CHAPTER 9

Managing State Information and Security

In this chapter, you will:

- Learn about state information
- Save state information with hidden form fields, query strings, and cookies
- Learn about security issues

The Web was not originally designed to store information about a user's visit to a Web site. However, the ability to store user information, including preferences, passwords, and other data, is very important because it allows you to improve the usability of a Web page. The three most common tools for maintaining state information are hidden form fields, query strings, and cookies, which you will study in this chapter. Given the sensitive nature of user information, it's also essential that you have a good understanding of the JavaScript security issues described in this chapter.

Understanding State Information

Hypertext Transfer Protocol (HTTP) manages the hypertext links used to navigate the Web and ensures that Web browsers correctly process and display the various types of information contained in Web pages. Information about individual visits to a Web site is called **state information**. HTTP was originally designed to be **stateless**, which means that Web browsers stored no persistent data about a visit to a Web site. The original stateless design of the Web allowed early Web servers to quickly process requests for Web pages, since they did not need to remember any unique requirements for different clients. Similarly, Web browsers did not need to know any special information to load a particular Web page from a server. Although this stateless design was efficient, it was also limiting; because a Web server could not remember individual user information, the Web browser was forced to treat every visit to a Web page as an entirely new session. This was true regardless of whether the browser had just opened a different Web page on the same server. This design hampered interactivity and limited the amount of personal attention a Web site could provide. Today, there are many reasons for maintaining state information. Among other things, maintaining state information allows a server to:

- Customize individual Web pages based on user preferences.
- Temporarily store information for a user as a browser navigates within a multipart form.
- Allow a user to create bookmarks for returning to specific locations within a Web site.
- Provide shopping carts that store order information.
- Store user IDs and passwords.
- Use counters to track the number of times a user has visited a site.

Saving State Information with Hidden Form Fields

A special type of form element, called a **hidden form field**, is not displayed by the Web browser and, therefore, allows you to hide information from users. You create hidden form fields with the <input> element. Hidden form fields temporarily store data that needs to be sent to a server along with the rest of a form, but that a user does not need to see. Examples of data stored in hidden fields include the result of a calculation or some other type of information that a program on the Web server might need. You create hidden form fields by using the same syntax used for other fields created with the <input> element: <input type="hidden">. The only attributes that you can include with a hidden form field are the name and value attributes.

Creating a Calculator Script with Storage Functionality

To get some practice saving state with hidden form fields, you will now use JavaScript to create a calculator script using push buttons and onclick event handlers. You will use a variable named inputString to contain the operands and operators of a calculation. After a calculation is added to the inputString variable, the calculation is performed using the eval() function. The script will include a single function named updateString() that accepts a single value representing a number or operator. The value is then added to the inputString variable using the += assignment operator. After the inputString variable is updated, it is assigned as the value of a text box in a form.

To create a calculator script:

1. Create a new document in your text editor.

2. Type the <!DOCTYPE> declaration, <html> element, header information, and <body> element. Use the strict DTD and “Calculator” as the content of the <title> element.
3. Add the following script section to the document head:

```
<script type=”text/javascript”>
/* <![CDATA[ */
/* ]]> */
</script>
```

4. Add the following function to the script section. This function is used to update the inputString variable:

```
var inputString = “”;
function updateString(value) {
    inputString += value;
    document.forms[0].input.value = inputString;
}
```

5. Add the following <form> and <p> elements to the document body:

```
<form action=””>
<p>
</p>
</form>
```

6. Add a text box named input and a break element to the paragraph in the form, as follows:

```
<input type=”text” name=”input”
style=”width: 170px” /><br />
```

7. Add the following <input> elements to the form; these elements create buttons representing the numbers and calculator operators. Each element sends a value to the updateString() function, using an onclick method:

```
<input type=”button” name=”seven”
style=”width: 40px” value=”7”
onclick=”updateString(‘7’)” />
<input type=”button” name=”eight”
style=”width: 40px” value=”8”
onclick=”updateString(‘8’)” />
<input type=”button” name=”nine” style=”width: 40px”
   value=”9” onclick=”updateString(‘9’)” />
<input type=”button” name=”div” style=”width: 40px”
   value=”/” onclick=”updateString(‘/’)” /><br />
<input type=”button” name=”four” style=”width: 40px”
   value=”4” onclick=”updateString(‘4’)” />
<input type=”button” name=”five” style=”width: 40px”
   value=”5” onclick=”updateString(‘5’)” />
<input type=”button” name=”six” style=”width: 40px”
   value=”6” onclick=”updateString(‘6’)” />
<input type=”button” name=”times”
style=”width: 40px” value=”*”
onclick=”updateString(‘*’)” /><br />
<input type=”button” name=”one” style=”width: 40px”
```

