the browser caches the file when the page loads. When you modify the script to fix a problem or add a feature, you must either click the browser's Reload button or pull down the View menu and click Refresh. Otherwise, the browser displays the previously cached version of the file, and you can become frustrated because it appears as though your fix is not working.

Sometimes the browser tells you the number of the offending line of code that is causing the problem. When this happens, pull down the Notepad's Edit menu and choose Go to. Programming glitches can happen in the line or two above the one the browser reports, so look there as well as in the line flagged by the browser.

The greatest aid to finding a problem in a script is to inspect the values of your variables while the script executes. To take a sneak peek at a variable, you can insert an alert box at the point in the script at which you want to inspect the value of the variable. An **alert box** is a window that a script creates by executing the `alert()` method of the JavaScript window object.

Try This!

Rollover Effects

A **rollover** is a special graphical effect you create by using the JavaScript onmouseover and onmouseout event handlers. When the user mouses over something, the onmouseover event fires, causing your script to do something onscreen. Similarly, when the user mouses out of something, the onmouseout event fires, providing another opportunity to run a script. In this exercise, you create the most common kind of rollover, in which a script changes the source of the image when the user mouses over it. To create this rollover, follow these steps:

1. Pull down the Notepad's File menu and choose New to create a new file. Into this new file, type the following code. When you type this, replace *photo.gif* and *photo2.gif* by the actual names of the images you want to see during the rollover:

```
<html>
<head>
      <title>Rollover Effects</title>
</head>
<body>
<p>
Move your mouse on and off the picture.
</p>
<img src="photo.gif" name="MyPhoto" id="MyPhoto"
onmouseover="MyPhoto.src='photo2.gif'"
onmouseout="MyPhoto.src='photo.gif'">
</body>
</html>
```

Replace `photo.gif` and `photo2.gif` with the actual filenames of your images.

2. Pull down the Notepad's File menu and choose Save. When the Save dialog appears, save this file under the filename *rollover.html*. Then open the file with your browser. Move your mouse on and off the image. If the rollover works, congratulate yourself because you have just mastered one of the most popular special effects on the Web. If there are problems, go back to the previous step and troubleshoot the difficulties.
3. If you have the Internet Explorer Web browser, you can make a very nice rollover effect with a single image. Click to position your cursor in the notepad prior to the `</body>` tag that ends the document, and type the following code, replacing *photo.gif* by the name of your actual image:

```
<img src="photo.gif" name="PalePhoto" id="PalePhoto"
style="filter:progid:DXImageTransform.Microsoft.Alpha(opacity=30)"
onmouseover="PalePhoto.style.filter='progid:DXImageTransform.Microsoft.Alpha(opacity=FF)'"
onmouseout="PalePhoto.style.filter='progid:DXImageTransform.Microsoft.Alpha(opacity=30)'">
```

4. Save the file and view it with your browser. At first, the image appears pale onscreen. Mouse over the pale image, and its full color appears. The image filter that creates this effect is a Dynamic HTML technique described later in the Dynamic HTML part of this chapter.

Inside the parentheses of the `alert()` method, you insert the string of characters and variables you want displayed in the alert box. At runtime, when the script encounters the alert, the script pauses while you study the contents of the alert box onscreen. When you click to dismiss the alert box, the script continues executing. By putting alert boxes in strategic places

down through the path of execution in your code, you can step through the code and diagnose the point at which something is going wrong. To create an alert box, follow these steps:

1. Use the Notepad to open the page you want to debug. In this example, open the *clock.html* page you created earlier in this chapter.
2. Scroll to the point in the code at which you want to insert an alert box for debugging. In this example, imagine that you want to find out the value of the sPrint string. Position your cursor at the spot where you want the alert box and type the following code:

```
sDebug = "The value of sPrint is " + sPrint;
alert(sDebug);
```

3. Save the page and open it with your browser. See that the alert box comes onscreen. Reflect on what a simple yet powerful method this is for finding out what is going on behind the scenes when you are trying to solve a problem in your script.
4. After you finish debugging, you can either delete the debugging code you added to display the alert box, or you can comment it out without deleting it. Commenting out code that you think you might want to reactivate will save you the time needed to re-create the debugging code if you encounter another problem later on. To comment out this code, you type the comment character // in front of each line of debugging code, as follows:

The symbol // creates a comment that will not execute at runtime.

```
//sDebug = "The value of sPrint is " + sPrint;
//alert(sDebug);
```

5. To inspect the value of a variable at different points in a script, periodically insert into your code an alert box constructed in this manner:

Type the name of the variable here.

```
alert("The current value of the variable sVarName is: " + sVarName);
```

JavaScript Code Sources

Several Web sites provide free source code for creating a wide variety of JavaScript special effects. When you visit these sites, you will find thousands of scripts organized according to subject and topic. There is also a search feature you can use to locate scripts via keyword. Table 8-4 lists the JavaScript source code sites and tells what you will find there. Also remember to visit this book's Web site for links to new resources and updates that may have been made to the links in Table 8-4. Be sure to follow the link, for example, to the JavaScript clock at Spondoro.com. It is an amazing 3-D animated clock that will follow your mouse across the screen.

Site Name and Web Address	What You Will Find There
JavaScript Developer Central http://developer.netscape.com/tech/javascript	Netscape's JavaScript support site. Full of articles, documentation, and sample code. Includes a brief history of the DOM.
The JavaScript Source http://javascript.internet.com	Hundreds of cut-and-paste scripts to put on your Web page. See especially the generators that can create scripts according to your specifications.
Simply the Best Scripts http://www.simplythebest.net/info/dhtml_scripts.html	A collection of very good scripts. See especially the guitar chord machine at http://www.simplythebest.net/info/javascript38.html#.
Dynamic Drive http://www.dynamicdrive.com	Repository of DHTML scripts that use the latest JavaScript technology, with emphasis on practicality and backward compatibility.
JavaScript Kit http://www.javascriptkit.com	JavaScript tutorials, free JavaScripts, DHTML/CSS tutorials, Web building tutorials.
Builder.Com http://builder.com	To find the Builder.Com repository of JavaScripts, go to http://builder.com and search for JavaScript.
WebReference http://www.webreference.com/programming/javascript	JavaScript articles, tutorials, repositories, specifications, and documentation.
Webmonkey http://hotwired.lycos.com/webmonkey/reference/javascript_code_library/	The Webmonkey JavaScript code library, including free code samples and language extensions.

TABLE 8-4 *JavaScript Source Code Sites*

Maintaining State in Cookies

Do you remember the PacMan game that was popular in video arcades in the 1980s? The game contained magic cookies that, when devoured, made you more powerful. On the Internet, cookies work a little differently but are much more powerful.

What Is a Cookie?

A **cookie** is a place in the computer's memory where browsers can store information about the user. If someone buys an audio CD by Britney Spears, for example, the site might create a cookie indicating that the user likes pop music. The next time the user visits the site, it might display ads for similar pop music titles. Or the cookie might keep track of screens visited and use that information to resume where the user left off.

There are two kinds of cookies, namely, *persistent cookies* that are stored in small text files on the user's hard disk, and *per-session cookies* that are stored in RAM. Because persistent cookies are stored on disk, they survive from session to session, even after the user closes the Web browser and turns off the computer. Per-session cookies, on the other hand, evaporate when the user closes the Web browser, which frees the RAM.

Why Does the Internet Need Cookies?

Without cookies, the Internet would be in deep trouble. Each time a user interacts with a Web site, the Internet closes the socket through which that interaction took place. In other words, the Internet hangs up on you after it finishes serving you the page. While you read what is onscreen, the Internet devotes its resources to serving other users. It does not pay you

any attention until you click or select something that requires interaction with a server. Each time you interact, a socket opens and then closes again as soon as the Internet finishes serving you the page. The Internet does not keep track of which users get which sockets. For this reason, the Internet is said to be stateless. The Internet needs cookies so it can maintain state from screen to screen. When you interact with a secure Web site, it creates one or more session cookies that the browser stores temporarily in your computer's RAM. The next time you interact with that site, it inspects the cookies to find out who you are. Thus, the Internet uses cookies to maintain state from page to page.

The per-session cookies are very secure. Only the server that set the cookies is able to read them, and the cookies evaporate at the end of the session. Persistent cookies, on the other hand, reside on the user's hard disk drive. Knowledgeable users can find the cookie files and read their contents with any text editor. If these contents are unencrypted, the cookies can be read in plain text. It is appropriate to use persistent cookies to store information that would not cause a security problem if sniffed. The advantage of using persistent cookies is that a server can keep data on the user's PC, thereby avoiding the need to store that data in a server-side database. Cookies thereby provide a mechanism you can use to store nonsensitive data on every computer on the Internet from which someone visits your site—unless the user turns off the browser's cookie feature. The vast majority of users have their cookies turned on because virtually all sites that have you log on and off, including all of the Internet's e-commerce sites, use cookies to maintain state.

How to Read and Write the Value of a Cookie

Table 8-2 (earlier in this chapter) identifies the commonly used JavaScript DOM objects. One of the DOM objects is the cookie collection, which you access via the document.cookie property. As you might expect, setting the value of a cookie requires a little scripting. The following example uses a cookie to keep track of how many times the user clicks a button onscreen. Although the task is very simple, the cookie functions can be used to read or set the value of any cookie. To create the cookie cutter, follow these steps:

1. Pull down the Notepad's File menu and choose New to start a new file. Type the following code or download it from this book's Web site, where the filename is *cookiecutter.txt*:

```
<html>
<head>
<title>Cookie Cutting</title>
<script language="JavaScript">
function setCookie(sName,sValue,iMinutes)
{
    if (iMinutes == 0)
        document.cookie = sName + "=" + sValue;
    else
    {
        dateExpires = new Date();
```

```
        dateExpires.setMinutes(dateExpires.getMinutes()+iMinutes);
        document.cookie = sName + "=" + sValue
                          + ";expires="
                          + dateExpires.toGMTString();
    }
}

function readCookie(sName)
{
    cookieChecker = document.cookie.split("; ");
    for (i=0; i<cookieChecker.length; i++)
    {
        if (sName == cookieChecker[i].split("=")[0])
            return cookieChecker[i].split("=")[1];
    }
    return ""; //returns nothing if cookie not found
}

</script>
</head>
<body>
<h1>Hit Counter</h1>

<script language="JavaScript">
iClickCounter = readCookie("ClickCounter");
</script>

<form method="get" action="cookies.html">
<input type="submit" value="Click Me!"
onClick="setCookie('ClickCounter',++iClickCounter,0);">
</form>

<p>Number of clicks:
<script language="JavaScript">
document.write(iClickCounter);
</script>
</p>
</body>
</html>
```

2. Save the file in your *website* folder under the filename *cookies.html*. To see what it does, open the *cookies.html* file with your browser. If you get any error messages, the browser tells you the number of the line that is causing the problem. Troubleshoot the problem by proofreading your code more carefully until you get the page to open in the browser without reporting any errors.
3. Click the button and observe how the click counter increases. Each time you click the button, the script is adding 1 to the value of the click counter stored in the cookie named ClickCounter.
4. As printed in this exercise, the cookie is set to expire at the end of the session. Close all of your browser windows; then use your browser to reopen the *cookies.html* file. Notice that the click

counter starts over when you start clicking the button. That is because the cookie expired when you closed the browser windows.