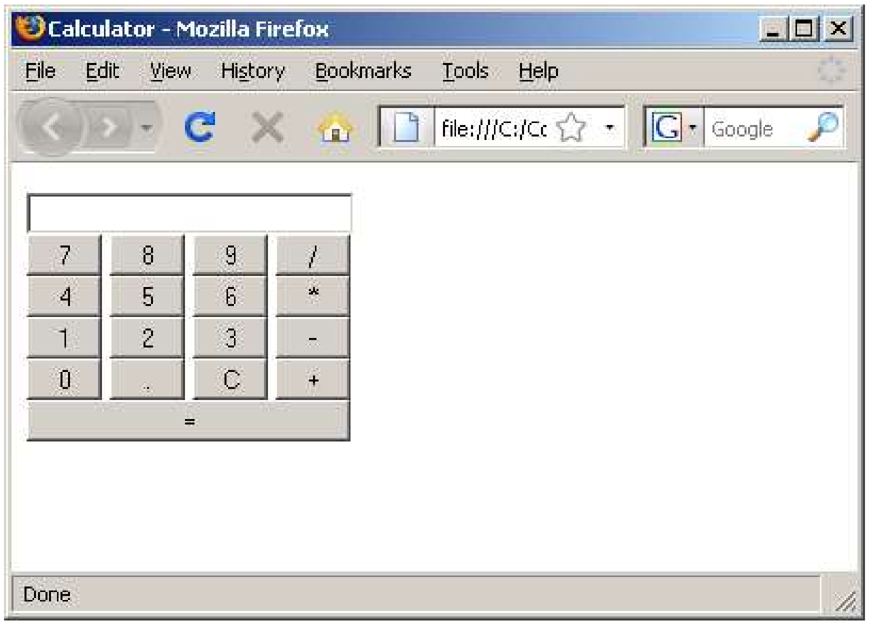
```
   value=”1” onclick=”updateString(‘1’)” />
<input type=”button” name=”two” style=”width: 40px”
   value=”2” onclick=”updateString(‘2’)” />
<input type=”button” name=”three”
style=”width: 40px” value=”3”
onclick=”updateString(‘3’)” />
<input type=”button” name=”minus”
style=”width: 40px” value=”-”
onclick=”updateString(‘-’)” /><br />
<input type=”button” name=”zero”
style=”width: 40px” value=”0”
onclick=”updateString(‘0’)” />
<input type=”button” name=”point”
style=”width: 40px” value=”.”
onclick=”updateString(‘.’)” />
<input type=”button” name=”clear”
style=”width: 40px” value=”C”
onclick=”document.forms[0].input.value=’’; ↵
   inputString=”” />
<input type=”button” name=”plus” style=”width: 40px”
   value=”+” onclick=”updateString(‘+’)” /><br />
```

8. Finally, locate the end of the paragraph in the form. At the end of that paragraph, add the following element for the calc button (the one with the equal sign). Notice that the onclick event for the calc button performs the calculation by using the eval() function with the inputString variable. The calculated value is then assigned as the value of the input text box.

```
<input type=”button” name=”calc”
style=”width: 172px” value=”=”
onclick=”document.forms[0].input.value ↵
  =eval(inputString); inputString=’’” />
```

9. Save the document as **Calculator.html** in the Chapter folder for Chapter 9, and then validate it with the W3C Markup Validation Service at *validator.w3.org/file-upload.html* and fix any errors that the document contains. Once the document is valid, open the **Calculator.html** document in your Web browser and test the calculation's functionality. Figure 9-1 shows how the Calculator.html document looks in a Web browser.

Figure 9-1: Calculator.html document in a Web browser


Calculator - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/Cc
Google
7 8 9 /
4 5 6 *
1 2 3 -
0 . C +
=
Done

10. Close your Web browser window.

Next, you will modify the calculator so that it includes storage functionality using a hidden form field.

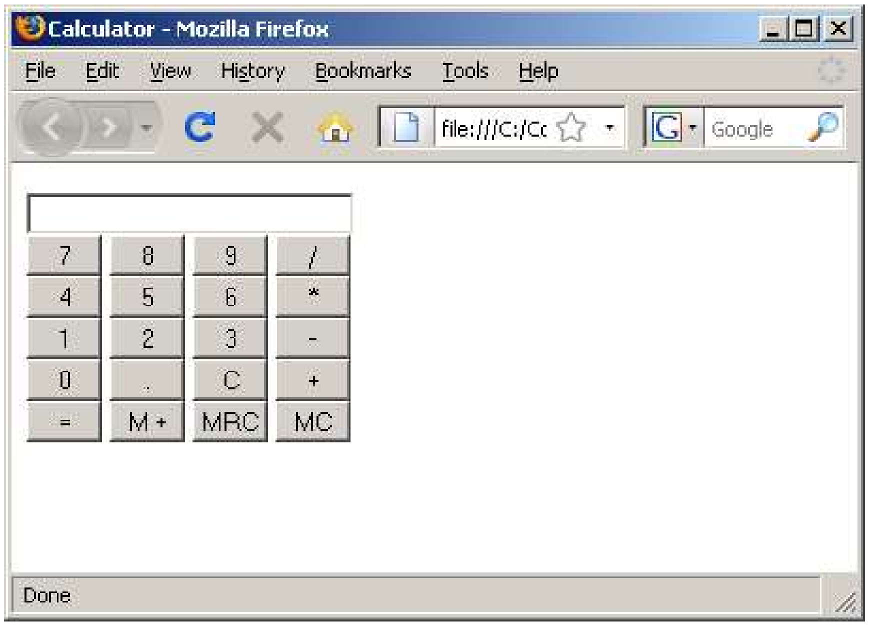
To add storage functionality to the calculator:

1. Return to the **Calculator.html** file in your text editor.
2. Locate the button that performs the calculation, and change the value assigned to the width style from 172px to **40px**.
3. Add the following elements above the closing </p> tag in the form. The first new button, named mem, adds the value of the input text box to the value stored in the hidden form field named storedValue. Notice that the mem button's onclick event handler uses two calls to the parseInt() function. Form text fields only store data in the form of text strings. For this reason, you must use the built-in parseInt() function to convert the contents of a text field to an integer. After this conversion, the contents of the text field can be used in a JavaScript calculation. If you do not use the parseInt() function in the mem button's onclick event handler, when you attempt to assign another number to the hidden storedValue field, the new number is concatenated with the contents of the storedValue field, just as when you combine two text fields. The second new button, named recall, retrieves the information stored in the hidden storedValue field and passes it to the updateString() function. The third new button, named memClear, clears the contents of the hidden storedValue field.

```
<input type=”button” name=”mem”
style=”width: 40px” value=”M +”
onclick=”document.forms[0].storedValue.value ↵
  = parseInt(document.forms[0].storedValue.value) ↵
  + parseInt(document.forms[0].input.value)” />
<input type=”button” name=”recall”
style=”width: 40px” value=”MRC”
onclick=”updateString(document.forms[0] ↵
  .storedValue.value)” />
<input type=”button” name=”memClear”
style=”width: 40px” value=”MC”
onclick=”document.forms[0].storedValue.value=0” />
<input type=”hidden” name=”storedValue” value=”0” />
```

4. Save the **Calculator.html** document and then validate it with the W3C Markup Validation Service at *validator.w3.org/file-upload.html*, and fix any errors that the document contains. Once the document is valid, open the **Calculator.html** document in your Web browser and test the storage functionality. Figure 9-2 shows how the Calculator.html document looks in a Web browser after adding the new buttons.

Figure 9-2: Calculator.html document in a Web browser after adding new buttons


Calculator - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/Cc
Google
7 8 9 /
4 5 6 *
1 2 3 -
0 . C +
= M + MRC MC
Done

5. Close your Web browser window and the **Calculator.html** file in your text editor.

Using Hidden Form Fields with the Printer Product Registration Page

For learning state preservation techniques, the calculator script is limited because you can only save the most recently calculated value in a hidden form field. To learn more advanced state information techniques, you will turn your attention to a frame-based Printer Product Registration Web page. The Printer Product Registration Web page consists of two Web pages: the first page contains a form for recording customer information, and the second page contains a form for recording product information. The documents are already created for you; you can find them in your Chapter folder for Chapter 9. Figure 9-3 shows the Customer Information form and Figure 9-4 shows the Product Information form of the Printer Product Registration page.