5. To make the cookie persist, increase the value of the following variable, which tells the browser how many minutes to make the cookie last. In this example, set the value to 1, to make the cookie last one minute:

```
<form method="post" action="cookies.html">
<input type="submit" value="Click Me!"
onClick="setCookie('ClickCounter',++iClickCounter,0);">
</form>
```

Replace this 0 with the number of minutes you want the cookie to last.

6. Close all your browser windows. Then open the *cookies.html* page with your browser. Click the button a few times. Make a note of the value of the click counter. Then close all your browser windows. Immediately use your browser to reopen the *cookies.html* page. Notice that the value persists because you started the new session before the cookie expired.

7. Close all your browser windows again. This time, wait for a couple of minutes before you reopen the *cookies.html* file. This causes the one-minute cookie to expire. Open the page and notice that the click counter restarts because the cookie expired. In practice, you normally set a persistent cookie to last for several days or months, depending on the purpose. In this example, you set it to last just a minute so you can observe what happens when it expires, without having to wait so long.

Cookie Cutter Code Analysis

Figure 8-11 contains the completed code of the *cookies.html* page, and Figure 8-12 shows it running onscreen. The script stores the click counter in a cookie called ClickCounter. The page reads the value of this cookie into a variable named iClickCounter via the following script:

`ClickCounter` is the name of the cookie that keeps track of how many times the user clicks the button.

L 8-19

`iClickCounter` is a variable that will be used again later on this page.

```
<script language="JavaScript">
iClickCounter = readCookie("ClickCounter");
</script>
```

`readCookie()` is one of the functions defined in the `<head>` of the document.

Each time the user clicks the "Click Me!" button, the *onclick* event fires. The button's `<input>` tag takes this opportunity to increase the value of the variable iClickCounter and store the new value in the cookie called ClickCounter. If this code seems a little complicated, worry not, because

you do not need to know this to pass the CIW exam. By studying the following callouts, however, you can get an idea of how this code works:

The onClick event fires when the user clicks the button.

ClickCounter is the name of the cookie you are setting.

Setting the number of minutes to 0 makes this a per-session cookie that will not persist on the user's hard drive.

L 8-20 `onClick="setCookie('ClickCounter',++iClickCounter,0);"`

`setCookie()` is a function defined in the `<head>` of the document.

The ++ prefix causes the value of iClickCounter to increase by one each time it is passed to the `setCookie()` function as the value of the cookie.

Many Web developers criticize JavaScript for not having the `readCookie()` and `setCookie()` functions built in. Now that you have these functions working, you can use them on other pages any time you need to read or set the value of a cookie.

cookies.html - Notepad

File Edit Format View Help

```
<html>
<head>
<title>Cookie Cutting</title>
<script language="JavaScript">

function setCookie(sName,sValue,iMinutes)
{
    if (iMinutes == 0)
        document.cookie = sName + "=" + sValue;
    else
    {
        dateExpires = new Date();
        dateExpires.setMinutes(dateExpires.getMinutes()+iMinutes);
        document.cookie = sName + "=" + sValue
                          + ";expires="
                          + dateExpires.toGMTString();
    }
}

function readCookie(sName)
{
    cookieChecker = document.cookie.split("; ");
    for (i=0; i<cookieChecker.length; i++)
    {
        if (sName == cookieChecker[i].split("=")[0])
            return cookieChecker[i].split("=")[1];
    }
    return ""; //returns nothing if cookie not found
}

</script>
</head>

<body>
<h1>Hit Counter</h1>

<script language="JavaScript">
iClickCounter = readCookie("ClickCounter");
</script>

<form method="post" action="cookies.html">
<input type="submit" value="Click Me!"
onClick="setCookie('ClickCounter',++iClickCounter,0);">
</form>

<p>Number of clicks:
<script language="JavaScript">
document.write(iClickCounter);
</script>
</p>
</body>
</html>
```

FIGURE 8-11 *The head section of the cookies.html page contains JavaScript functions that you can use whenever you need to set or read the value of a cookie.* ■

FIGURE 8-12 *Powered by the code in Figure 8-11, the browser uses a cookie to keep track of how many times the user clicks the button.* ■

Try This!

Inspecting Your Computer's Cookies

If you have never looked around your computer to see what cookies are stored there, you will be amazed by what this exercise will turn up. Remember that there are two kinds of cookies: (1) per-session cookies that are stored in RAM and evaporate when the user closes the browser windows and (2) persistent cookies that survive from session to session. Persistent cookies endure even when the user reboots or powers off the computer. The persistent cookies stick around because the browser stores them on the computer's hard drive. To inspect the cookies the browser is storing on your hard drive, follow these steps:

1. Click the Windows Start button and choose Search to make the Search window appear.
2. Click the option to search all files or folders. When the search criteria form appears, type the word *cookie* into the field titled *All or part of the file name*. Leave the other fields blank.
3. Click to reveal the more advanced options and set the options to search for hidden files and system files.
4. Click the Search button and wait while your computer looks for filenames containing the word *cookie*. One by one, your cookie files will begin to appear in the search window.
5. Look for a folder called *cookies*. Right-click it and choose the option to enter that folder. Here you will probably find many more cookie files.
6. To inspect a cookie file, right-click its filename to bring up the quick menu, choose Open, and choose the option to open the file with the Notepad. Many cookies are encrypted, so do not be disappointed if you cannot decipher the contents of these files.
7. Reflect on how servers all over the Internet, including commercial Web sites, are using your computer's hard drive as a storage medium.

Cascading Style Sheets

A **cascading style sheet (CSS)** is a set of rules that define styles to be applied to entire Web pages or individual Web page elements. Each rule consists of a selector followed by a set of curly braces containing the style properties and their values. The selector can be an HTML element, a user-defined style known as a class, or the ID of a specific element on a page. Here are some examples of style definitions that you might find on a CSS:

L 8-21

```
a:link{color: rgb(255,204,0) }
a:visited{color: rgb(153,204,204) }
a:active {color: rgb(102,255,0) }
body
{
      font-family: Garamond, Times New Roman, Times;
      background-color: rgb(51,102,204);
      color: rgb(255,255,153);
}
table
{
      table-border-color-light: rgb(153,255,204);
      table-border-color-dark: rgb(0,0,51);
}
```

```
h1, h2, h3, h4, h5, h6{font-family: Verdana, Arial, Helvetica}
h1 {color: rgb(255,204,0) }
h2 {color: rgb(153,255,51) }
h3 {color: rgb(0,255,204) }
h4 {color: rgb(255,204,0) }
h5 {color: rgb(153,255,51) }
h6 {color: rgb(0,255,204) }
.callout { font-size: small }
#trailer { font-family: serif }
```

User-defined style classes begin with a period.

ID selectors begin with a pound sign.

Three Kinds of Cascading Style Sheets

There are three ways of applying cascading style sheets to a Web page: external, embedded, and inline. An **external CSS** keeps all the style definitions in a separate CSS file that you include in a Web page at runtime by using the `<link>` tag to apply the style sheet to the page. An **embedded CSS** is a style sheet that gets copied physically into the head of the Web page and applies to the Web page as a whole. An **inline CSS** is a style sheet that applies to only one page element so it gets copied "inline" on the page with that element. The following exercises help you understand these definitions by walking you through some examples of the three ways of applying styles to a Web page.

Creating an Inline CSS

Sometimes, no matter how hard you try, you just aren't satisfied with the look of something on your page. You want part of the text to stand out more, for example, or you want to soften the appearance of something. Enter the inline CSS, which was invented to provide you with a way to change a single element on a page without affecting any others. Suppose you want to make the name and e-mail fields stand out a little more on the *subscribe.html* form you created in the previous chapter. You want to modify those two fields only, without changing the appearance of any other input controls onscreen. Follow these steps:

1. Use the Notepad to open the *subscribe.html* page you created in the previous chapter. You will find this file in your *website* folder.
2. Click to position your cursor inside the tag you want to stylize. In this example, click to position your cursor before the > that concludes the input tag that creates the Name field. Modify this tag to read as follows:

```
<input type="text" name="Name" size="50" maxlength="150"
style="border-style: inset; border-width: 4">
```

When an inline CSS has more than one style change, you separate the styles with a semicolon.

3. Save the file and open it with your browser to see how this inline style changes the style and width of the Name field's border.
4. Now that you have stylized the Name field, you should make a similar change to the Email field. Try to do this on your own, but if you need help, here is the modification to make to the Email field's input tag:

```
<input type="text" name="Email" size="50" maxlength="150"
style="border-style: inset; border-width: 4">
```

5. Save the file and click the browser's Refresh button to view the latest version of the page. Figure 8-13 shows that the text fields appear inset into the screen.

Creating an Embedded CSS

The purpose of an embedded CSS is to enable you to make style changes that apply to the Web page as a whole. The embedded CSS goes into the head section of the page, and the style rules defined there apply to the whole page. If any element on the page has a style whose attributes have been affected by the style sheet, those rules will take effect in displaying the element onscreen. The best way to understand this is to work through an example. Suppose you like the color blue and you want to make certain Web page elements appear in blue. To do this with an embedded CSS, follow these steps:

1. Use the Notepad to open the *subscribe.html* page you created in the previous chapter. You will find this file in your *website* folder.
2. Click to position your cursor in the head section of the document. Immediately prior to the `</head>` stop tag, type the following embedded style sheet. Notice that the `<style>` start and `</style>` stop tags demarcate the embedded style:

```
<head>
<title>Newsletter Subscription Form</title>
<style>
      h1 { font-family: Comic Sans MS; color: #0000DD }
</style>
</head>
```

3. Save the file and open it with your browser. The heading should appear blue in the comic font. If you do not see this on your screen, return to the previous step, proofread the code, and troubleshoot any problems.
4. Suppose you also want to make the paragraph text appear blue in the comic font. Use the Notepad to add the following line to the embedded style sheet in the head section of the *subscribe.html* page:

```
<style>
     h1 { color: #0000DD; font-family: Comic Sans MS }
     p { font-family: Comic Sans MS; color: #0000DD }
</style>
```

5. Save the file, view it with your browser, and click the browser's Refresh button. Figure 8-14 illustrates that this embedded style sheet makes the heading and paragraph text appear blue in the comic font.

Creating an External CSS

When you want a style to apply to multiple Web pages, you should create an external CSS and link the Web pages to it. This approach has two advantages. First, it saves you time when you create a new page. Instead of typing the styles into the page, you link the page to the style sheet, thereby saving the time you would otherwise spend keyboarding. Second, and more important, using an external style sheet makes your Web easier to maintain. If you want to make a style change that applies to the entire Web, you simply make that change to the external style sheet, thus saving

The inline style "border-style: inset; border-width: 4" causes the text input fields to appear inset into the screen.

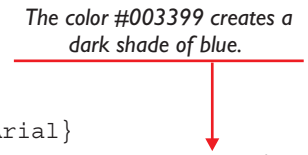
FIGURE 8-14 *An inline style applies only to the particular element(s) in which you put it. In this example, the inline style "border-style: inset; border-width: 4" is in the Name and Email fields, causing them to appear inset into the screen.* ■

h1 {color: #0000DD; font-family: Comic Sans MS} is the style rule that creates this shade of blue in the comic font.

FIGURE 8-13 *An embedded style sheet applies to every instance of a Web page element onscreen. In this example, the blue text and the comic font come from the style settings embedded in the <head> of the document.* ■

the time you would otherwise need to spend changing the style on every page in the Web. To create an external cascading style sheet, follow these steps:

1. Pull down the Notepad's File menu and choose New to create a new file. Into this new file, type the following code. This code contains styling for some of the more popular font and color choices you see out on the Web. After you create this style sheet and gain experience using it, you can modify and add new settings to create your own style:

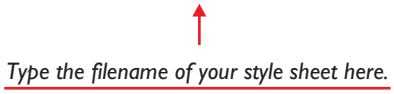



```
BODY  {background-color:"#FFFFFF"}
H1    {color:#003399; font-family:Arial}
A:link      {text-decoration:underline; color:#003399}
A:visited   {text-decoration:underline; color:#003399}
A:hover     {text-decoration:underline; color:red}
```

2. Pull down the Notepad's File menu and choose Save As. When the Save dialog appears, pull down the Save as Type menu and set it to All Files. Then use the controls to save this file in your *website* folder under the filename *MyStyles.css*.

3. You use the `<link>` tag to link the style sheet to any page in the Web. Suppose you want to link the style sheet, for example, to your Web page résumé. Use the Notepad to open the *resume.html* page and click to position your cursor in the `<head>` of the document. Immediately prior to the `</head>` tag, type the following `<link>` tag:

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



4. Save the file and open it with your browser. Your *resume.html* page now appears with the styling in the cascading style sheet. The a:link styling makes the hyperlinks change color when the user mouses over them. To see this happen onscreen, move your cursor over one of the links in your bulleted table of contents, for example, and observe that the text changes color.

5. If you experiment with changing the style settings in your cascading style sheet, remember to save the file after you make the changes, and then click the browser's Refresh button. If the changes do not appear onscreen, click Ctrl-Refresh, which makes the browser reload everything associated with the page.

When to Use *the and Tags*

Sometimes you may want to stylize part, instead of all, of a Web page element. To provide a way for you to do that, the W3C invented the HTML inline `<span>` start and `</span>` stop tags. Suppose you wanted to

colorize a few words in a paragraph. Instead of applying the color style to the <p> tag, which would colorize the entire paragraph, you can instead create a around the words you want colorized, and apply the color to the tag, as follows:

L 8-22
```
<p>Notice how <span style="color: yellow">yellow words</span> appear onscreen.</p>
```

Other situations may arise in which you want to apply a style to larger divisions of a document at the block level. You create block-level divisions with the start and stop tags, where stands for division. The syntax for the tag is exactly the same as for the tag. Because is a block-level tag, however, the browser begins a new line at the beginning of the division. If you do not want a new line, use the tag, which enables you to stylize elements inline without starting a new block onscreen.

When to Create Style Classes and IDs

In cascading style sheets, a **class** is a named definition of one or more styles. You create the class by prefixing its name with a dot in the CSS file. Suppose you want a class you can use whenever you are displaying warning messages onscreen. In the style sheet, you could create such a class, as follows:

L 8-23
```
<style>
.warning { color: red; font-family: arial; font-weight: bold}
</style>
```

Whenever you want an HTML element to have that style, you use the *class* attribute, as in this example:

L 8-24
```
<p>Be careful when you try this, because <span class="warning">
the burner is hot!</span></p>
```

The "warning" class makes red and bold ***the burner is hot!***

Style sheets also enable you to stylize elements via unique identifiers called IDs. In cascading style sheets, an ID is a unique style identifier intended to apply to one, and only one, Web page element onscreen. In the style sheet, you create an ID via the # character. Suppose there is a trailer that appears once, and only once, onscreen. In the style sheet, you could create a unique styling for the trailer, as follows:

L 8-25
```
<style>
#trailer { color: #808080; font-family: Century Schoolbook }
</style>
```

You can make the trailer have this style by referring to it in your HTML, as follows:

L 8-26

```
<span ID="trailer">
<p>Everything inside this span is the trailer.</p>
<p>It appears onscreen in the style of the ID named "trailer".</p>
</span>
```

What Is Absolute Positioning?

On the cutting edge of cascading style sheets is a feature called **absolute positioning**, which enables you to position page elements precisely onscreen based on x,y coordinates. The upper-left corner of the browser window is position 0,0. To position a page element 40 pixels down and 30 pixels across the screen, for example, the inline style would appear as follows:

L 8-27

```
style="position: absolute; top: 40; left: 30;"
```

warning *You need to be aware that some browsers do not support absolute positioning. It is a hot feature that all browsers should support, but until they do, you may need to postpone creating pages with absolute positioning if any of your users have browsers that do not support it yet.*

Absolute positioning makes it possible to layer objects on top of each other. To provide control over the order in which the objects appear onscreen, you can use an attribute called the **z-index**. You can set the value of z-index to tell the browser the order in which to display objects that overlap. The lower the value, the sooner the layer displays onscreen. In other words, an item with a lower z-index will appear underneath overlapping items with higher z-index values.

Try This!

Pile Rocks with Absolute Positioning and z-index

With absolute positioning, you can be creative in placing graphics onscreen. In this example, you download the images of three rocks and view them individually onscreen. Then you use absolute positioning to create a rock pile in which the images appear in overlapping layers onscreen. Thus, you experience firsthand the concept of absolute positioning and layering. To create the rock pile, follow these steps:

1. The three rocks are located on the book's Web site. Right-click each rock, and use the quick menu to save the rocks in your *website* folder. Do not change the filenames, which are *igneous.gif*, *metamorphic.gif*, and *sedimentary.gif*. If you have ever studied geology, you will recognize igneous, metamorphic, and sedimentary as the three basic kinds of rocks.
2. Pull down the Notepad's File menu and choose New to create a new file. Type the following code, which displays the three rocks onscreen:

```
<html>
<head>
<title>Rocks</title>
</head>
<body>
<img src="igneous.gif" width="320" height="240">
<img src="metamorphic.gif" width="320" height="240">
<img src="sedimentary.gif" width="320" height="240">
</body>
</html>
```

Try This!
continued

3. Pull down the Notepad's File menu and choose Save As. When the Save dialog appears, pull down the Save as Type menu and set it to All Files. Then use the controls to save this file in your *website* folder under the filename *rockpile.html* and open it with your browser. The three rocks appear onscreen, side by side, as illustrated in Figure 8-15.
4. Now comes the fun part. Use the Notepad to add the following tags to the images displayed on the *rockpile.html* page. These tags use inline styles to layer the rocks on top of each other:

```
<body>
<span style="position: absolute; left: 30; top: 150; z-index: 0">
<img src="igneous.gif" width="320" height="240">
</span>
<span style="position: absolute; left: 150; top: 35; z-index: 1">
<img src="metamorphic.gif" width="320" height="240">
</span>
<span style="position: absolute; left: 220; top: 140; z-index: 2">
<img src="sedimentary.gif" width="320" height="240">
</span>
</body>
```

5. Save the file and view it with your browser. Click Refresh to make sure you are viewing the current version of the file. Figure 8-16 shows how the rocks layer onscreen. You have made a rock pile!
6. Now reflect: Inside the tags can go other html elements, such as headings and paragraphs and tables. In fact, any page element can be contained by the span and thereby positioned onscreen via absolute positioning. In the next chapter, you study how you can use CSS to lay out Web pages that may be more accessible than some kinds of table-driven layouts.

This is a dime. Geologists put coins atop rock samples to enable you to gauge the relative sizes of the rocks.

FIGURE 8-15 *Without layering, images appear individually onscreen. Compare this to Figure 8-16, which uses absolute positioning to layer the same images on top of each other.* ■

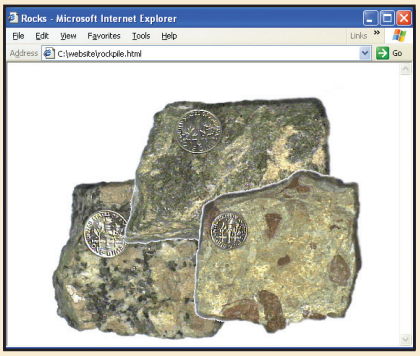


FIGURE 8-16 *A pile of three rocks created via absolute positioning. Compare this to Figure 8-15, in which the same images appear without the inline styles that layer the rocks into a pile.* ■

Dynamic HTML

Dynamic HTML is a term invented by Microsoft to refer to the animated Web pages you can create by using the DOM to combine HTML with style sheets and scripts that bring Web pages to life. Some people get confused by the term Dynamic HTML because they think it refers to some kind of a product. Dynamic HTML is not a product; rather, it is a concept. Whenever you create dynamic effects onscreen by manipulating objects in the DOM, you are doing what Microsoft refers to as Dynamic HTML. You will understand this more by working through the following examples.

Dynamic Animation Effects

In the style sheet section of this chapter, you learned how to use absolute positioning to place an image at any x,y location onscreen. JavaScript has a timer event you can use to manipulate the values of x and y dynamically, thereby creating an animation onscreen. To create such an animation, follow these steps:

1. Pull down the Notepad's File menu and choose New to create a new file. Into this new file, type the following code:

```
<html>
<head>
<title>Dynamic HTML Example</title>
<script language="JavaScript">
var id;
function BeginAnimation()
{
    id = window.setInterval("ContinueAnimation()",50);
}
function ContinueAnimation()
{
    MyPhoto.style.pixelLeft += 10;
    MyPhoto.style.pixelTop += 4;
    if (MyPhoto.style.pixelLeft>200)
    {
        window.clearInterval(id);
    }
}
</script>
</head>
<body onload="BeginAnimation()" marginheight="0" topmargin="0" leftmargin="0">
<img id="MyPhoto" style="position:absolute; top:0; left:0; z-index:-1" src="photo.gif">
</body>
</html>
```

Type the filename of the image you want this code to animate.

2. Pull down the Notepad's File menu and choose Save As. When the Save dialog appears, save this file under the filename *dynamic.html*. Then open the file with your browser. You should see the image move around the screen as the script alters the x,y values that position the image onscreen.