Figure 9-3: Customer Information form

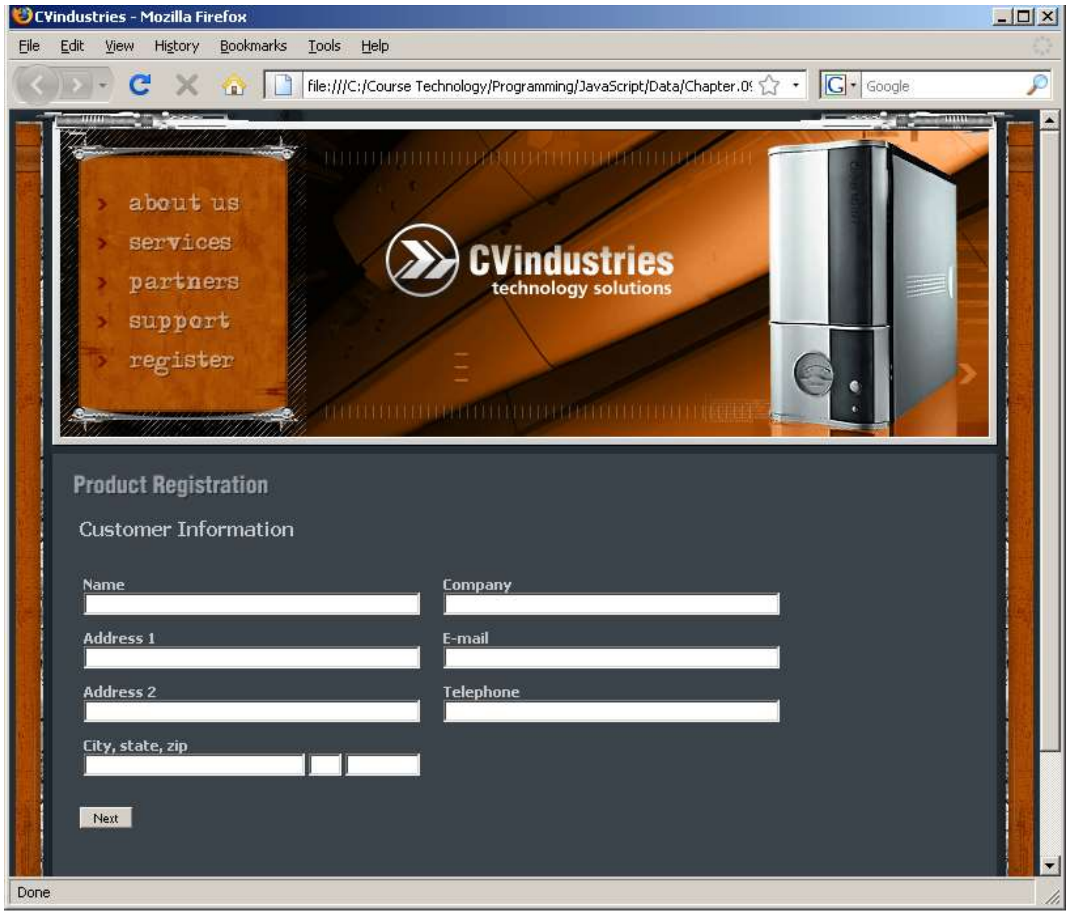
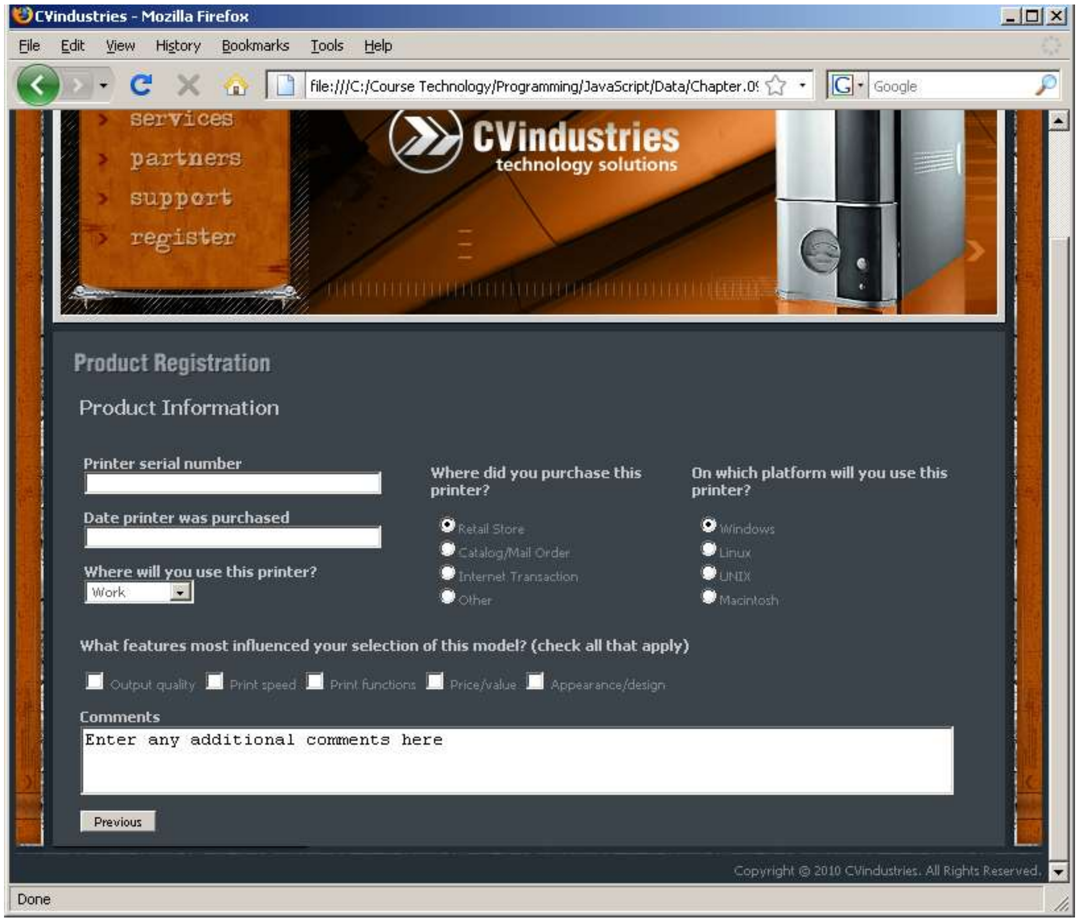