Dynamic HTML Code Analysis

To create an animation with Dynamic HTML, you need to latch onto an event that can get the animation started. The *dynamic.html* script does this in the `<body>` tag via the *onload* event, one of the intrinsic events identified earlier in this chapter in Table 8-3. Notice that the `<body>` start tag is programmed to fire the `BeginAnimation()` function when the page loads:

L 8-28
```
<body onload="BeginAnimation()">
```

The `BeginAnimation()` function is very brief. It calls upon the `setInterval()` method of the JavaScript window object to set a timer that will go off after 50 milliseconds and fire the `ContinueAnimation()` function:

L 8-29
```
function BeginAnimation()
{
    id = window.setInterval("ContinueAnimation()",50);
}
```

Type the number of milliseconds after which you want the timer to fire.

After you get an animation started, you need to keep it going. The `ContinueAnimation()` function does that by computing the image's next position and setting another timer. This process continues until the image moves past the point at which the IF statement stops the animation by calling upon the `clearInterval()` method to stop the timer from going off any more:

L 8-30
```
function ContinueAnimation()
{
    MyPhoto.style.pixelLeft += 10;
    MyPhoto.style.pixelTop += 4;
    if (MyPhoto.style.pixelLeft>200)
    {
        window.clearInterval(id);
    }
}
```

This controls how far over the picture will move.

This controls how far down the picture will move.

This determines when the animation will stop.

The `clearInterval()` method stops the timer from firing.

The variable named `id` contains the identification number of the timer to be stopped.

There is no limit to the number of animation patterns you can create onscreen. This example moved the image along a straight line to keep the coding straightforward. If you know your math, however, there is no limit to the patterns of movement you can create onscreen.

Dynamic Gradient Effects

One of my favorite background effects is the **gradient**, a graphical effect created by colors fading gradually across or down the screen. Figure 8-17 shows examples of some of the gradients you can create with dynamic HTML. The code needed to do this is very straightforward, thanks to the built-in gradient method that creates these effects. To create a gradient background on any Web page, follow these steps:

1. Use the Notepad to open the page on which you want to create the gradient background. In this example, open the *resume.html* file you created in the previous chapter.
2. Modify the `<body>` tag to read as follows:

```
<body style="filter: progid:DXImageTransform.Microsoft.gradient
(startColorstr=#88BBF7F6, endColorstr=#FFFFFFC0)" >
```

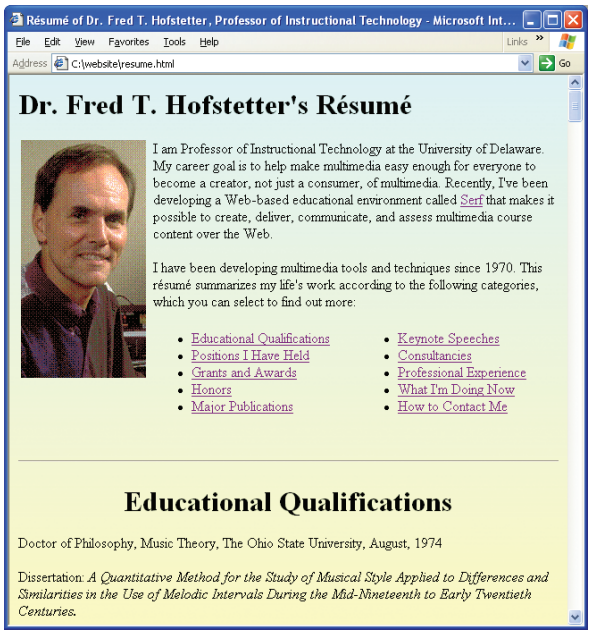
3. Pull down the File menu and choose Save to save the page; then open it with your browser. If you typed everything correctly, you will see the gradient onscreen.
4. To experiment with different colors, you can alter the start and stop color strings, which are named *startColorstr* and *endColorstr*, respectively. Save the file and click the browser's Refresh button to view the revised gradient onscreen. The format of the color string is #AARRGGBB, where AA is the alpha value, RR is red, GG is green, and BB is blue. The alpha value controls opacity, with values ranging from 00 (transparent) to FF (full color).

Dynamic Page Transitions

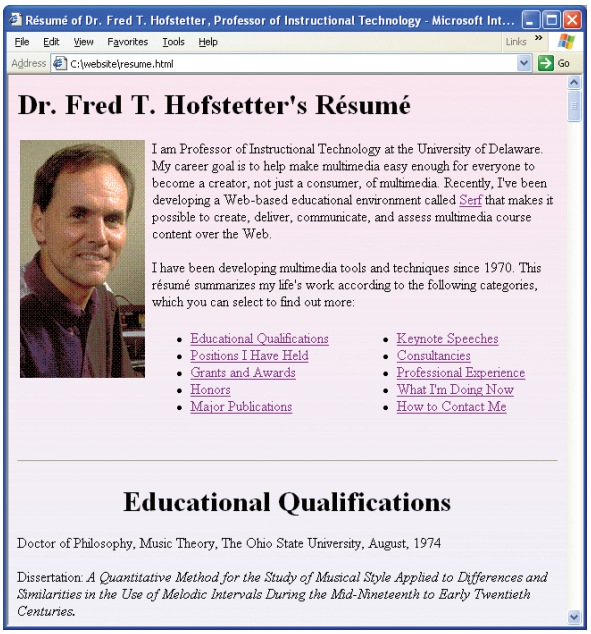
A **page transition** is the style or manner in which the screen changes when the browser brings up a new document and displays it onscreen. You can use a wide variety of page transition effects for pizzazz. As with all special effects, you should not overuse page transitions or feel compelled to display a different effect each time you display a page. If there is an effect you like, however, you certainly may use it in good taste. To create a Dynamic HTML page transition, follow these steps:

1. Use the Notepad to open the page for which you want to create a transition effect. Click to position your cursor in the head of the document, prior to the `</head>` stop tag.
2. To set the transition effect that users will see when the page comes onscreen, type the following code:

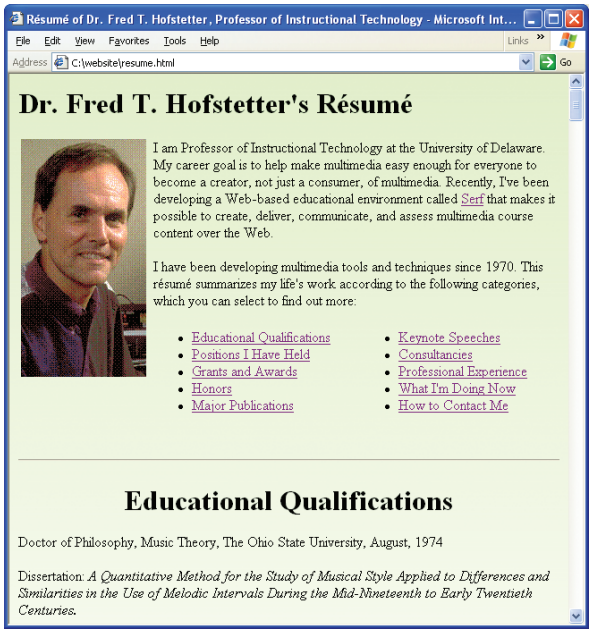
```
<META http-equiv="Page-Enter"
CONTENT="progid:DXImageTransform.Microsoft.Blinds(Duration=2)" />
```



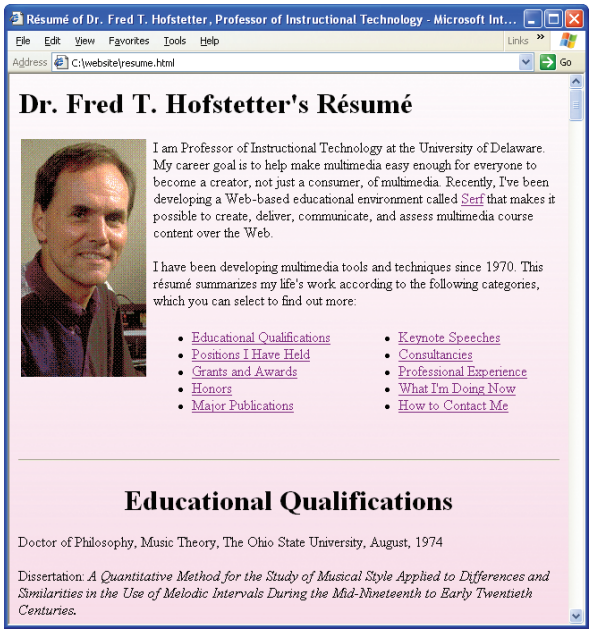
startColorstr=#88BBF7F6, endColorstr=#FFFFFFC0



startColorstr=# F8D2EF, endColorstr=#C2C3FE



startColorstr=#88CCFBA3, endColorstr=#88EDFEDE



startColorstr=#44F8E4ED, endColorstr=#88F4C2DA

FIGURE 8-17 *These gradient backgrounds are some of the Dynamic HTML effects you can create via the gradient method of the DXImageTransform object. You can make millions of different gradients by manipulating the start and stop color strings.* ■

3. To set the transition effect that users will see when the page leaves the screen, type the following code:

```
<META http-equiv="Page-Exit"
CONTENT="progid:DXImageTransform.Microsoft.Slide(Duration=2.500,slidestyle='HIDE')" />
```

note *There are many more transition effects besides the common examples listed in this table. For more, go to the DHTML workshop at msdn.microsoft.com/library/default.asp?url=/workshop/author/dhtml/dhtml.asp.*

4. Save the file and open it with your browser. Notice the effects you specified when the page goes on or off screen. Remember not to overuse these effects. They are cool, but users can grow tired of them, especially if you make the transitions last more than a second or two. Table 8-5 lists some of the other page transition effects. To audition them, go back to steps 2 and 3 and modify the settings according to the examples provided in Table 8-5.

Effect	Style Setting
Blinds Horizontal	DXImageTransform.Microsoft.Blinds(direction='down')
Blinds Vertical	DXImageTransform.Microsoft.Blinds(direction='right')
Box In	DXImageTransform.Microsoft.Iris(irisstyle='SQUARE', motion='in')
Box Out	DXImageTransform.Microsoft.Iris(irisstyle='SQUARE', motion='out')
Checkerboard Across	DXImageTransform.Microsoft.CheckerBoard(direction='right')
Checkerboard Down	DXImageTransform.Microsoft.CheckerBoard(direction='down')
Circle In	DXImageTransform.Microsoft.Iris(irisstyle='CIRCLE', motion='in')
Circle Out	DXImageTransform.Microsoft.Iris(irisstyle='CIRCLE', motion='out')
Random Bars Horizontal	DXImageTransform.Microsoft.RandomBars(orientation='horizontal')
Random Bars Vertical	DXImageTransform.Microsoft.RandomBars(orientation='vertical')
Random Dissolve	DXImageTransform.Microsoft.RandomDissolve
Split Horizontal In	DXImageTransform.Microsoft.Barn(orientation='horizontal', motion='in')
Split Horizontal Out	DXImageTransform.Microsoft.Barn(orientation='horizontal', motion='out')
Split Vertical In	DXImageTransform.Microsoft.Barn(orientation='vertical', motion='in')
Split Vertical Out	DXImageTransform.Microsoft.Barn(orientation='vertical', motion='out')
Strips Left Down	DXImageTransform.Microsoft.Strips(motion='leftdown')
Strips Left Up	DXImageTransform.Microsoft.Strips(motion='leftup')
Strips Right Down	DXImageTransform.Microsoft.Strips(motion='rightdown')
Strips Right Up	DXImageTransform.Microsoft.Strips(motion='rightup')
Wipe Up	DXImageTransform.Microsoft.Blinds(direction='up', bands=1)
Wipe Down	DXImageTransform.Microsoft.Blinds(direction='down', bands=1)
Wide Right	DXImageTransform.Microsoft.Blinds(direction='right', bands=1)
Wipe Left	DXImageTransform.Microsoft.Blinds(direction='left', bands=1)

TABLE 8-5 *Page Transition Effects*

Try This!

Dynamic HTML Code Generator

Microsoft's Dynamic HTML site has an HTML code generator called the Master Sample that lets you try out a wide range of special effects. For each kind of effect, there are controls that let you manipulate the values of the parameters that affect what you see onscreen. Figure 8-18 shows that you can audition the effects and fine-tune the settings until you get it just the way you want it. Then you can copy and paste the code to create the effect on pages of your own. To generate some Dynamic HTML code, follow these steps:

1. Go to msdn.microsoft.com/workshop/samples/author/dhtml/DXTidemo/DXTidemo.htm. If this does not bring up the HTML code generator, its address may have changed—follow this book's link to the Dynamic HTML Code Generator.
2. Choose the type of Dynamic HTML effect you would like to explore. The choices are Filters or Transitions. In this example, choose Transitions to bring up the Transition controls.
3. Pull down the menu to select the specific effect you want to explore. In this example, pull down the Select a Transition menu, choose GradientWipe, and click the Play button to see what the effect causes onscreen.
4. Use the controls beneath the Play button to explore the customization settings. Following your instincts, set the controls however you want, and click the Play button to see what happens. In this example, set the gradient size to 0.25, make the wipe style go from left to right, and set the motion to reverse. Then click Play to audition this effect.
5. When you have the effect ready to use on your page, click the Copy Code to Clipboard button. Then click in the Notepad to position your cursor at the spot in your page where you want to insert this effect. Press CTRL-V, or pull down the Edit menu and choose Paste, to insert the effect into your page. Save the page and open it with your browser to see the effect happen on your page.

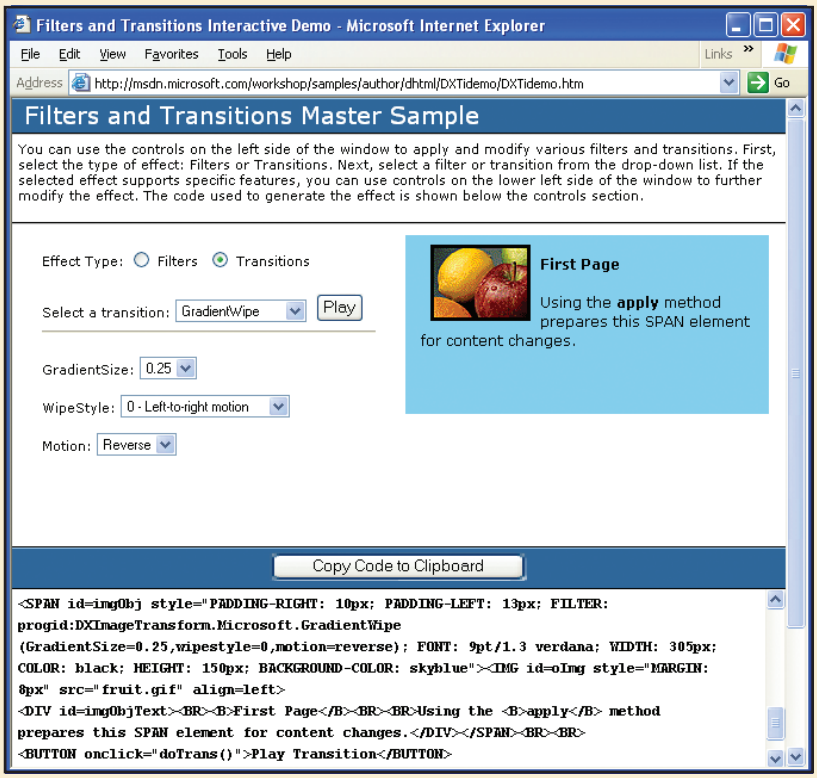


FIGURE 8-18 *Microsoft's Dynamic HTML online workshop includes this master sample that lets you select a transition and modify its parameters by manipulating the controls onscreen. To audition the effect, you click the Play button. When you have it working the way you want, click the Copy Code to Clipboard button. This enables you to paste the code onto your Web page.* ■

XML and XHTML

The hottest three letters in advanced Web design are XML, which stands for eXtensible Markup Language. As the word extensible implies, XML enables you to create special tags for encoding different kinds of data. Virtually any kind of data can be represented in XML.

What Is XML?

XML is a simple, self-describing markup language that enables computers to read and understand the structure of different kinds of documents and to exchange data across different operating systems, software applications, and hardware configurations without requiring any human intervention. Like HTML, XML has tags, but there is an important difference in how the tags are used. In HTML, the tags mostly define the appearance of the content. In XML, on the other hand, the tags define the structure of the data.

Another important difference between HTML and XML is that in HTML, the tags are specified by the World Wide Web Consortium (W3C). If you want to create a new HTML tag, you cannot do so on your own; rather, you propose the new tag to the W3C and work through a lengthy standardization process. With XML, on the other hand, you can create your own customized tags.

Many disciplines are working to create XML encodings to enable the exchange of data and the creation of new document types beneficial to the industry. To peruse dozens of discipline-based XML projects, go to www.xml.org and follow the links to the various focus areas.

What Is an XML Schema?

An **XML schema** is the structural definition of the types of elements that can appear in a document, the attributes each element may have, and the relationships among the elements. The best way to understand this is to compare the structure of a familiar document to its schema. A document with which everyone is familiar is your checkbook. For each entry in a checkbook, you record the check number, the date, the payee, and the amount of money you are paying. Figure 8-19 shows the XML schema that defines this data structure. The name of the file that contains this schema is *checkbook.xsd*. The filename extension XSD stands for XML Schema Definition.