Figure 9-4: Product Information form of the Printer Product Registration page


CVindustries - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/Course Technology/Programming/JavaScript/Data/Chapter.0
Google
services
partners
support
register
CVindustries
technology solutions
Done

Product Registration

Product Information

Printer serial number

Date printer was purchased

Where will you use this printer?
Work

Where did you purchase this printer?
[✓ Retail Store] Catalog/Mail Order Internet Transaction Other

On which platform will you use this printer?
[✓ Windows] Linux UNIX Macintosh

What features most influenced your selection of this model? (check all that apply)
☐ Output quality ☐ Print speed ☐ Print functions ☐ Price/value ☐ Appearance/design

Comments
Enter any additional comments here

Previous

Copyright © 2010 CVindustries. All Rights Reserved.

The forms are designed so that data entered by the user on both forms can be submitted to a Web server simultaneously. This makes sense because the data collected by both forms are really part of the same data set; the forms are broken into two Web pages only to make it easier for the user to enter the necessary information. The problem with these Web pages is that, if a user moves from the Customer Information page to the Product Information page, the data entered on the Customer Information page is lost. In this chapter, you will learn how to save the values entered into the two Web pages by using hidden form fields, query strings, and cookies.

The frameset Web page, which is named ProductRegistration.html, creates a top frame and a bottom frame. The top frame is not visible because it is assigned a height of 0px and because the border attribute is also assigned a value of 0px. The invisible top frame will be used to maintain state information with hidden form fields, which you will study first.

Next, you add hidden form fields to the Printer Product Registration script. These fields will store customer information when the user moves from the Customer Information form to the Product Information form. The Web pages containing the forms are displayed in the bottom frame of a frame-based Web page. The Product Information form is displayed when a user clicks the Next button at the bottom of the Customer Information form. The problem is that once you click the Next button to move to the Product Information page (when you click the Previous button to move from the Product Information page back to the Customer Information page), the form values are lost. To fix this problem, you will add hidden form fields to the hidden top frame. When you click the Next or Previous buttons, the values in the forms will be copied into the hidden form fields in the top frame. You will also add a Submit button to the Product Information form that will not, in fact, submit the Product Information form to a Web server. Instead, the Submit button will copy the values of the Product Information form's fields into the hidden form fields in the top frame. Then, the form within the top frame will be submitted to a document named FormProcessor.html, using the Form object's `submit()` method. First, you will add the hidden form fields to the top frame.