checkbook.xsd - Notepad

File Edit Format View Help

```
<?xml version="1.0" ?>
<xs:schema id="checkbook" xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="checkbook">
      <xs:complexType>
         <xs:choice maxOccurs="unbounded">
            <xs:element name="check">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="number" type="xs:int" minOccurs="0" />
                     <xs:element name="date" type="xs:date" minOccurs="0" />
                     <xs:element name="payee" type="xs:string" minOccurs="0" />
                     <xs:element name="amount" type="xs:float" minOccurs="0" />
                  </xs:sequence>
               </xs:complexType>
            </xs:element>
         </xs:choice>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

FIGURE 8-19 *This is the XML schema in the file checkbook.xsd, which defines the elements and the structure of the checkbook document. I made up the names of the elements in this file. When you create a schema, you get to name the elements that are in it.* ■

Encoding Data in XML

Number	Date	Payee	Amount
1001	9/15/04	Columbia Gas	$248.29
1002	9/18/04	Sears	$327.99
1003	9/23/04	United Postal Service	$15.45

TABLE 8-6 *Three Hypothetical Checkbook Entries* ■

For the sake of this example, imagine that since opening your checking account, you have written the three checks illustrated in Table 8-6. To encode this data in XML, you must represent it inside the tags defined by the checkbook schema. Figure 8-20 shows such an encoding. Take special note of the second line of this file, which is a DOCTYPE declaration. For an XML document to be well formed, it must have a **DOCTYPE declaration**, which is a line of code that identifies the XML schema that defines the tag structure. In this example, the DOCTYPE is defining *checkbook.xsd* as the XML schema for this document. When all the tags in a document follow precisely the structural definitions in the schema, the document is said to *validate*. Documents that validate can be opened and processed with a variety of XML tools. Documents that do not validate are said to be *malformed* and will be rejected by XML tools that require strict adherence to the rules of XML. The XML document in Figure 8-20 validates against the *checkbook.xsd* schema and is well formed.

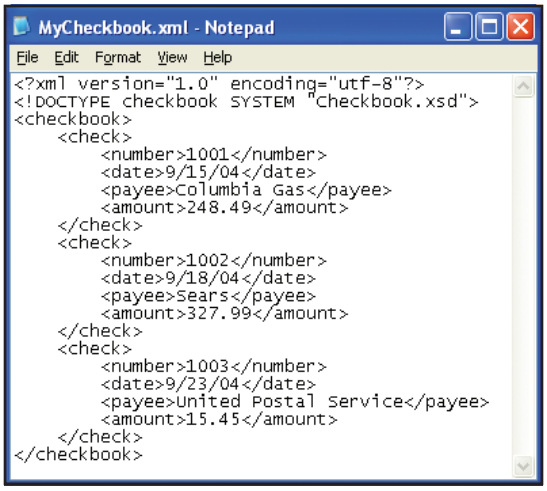


FIGURE 8-20 *This is an XML file that encodes the three checkbook entries in Table 8-6. Notice that the DOCTYPE statement at the beginning of this file defines checkbook.xsd as the schema containing the structural rules to which this document must adhere. If you compare the contents of this file to the schema shown earlier in Figure 8-19, you will see that this file follows the rules of the schema and is well formed.* ■

Editing the Data in an XML File

Because XML files are plain text, you can edit them with any text editor, such as the Notepad. If you have a lot of data to edit, or if the schema is complicated, however, you are better off using an XML editor to edit the data. When I edit an XML file, for example, I use the XML editing tools in Visual Studio .NET, Microsoft's premier suite of tools for creating Web applications. Figure 8-21 shows how Visual Studio reads the XML schema of a file and presents you with a visual tool for editing the data in a spreadsheet view. Any change or addition you make to this visual view of the data gets updated by Visual Studio in the XML view of the file. One of the advantages of editing an XML file with a tool such as Visual Studio is that you can be sure the XML will validate and be well formed. When you edit your XML with a text editor, on the other hand, you run the risk of making errors that may cause problems down the road.

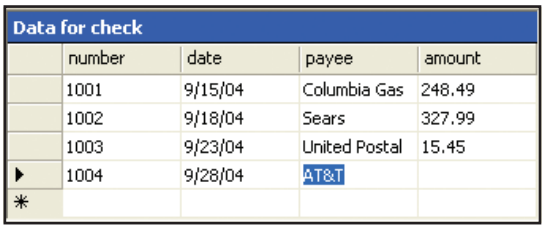


FIGURE 8-21 *Visual Studio .NET contains this visual tool for editing the contents of an XML file. Any change or addition you make to this visual view of the data gets updated by Visual Studio in the XML view of the file.* ■

What Is the Extensible Stylesheet Language (XSL)?

Much XML data never appears onscreen. In business-to-business applications, for example, there are many server-to-server communications that users never see. Data that never appears onscreen does not need stylistic layout. What do you do, however, when you need to display XML data onscreen? How do you transform that data into a representation that the browser can display with good style? That is where XSL comes in.

The **extensible stylesheet language (XSL)** is an XML dialect that Web designers use to specify the styling, layout, and pagination of the structured content in an XML document for some targeted presentation medium, such as a Web browser, a printer, an eBook, a screen reader, or a hand-held device. The stylesheet language elements are defined by the XSL Working Group of the W3C Style Activity. The official W3C documentation is at www.w3.org/TR/xsl.

What Is the XSL Transformation Language?

The **XSL Transformation (XSLT)** language is an XML dialect that Web designers use to transform documents from one format into another. Although it is beyond the scope of this book to teach XSLT, the concept of document transformation is important for any IT professional to understand.

XSLT enables you to define a template into which you can flow part or all of the content of an XML document. The template combines the data from the XML file with device-specific formatting codes. If you want to format the document for display on a printer, for example, you would embed printer codes in the XSLT template. To display the document on a cell phone, you would use an XSLT template that creates codes according to the wireless application protocol (WAP). If a Web browser is the targeted display medium, you would use an XSLT template that formats the XML data in HTML for display onscreen. Figure 8-22 shows an XSLT template, for example, which formats the *checkbook.xml* file for display in a browser. Figure 8-23 shows how the browser displays the checkbook data via the HTML generated by the template in Figure 8-22.

For each check in the checkbook…

…these XSL commands select the values of the check fields.

Checkbook.xslt

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:template match="/">
<html>
<body>
<table style="border-color:silver; border-width:2;
    border-style:groove; background-color:#C6E4FE;">
  <xsl:for-each select="checkbook/check">
    <tr style="background-color:#F1F9FF;">
      <td>
        <xsl:value-of select="number"/>
      </td>
      <td>
        <xsl:value-of select="date"/>
      </td>
      <td>
        <xsl:value-of select="payee"/>
      </td>
      <td align="right">
        <xsl:value-of select="amount"/>
      </td>
    </tr>
  </xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

FIGURE 8-22 *This XSLT template reads the XML file checkbook.xml and formats the data in HTML for display by a Web browser. Compare this code to Figure 8-23, which shows how the browser displays the table. Just as this template formats the data for display in HTML, so also can you create XSLT templates for transforming XML files for different devices, including printers, PDAs, and cell phones. You can also use XSLT to transform documents into different file formats, such as PDF, DOC, XLS, and RTF.* ■

What Are the Flavors of XHTML?

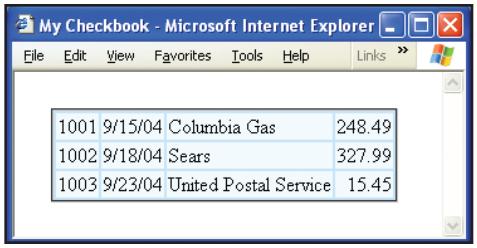


FIGURE 8-23 *This is how the browser displays the XML file checkbook.xml when it is transformed into HTML by the XSLT template in Figure 8-22. If you compare the XML data in Figure 8-20 to the template in Figure 8-22, you can begin to grasp the significance of being able to transform data in this manner.* ■

Now that you have learned the basics of XML, it is time to revisit the definition of XHTML. In Chapter 6, you learned that **XHTML** is a reformulation of HTML in XML. Now it is time to get more specific about the schema that differentiate the structure of HTML and XHTML files.

Most Web page editors begin a new page by declaring that the structural rules will use a loose, as opposed to strict, definition of HTML. This loose definition enables you to make use of presentation elements that still are in use today but will fade in the future when style sheets achieve widespread use. To declare that a Web page uses the loose definition, you insert the following declaration prior to the `<html>` tag at the top of the page:

L 8-31
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
        "http://www.w3.org/TR/html4/loose.dtd">
```

If your page avoids the deprecated presentation tags and instead uses style sheets to define the presentation of HTML elements onscreen, you can use the strict version of HTML by making your declaration read as follows:

L 8-32
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
       "http://www.w3.org/TR/html4/strict.dtd">
```

XHTML also has loose and strict versions. The loose version is the same as HTML transitional except for changes due to the differences between XML and SGML. To use the loose version of XHTML, you make the DOCTYPE read as follows:

L 8-33
```
<!DOCTYPE html  PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
       "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

Similarly, the strict version of XHTML is the same as HTML strict except for changes due to the differences between XML and SGML. The DOCTYPE for strict XHTML is

L 8-34
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

For an excellent discussion of the differences between strict and transitional XHTML, go to www.w3.org/TR/xhtml1.

What Is an XML Module?

An **XML module** is a collection of semantically related XML elements and attributes oriented toward accomplishing a certain task or function. An excellent example of modularization is the manner in which the W3C has organized into modules the various parts of the **Synchronized Multimedia Implementation Language (SMIL)**. SMIL is an XML-based language that was created by the W3C for the purpose of enabling developers to include multimedia events in Web documents.

When this book went to press, the SMIL modules were organized into ten functional areas: (1) timing, (2) time manipulations, (3) animation, (4) content control, (5) layout, (6) linking, (7) media objects, (8) metainformation, (9) structure, and (10) transitions.

What Is XHTML+SMIL?

A language profile is the combination of modules to create an XML language designed to meet certain objectives. The W3C has created a language profile you can use to add multimedia functionality to XHTML. The name of this profile is **XHTML+SMIL**. Its goal is to permit the Web designer to use SMIL animations, timings, and transitions within a conventional HTML or CSS page layout. Thus, XHTML+SMIL omits modules related to layout. As a result, the Web designer can create a layout via style sheets or HTML and include SMIL animations, timings, and transitions in traditional Web page elements.

Microsoft has created an implementation of XHTML+SMIL called **HTML+TIME** that works with Internet Explorer versions 5.5 and later. True to its name, the most important part of HTML+TIME is the timing module, because that is how you create synchronized multimedia events. An excellent example of a multimedia event that requires timing is the captioning of a video, which requires you to display onscreen subtitles at precise times in sync with the video. This chapter concludes with a Try This! exercise that lets you experience this for yourself onscreen.

Try This!

The Kennedy Moon Challenge Project

Because of its built-in support of HTML+TIME, the Internet Explorer Web browser gives you considerable multimedia authoring capability without requiring you to own any other tools. This exercise provides an example of how you can caption a video by displaying onscreen subtitles that appear at precise times in sync with the video. All you need to accomplish this is the Notepad text editor and the IE browser, version 5.5 or later. The video you will caption is a famous passage from President John F. Kennedy's famous moon challenge speech. To caption the video and play it onscreen, follow these steps:

1. Go to this chapter's section of this book's Web site. You will see a link to click to download a famous 30-second video clip from JFK's moon-challenge speech. Click the link and save the file in your computer's *website* folder. Do not change the file's name, which is KennedyMoonChallenge.avi. The file size is 3.6 megabytes.
2. Pull down the Notepad's File menu and choose New to begin a new file. Type the following code to get the page started. Notice that the HTML tag has an XML namespace (`xmlns`) attribute that associates the prefix *t* with Microsoft's HTML+TIME schema:

```
<html xmlns:t="urn:schemas-microsoft-com:time">
<head>
<title>JFK SMIL Example</title>
</head>
<body>
</body>
</html>
```

3. Pull down the Notepad's File menu and choose Save As; use the Save controls to save this file in your *website* folder under the filename *Kennedy.html*.
4. In the head section of the document, type the following `<style>` tag to make the page use HTML+TIME version 2:

```
<head>
<title>JFK SMIL Example</title>
<style>
    .time{ behavior: url(#default#time2);}
</style>
</head>
```

5. In the body of the document, position your cursor between the `<body>` start and `</body>` stop tag and type the following code. This creates a table with two rows. The top row shows the video, and the bottom row uses the sequence element seq to display the captions in time with the video. You can also download this code from this book's Web site, where the filename is *KennedyTable.txt*:

Try This! *continued*

```
<body>
<table cellspacing="0" style="background-color:White;font:bold;font-size:11pt;">
<tr>
    <td>
        <t:video class="time" ID="movie" src="KennedyMoonChallenge.avi" />
    </td>
</tr>
<tr>
    <td id="caption" align="center">
        <t:seq id="txSeq" class="time" begin="movie.begin+.5">
        <span id="1" class="time" dur="2">We choose to go to the moon in</span>
        <span id="2" class="time" dur="2.5">this decade and do the other things,</span>
        <span id="3" class="time" dur="2.15">not because they are easy,</span>
        <span id="4" class="time" dur="2.0">but because they are hard.</span>
        <span id="5" class="time" dur="1.8">Because that goal</span>
        <span id="6" class="time" dur="1.8">will serve to organize</span>
        <span id="7" class="time" dur="1.8">and measure the best</span>
        <span id="8" class="time" dur="2.2">of our energies and skills.</span>
        <span id="9" class="time" dur="2.1">Because that challenge is one</span>
        <span id="10" class="time" dur="1.9">that we're willing to accept,</span>
        <span id="11" class="time" dur="2.2">one we are unwilling to postpone,</span>
        <span id="12" class="time" dur="2.3">and one we intend to win,</span>
        <span id="13" class="time" dur="3">and the others too.</span>
        </t:seq>
    </td>
</tr>
</table>
</body>
```

5. Save the *Kennedy.html* file and open it with your browser. Notice how the captions appear onscreen in time with the video. You have successfully used XML to caption a video.
6. By modifying this file, you can make it display captions for any video. To change the video, replace *KennedyMoonChallenge.avi* by the movie you want to play instead. To change the captions, edit the `<span>` elements within which the caption attributes are set. The attribute *dur* creates the timing. In SMIL, *dur* stands for duration and specifies how many seconds the element will last. In this example, *dur* determines how many seconds the caption will remain onscreen.
7. This exercise demonstrates just one of the many multimedia capabilities of HTML+TIME. For more on the built-in multimedia capabilities of the IE Web browser, follow this book's Web site link to the HTML+TIME language reference.

Chapter 8 Review

Chapter Summary

After reading this chapter and completing the step-by-step tutorials and Try This! exercises, you should understand the following facts about creating active Web pages:

Introduction to Scripting

- A static Web page is a document in which the content is fixed in HTML codes that make the document always read the same when viewed in a browser. An active Web page uses the browser's window as a display surface through which the user can interact with dynamic objects onscreen.
- Scripting is the act of writing little computer programs that can enhance the appearance and functionality of a Web page. Browsers render Web pages by placing objects onscreen. Scripts let you grab hold of those objects to make them do special things.
- There are many brands of scripting languages. Most well known is JavaScript, the language that runs client-side in the browser without requiring any server-side processing. On the server side, VBScript and JScript are Microsoft's Active Server Page (ASP) languages. Other popular server-side languages include C#, Java, and J#, Microsoft's version of Java.
- You can put JavaScript in the head or in the body section of a Web page. Scripts can also reside in a separate file called an include file, which gets included in the page at runtime.
- If a script is brief and is not used a lot, you can simply type it into the body of the page. If you find yourself typing the same code often, however, it is better to put it inside a reusable function that goes into the head section of the Web page.
- A function is a named procedure you can call upon by name any time you need to execute the code that function contains. When you call the function, you can pass to it one or more parameters that preset the values of variables that the function manipulates. When the function finishes executing, it can return values to the script that called it.
- A variable is a place in the computer's RAM that remembers, or stores, the value of something changeable. It is called a variable because its value is subject to change. A variable name is the identifier used to refer to, or call upon, a place in computer memory that stores the value of a variable.
- A string is a sequence of one or more alphanumeric characters. A string variable is a place in computer memory that remembers, or stores, the alphanumeric characters in a string.
- A numeric variable is a place in computer memory that remembers, or stores, a number. In a script, the numbers can be integers or floating point. An integer is a whole number with no decimal point. A floating point number has a decimal point with one or more numbers after the decimal point.
- An operator is a symbol that causes a script to perform some kind of action on a variable or a value. The most fundamental operator is the assignment operator, which assigns values to variables. The assignment operator uses the = symbol.
- To concatenate means to join strings together via the concatenation operator. In JavaScript, the concatenation operator is the + sign.
- In computing, an object is a self-contained entity consisting of properties and methods enabling you to do something programmatically. A method is a little computer program that has been built into an object to enable the object to do something. A property is an attribute of an object that has a value.
- In computer programming, an event is an action that provides an opportunity to trigger a script, which can use objects to inspect properties and execute methods that make things happen onscreen or keep records behind the scenes. The most commonly used events are (1) the mouseover, which fires when the user mouses over an object onscreen; (2) the click, which fires when the user clicks the mouse; (3) the double-click, which the user triggers by clicking twice quickly; and (4) the page load, which fires when the user visits a Web site and a page first comes onscreen.

Document Object Model (DOM)

- The document object model (DOM) is the official W3C structural definition of the objects, methods, and properties that comprise documents on the World Wide Web. The latest version is at www.w3.org/DOM.
- In JavaScript, the most commonly used DOM objects are the ones that provide access to the HTML elements that comprise the document displayed inside the browser window. Of particular importance are the elements in the forms through which the user interacts by selecting things, entering textual information, and clicking to submit the form.
- In addition to defining the objects that enable a script to manipulate elements on a Web page, the W3C has defined the intrinsic events that can trigger such a script. Two popular events are onmouseover and onmouseout, which you use to create rollover effects onscreen. When the user mouses over something, the onmouseover event fires, causing your script to do something onscreen. Similarly, when the user mouses out of something, the onmouseout event fires, providing another opportunity to run a script.
- To access DOM objects in a script, you use dot notation to refer to the objects you want to manipulate. Following the document's hierarchical structure, dot notation places to the left elements that are structurally higher than elements further down the tree. An example is document.title, which you use to retrieve or set the title of the page.
- An array is a named table of memory locations in which values can be stored. When a Web page loads, the browser creates arrays for the images, forms, links, anchors, and all the other elements onscreen. As the browser encounters these objects on the page, it places them into these arrays. The arrays are indexed sequentially, beginning with zero. The first image on the page, therefore, goes into slot 0 of the images array. You could refer to it in a script as document.images[0].
- An easier way to access elements on a Web page is to name the elements and then refer to them by name. An image named MyPhoto is easier to refer to, for example, as MyPhoto.width instead of document.images[0].width.
- An alert box is a window that a script creates by executing the `alert()` method of the JavaScript window object. Inside the parentheses of the `alert()` method, you insert the string of characters and variables you want displayed in the alert box. At runtime, when the script encounters the alert, the script pauses while you study the contents of the alert box onscreen. When you click to dismiss the alert box, the script continues executing. By putting alert boxes in strategic places down through the path of execution in your code, you can step through the code and diagnose the point at which something is going wrong.
- A rollover is a special graphical effect you create by using the JavaScript onmouseover and onmouseout event handlers. When the user mouses over something, the onmouseover event fires, causing your script to do something onscreen. Similarly, when the user mouses out of something, the onmouseout event fires, providing another opportunity to run a script. In the most common kind of rollover, a script changes the source of the image when the user mouses over it and reverts to the original image when the user mouses out of it.

Maintaining State in Cookies

- A cookie is a place in the computer's memory where browsers can store information about the user. If someone buys an audio CD by Britney Spears, for example, the site might create a cookie indicating that the user likes pop music. The next time the user visits the site, it might display ads for similar pop music titles.
- There are two kinds of cookies, namely, persistent cookies that are stored on the user's hard disk and per-session cookies that are stored in RAM. Because persistent cookies are stored on disk, they survive from session to session, even after the user closes the Web browser and turns off the computer. Per-session cookies, on the other hand, evaporate when the user closes the Web browser, which frees the RAM.
- The Internet needs cookies to maintain state from screen to screen. Virtually all sites that have you log on and off, including all of the Internet's e-commerce sites, use per-session cookies to maintain state.

- Knowledgeable users can find the persistent cookie files and read their contents with any text editor. If these contents are unencrypted, the cookies can be read in plain text. It is appropriate to use persistent cookies to store information that would not cause a security problem if sniffed. The advantage of using persistent cookies is that a server can keep data on the user's PC, thereby avoiding the need to store that data in a server-side database.

Cascading Style Sheets

- A Cascading Style Sheet (CSS) is a set of rules that define styles to be applied to entire Web pages or individual Web page elements. Each rule consists of a selector followed by a set of curly braces containing the style properties and their values. The selector can be an HTML element, a user-defined style known as a class, or the ID of a specific element on a page. There are three ways of applying cascading style sheets to a Web page: external, embedded, and inline.
- An external CSS keeps all the style definitions in a separate CSS file that you include in a Web page at runtime by using the `<link>` tag to apply the style sheet to the page.
- An embedded CSS is a style sheet that gets copied physically into the head of the Web page and applies to the Web page as a whole.
- An inline CSS is a style sheet that applies to only one page element, so it gets copied "inline" on the page with that element.
- The W3C invented the HTML inline `<span>` start and `</span>` stop tags to provide you with a way to stylize part, instead of all, of a Web page element. An example is

```
Notice how <span style="color: yellow">
yellow words</span> appear onscreen.
```

- Other situations may arise in which you want to apply a style to larger divisions of a document at the block level. You create block-level divisions with the `<div>` start and `</div>` stop tags, where `<div>` stands for division. The syntax for the `<div>` tag is exactly the same as for the `<span>` tag. Because `<div>` is a block-level tag, however, the browser begins a new line at the beginning of the division.
- In cascading style sheets, a class is a named definition of one or more styles. You create the class by prefixing its name with a dot in the CSS file.
- On the cutting edge of cascading style sheets is a feature called absolute positioning, which enables you to position page elements onscreen based on x,y coordinates. The upper-left corner of the browser window is position 0,0.
- To provide control over the order in which the objects appear onscreen, absolute positioning has an attribute called the z-index, which tells the browser the order in which to display objects that overlap. The lower the z-index value, the sooner the layer displays onscreen. In other words, an item with a lower z-index will appear underneath overlapping items with higher z-index values.

Dynamic HTML

- Dynamic HTML is a term invented by Microsoft to refer to the animated Web pages you can create by using the DOM to combine HTML with style sheets and scripts that bring Web pages to life.
- JavaScript has a method called `setInterval()` that you can use to set a timer that fires periodic timer events. By manipulating the absolute positioning values of x and y dynamically during these timer events, you can create an animation onscreen. If you know your math, there is no limit to the patterns of movement you can create onscreen.
- A gradient is a graphical effect created by colors fading gradually across or down the screen. You can make millions of different gradients by manipulating the start and stop color strings in the `gradient()` method of the DXImageTransform object.
- A page transition is the style or manner in which the screen changes when the browser brings up a new document and displays it onscreen. You can create a wide range of page transitions via the `Barn()`, `Blinds()`, `Checkboard()`, `Iris()`, `RandomDissolve()`, `Slide()`, and `Strips()` methods of the DXImageTransform object.
- Microsoft's Dynamic HTML site has an HTML code generator called the Master Sample that lets you try out a wide range of special effects. For each kind of effect, there are controls that let you

manipulate the values of the parameters that affect what you see onscreen. You can audition the effects and fine-tune the settings until you get it just the way you want it. Then you can copy and paste the code to create the effect on pages of your own.

XML and XHTML

- XML is a simple, self-describing markup language that enables computers to read and understand the structure of different kinds of documents and to exchange data across different operating systems, software applications, and hardware configurations without requiring any human intervention. Like HTML, XML has tags, but there is an important difference in how the tags are used. In HTML, the tags mostly define the appearance of the content. In XML, on the other hand, the tags define the structure of the data.
- Another important difference between HTML and XML is that in HTML, the tags are specified by the World Wide Web Consortium (W3C). If you want to create a new HTML tag, you cannot do so on your own; rather, you propose the new tag to the W3C and work through a lengthy standardization process. With XML, on the other hand, you can create your own customized tags. At www.xml.org, there are many focus areas in which various disciplines are creating XML tags for use within their industries.
- An XML schema is the structural definition of the types of elements that can appear in a document, the attributes each element may have, and the relationships among the elements.
- For an XML document to be well formed, it must have a DOCTYPE declaration, a line of code at the top of the file that identifies the XML schema that defines the tag structure. When all the tags in a document follow precisely the structural definitions in the schema, the document is said to validate. Documents that do not validate are said to be malformed and will be rejected by XML tools that require strict adherence to the rules of XML.
- The extensible stylesheet language (XSL) is an XML dialect that Web designers use to specify the styling, layout, and pagination of the structured content in an XML document for some targeted presentation medium, such as a Web browser, a printer, an eBook, a screen reader, or a hand-held device.
- The XSL Transformation (XSLT) language is an XML dialect that Web designers use to transform documents from one format into another, such as HTML, PDF, DOC, XLS, and RTF.
- XHTML is a reformulation of HTML in XML. The Web is a work in progress and is transitioning to rely more on XML than HTML. You use the DOCTYPE declaration to specify how strictly to adhere to the new rules.
- A so-called HTML Transitional document type definition (DTD) enables you to make use of presentation elements that are still in use today but will fade in the future when style sheets achieve widespread use. If your page avoids the deprecated presentation tags and instead uses style sheets to define the presentation of HTML elements onscreen, you can use the DTD called HTML Strict.
- XHTML also has loose and strict versions. The loose version is the same as HTML Transitional except for changes due to the differences between XML and SGML. Similarly, the strict version of XHTML is the same as HTML Strict except for the differences between XML and SGML. These differences are explained at www.w3.org/TR/xhtml1.
- An XML module is a collection of semantically related XML elements and attributes oriented toward accomplishing a certain task or function. An excellent example of modularization is the manner in which the W3C has organized into modules the various parts of the Synchronized Multimedia Implementation Language (SMIL), an XML-based language that enables you to include multimedia events in Web documents.
- The W3C has created a language profile called XHTML+SMIL that enables you to create a layout via style sheets or HTML and include SMIL animations, timings, and transitions in traditional Web page elements. Microsoft has created an implementation of XHTML+SMIL called HTML+TIME that works with Internet Explorer versions 5.5 and later. Due to its built-in support of HTML+TIME, the Internet Explorer Web browser gives you considerable multimedia authoring capability without requiring you to own any other tools.

Key Terms

absolute positioning *(38)*
active Web page *(2)*
alert box *(24)*
assignment operator *(7)*
cascading style sheet (CSS) *(32)*
class *(37)*
concatenate *(9)*
concatenation operator *(9)*
cookie *(27)*
DOCTYPE declaration *(47)*
document object model (DOM) *(17)*
dynamic HTML *(40)*
embedded CSS *(33)*
event *(14)*
extensible stylesheet language (XSL) *(48)*
external CSS *(33)*
floating point *(7)*
function *(3)*
gradient *(42)*
HTML+TIME *(50)*
include file *(3)*
inline CSS *(33)*
integer *(7)*
JavaScript *(3)*
method *(13)*
numeric variable *(7)*
object *(13)*
operator *(7)*
page transition *(42)*
property *(14)*
rollover *(25)*
scripting *(2)*
static Web page *(2)*
string *(7)*
string variable *(7)*
Synchronized Multimedia Implementation Language (SMIL) *(50)*
variable *(6)*
variable name *(7)*
XHTML *(49)*
XHTML+SMIL *(50)*
XML *(46)*
XML module *(50)*
XML schema *(46)*
XSL Transformation (XSLT) *(48)*
z-index *(38)*

Key Terms Quiz

1. A(n) _____ uses the browser's window as a display surface through which users can interact with dynamic objects onscreen.
2. _____ is the act of writing little computer programs that can enhance the appearance and functionality of a Web page.
3. The most well-known scripting language is _____, which runs client-side in the browser without requiring any server-side processing.
4. A(n) _____ is a place in the computer's RAM that remembers, or stores, the value of something changeable.
5. A(n) _____ is a sequence of one or more alphanumeric characters.
6. The _____ is the official W3C structural definition of the objects, methods, and properties that comprise documents on the World Wide Web.
7. A(n) _____ is a set of rules that define styles to be applied to entire Web pages or individual Web page elements.
8. _____ enables you to position page elements onscreen based on x,y coordinates. The upper-left corner of the browser window is position 0,0.
9. _____ is a term invented by Microsoft to refer to the animated Web pages you can create by using the DOM to combine HTML with style sheets and scripts that bring Web pages to life.
10. _____ is a simple, self-describing markup language that enables computers to read and understand the structure of different kinds of documents and to exchange data across different operating systems, software applications, and hardware configurations without requiring any human intervention.

Multiple-Choice Quiz

1. If a script is brief and is not used a lot, you can simply type it into the body of the page. If you find yourself typing the same code often, however, it is better to put it inside a reusable:
   a. cookie
   b. function
   c. method
   d. property
2. In JavaScript, the concatenation operator uses the symbol:
   a. =
   b. *
   c. \
   d. +
3. Which event fires when the user visits a Web site and a page first comes onscreen?
   a. onclick
   b. ondblclick
   c. onmouseover
   d. onload
4. You can step through your script and diagnose the point at which something is going wrong by putting what diagnostic aid in strategic places down through the path of execution in your code?
   a. Alert box
   b. Documentation
   c. Help index
   d. Rollover
5. Which kind of cookie resides on the user's hard disk and survives even if the computer is rebooted?
   a. Incessant
   b. Insistent
   c. Persistent
   d. Per-session
6. The Internet needs per-session cookies to provide a way to:
   a. count hits on a Web page
   b. keep the socket open
   c. maintain state from screen to screen
   d. cut down on the amount of spam
7. Which kind of cascading style sheet keeps the style definitions in a separate CSS file?
   a. Embedded CSS
   b. External CSS
   c. Inline CSS
   d. Internal CSS
8. Which pair of tags would you use to stylize one small part of a Web page element without causing the browser to begin a new line onscreen?
   a. `<body> </body>`
   b. `<div> </div>`
   c. `<span> </span>`
   d. `<table> </table>`
9. In absolute positioning, what is the name of the attribute that enables you to control the order in which the Web page elements appear onscreen?
   a. x
   b. y
   c. z
   d. z-index
10. Which DTD enforces the structural rules of XML and demands that the Web page use style sheets in lieu of deprecated HTML positioning elements?
    a. HTML Transitional
    b. HTML Strict
    c. XHTML Transitional
    d. XHTML Strict

Essay Quiz

1. The *hello.html* script you created in this chapter wrote "Hello, world!" onscreen. Modify the script to say a little more. For example, edit the string "Hello, world!" to make it say something longer, such as, "Hello, world, from JavaScript!" Experiment with putting the break tag `<br>` somewhere in the midst of the string. What does the `<br>` cause onscreen when you view the page with a Web browser?
2. The *variables.html* example in this chapter used information for a person with the first name of Santa and the last name of Claus. Modify the *variables* script to make the names be your own first name and last name. Then run the script by opening the *variables* file with your Web browser. Does the script operate correctly when it computes your full name by concatenating your first and last names?

3. Between the first and last names of the *variables.html* example, add a variable to hold the person's middle initial, and modify the example to include this initial when the script concatenates the full name. When you make up a name for the variable that is going to hold the middle initial, make the variable name be something that indicates it is meant to hold a middle initial. Remember to prefix the name of the middle initial variable with an *s* to indicate its value will be a string. What name did you create for the variable that holds the value of the person's middle initial?
4. Suppose you want to change the look of an individual page element that appears many times on a Web page. You want to change just one occurrence of the element, without altering other renderings of this element onscreen. What kind of cascading style sheet would you use to accomplish this?
5. In the *rockpile.html* page, the rock pile you created with absolute positioning contained only three images. Increase the size of the rock pile by creating more instances of the three rocks. Vary the size and position of the rocks. Use the z-index to layer the rocks. See how creative you can be in creating a multi-layered rock pile. Make the rock pile contain at least a dozen rocks.

Lab Projects

• Lab Project 8-1: Dynamic Content

A static page that never changes does not project a very dynamic image out on the Web. Imagine that your school or workplace has hired you to transform its home page from static to dynamic. Your task is to create active Web page elements into which the site will flow dynamic content instead of displaying the same information every time someone visits the site. Before bringing these active elements online, your superior has asked you to submit a plan in which you (1) propose which parts of the page should become dynamic, (2) describe the manner in which they will become active, and (3) define the source of the content that will feed into the page. Use your word processor to create this plan. In developing your proposal, consider the following issues:

- **DOM Elements** The DOM defines the page elements you can manipulate dynamically. List the DOM elements you feel should be manipulated dynamically at your site. For each element you list, describe what the dynamism will be. The page title, for example, is a DOM element. Consider whether there is any active content you would want to put into the title of the page.
- **Rotating Banners** Many Web sites have banners that change depending on factors such as the season, the time of day, or user history stored in cookies that keep track of what the user has been doing at the site. Consider whether the banner on your page should be static or dynamic. If dynamic, describe how and when the banner will change.
- **RSS Feeds** In Chapter 2, you learned about RSS. Because the content of an RSS feed changes, it can be considered a dynamic element onscreen. Can you think of any ways in which an RSS feed, perhaps from other parts of your site or from related sites, would provide relevant dynamic content for your school or company home page?
- **Time and Date** Consider whether your site should display the current date or time. If so, recommend the strategy you will use to make sure your page displays the date or time in a format that is meaningful and unambiguous to the end-user.

If your instructor has asked you to hand in this assignment, make sure you put your name at the top of your recommendation; then copy it onto a disk or follow the other instructions you may have been given for submitting this assignment.

• Lab Project 8-2: Using Cookies

Imagine that your superior has heard about cookies enabling certain Web sites to keep user information on the client side without requiring a server-side database. You have been asked to recommend whether your workplace could use this capability to reduce the load on your server. Use your word processor to write an essay in which you take a position on cookies and recommend how your site should use them. In developing your position, consider the following issues:

- **Data** What kinds of data will you store in the cookies, and how will that data be used by your site?
- **Expiration** How long should the cookies persist on the user's hard drive, if at all? In answering this question, consider the nature of the data, how it affects your site, and the amount of time after which the data would become irrelevant, if any.
- **Security** Knowledgeable users can inspect the contents of persistent cookies that are stored on their hard drives. Consider whether the persistent cookies your site creates will need to be encrypted to prevent their being read in clear text.

If your instructor has asked you to hand in this recommendation, make sure you put your name at the top of your essay; then copy it onto a disk or follow the other instructions you may have been given for submitting this assignment.

• Lab Project 8-3: Style Sheet Strategy

Style sheets provide a way to maintain a consistent look and feel for Web page elements across all the pages at your site, without requiring you to edit each page every time your school or company decides to change the style of an element onscreen. Imagine that your superior has put you in charge of creating a style-sheet strategy for your school or company Web site. Your task is to develop a set of guidelines defining which elements of your site's Web pages will be controlled by style sheets and specifying the techniques your team's Web authors should use in applying the styles to your site's content. Use your word processor to write an essay in which you present these guidelines, taking into account the following issues:

- **Cascading** It is possible to link more than one style sheet to a document. Styles defined by the first style sheet cascade onto the second style sheet, which can either redefine them or leave them alone. If there is a third style sheet, the cascade continues. There is no limit to the number of style sheets that can be on the cascade. If you have a large organization, consider whether there should be an institutional style sheet that defines elements that will be stylized site-wide, followed by departmental style sheets that define certain other styles for use within a department.
- **Classes** Style sheets enable you to create a named class of style rules that apply only to elements that have the `class` attribute set to that name. Consider whether your site should use style classes, and if so, define the class names and describe what each class will do.
- **Conventions** List the coding conventions you want your Web authors to follow in regard to style sheets. Specify the Web page elements that should never be modified so they will always inherit their styles from the cascade. Conversely, list the elements that should use the `class` attribute to identify the name of the style class to which they belong.

If your instructor has asked you to hand in these guidelines, make sure you put your name at the top of your essay; then copy it onto a disk or follow the other instructions you may have been given for submitting this assignment.