To add the hidden form fields to the top frame of the Printer Product Registration frameset:

1. Start your text editor and open the **TopFrame.html** document from the CVindustries_hiddenfields folder in your Chapter folder for Chapter 9.
2. Add the following form and hidden form fields above the closing `</body>` tag. The form contains hidden form fields that will store values from both the Customer Information form and the Product Information form. Notice that the form will be submitted to the FormProcessor.html document.

```
<form action=”FormProcessor.html” method=”get”
enctype=”application/x-www-form-urlencoded”>
<p><input type=”hidden” name=”name” />
<input type=”hidden” name=”address1” />
<input type=”hidden” name=”address2” />
<input type=”hidden” name=”city” />
```

```
<input type=”hidden” name=”state” />
<input type=”hidden” name=”zip” />
<input type=”hidden” name=”company” />
<input type=”hidden” name=”email” />
<input type=”hidden” name=”telephone” />
<input type=”hidden” name=”serial” />
<input type=”hidden” name=”date” />
<input type=”hidden” name=”usedWhere” />
<input type=”hidden” name=”purchasedWhere” />
<input type=”hidden” name=”platform” />
<input type=”hidden” name=”quality” />
<input type=”hidden” name=”speed” />
<input type=”hidden” name=”functions” />
<input type=”hidden” name=”price” />
<input type=”hidden” name=”design” />
<input type=”hidden” name=”comments” /></p>
</form>
```

3. Save the **TopFrame.html** document, and then close it in your text editor.

Next, you add code to the Customer Information and Product Information documents that copies the form field values to the hidden form fields in the top frame of the Printer Product Registration frameset. For the Customer Information document, the values will be copied when you click the Next button, and for the Product Information document, the values will be copied when you click the Previous button.

To copy the form field values in the Customer Information and Product Information documents to the hidden form fields in the top frame of the Printer Product Registration frameset:

1. Open the **CustomerInfo.html** document from the -CVindustries_hiddenfields folder in your Chapter folder for Chapter 9 in your text editor.
2. The form in the CustomerInfo.html document includes a Next button with an onclick event handler that calls a function named nextForm(). The nextForm() function contains a single statement that opens the Product Information document using the href property of the Location object. Add the following statements above the single statement in the nextForm() function. The statements use the parent property to copy the values of the Customer Information form to the corresponding hidden form fields in the top frame.

```
parent.topframe.document.forms[0].name.value =
    document.forms[0].name.value;
parent.topframe.document.forms[0].address1.value =
    document.forms[0].address1.value;
parent.topframe.document.forms[0].address2.value =
    document.forms[0].address2.value;
parent.topframe.document.forms[0].city.value =
    document.forms[0].city.value;
parent.topframe.document.forms[0].state.value =
    document.forms[0].state.value;
parent.topframe.document.forms[0].zip.value =
    document.forms[0].zip.value;
parent.topframe.document.forms[0].company.value =
    document.forms[0].company.value;
```

```
parent.topframe.document.forms[0].email.value =
    document.forms[0].email.value;
parent.topframe.document.forms[0].telephone.value =
    document.forms[0].telephone.value;
```

3. Save the **CustomerInfo.html** document.
4. Open the **ProductInfo.html** document from the -CVindustries_hiddenfields folder in your Chapter folder for Chapter 9 in your text editor.
5. Add the following new function, named `saveProductData()`, to the end of the script section. The statements use the `parent` property to copy the values of the Product Information form to the corresponding hidden form fields in the top frame.

```
function saveProductData() {
   parent.topframe.document.forms[0].serial.value
      = document.forms[0].serial.value;
   parent.topframe.document.forms[0].date.value
      = document.forms[0].date.value;
   for (var i = 0; i < document.forms[0]
      .useLocation.length; ++i) {
      if (document.forms[0].useLocation
         .options[i].selected == true) {
         parent.topframe.document
            .forms[0].usedWhere.value
            = document.forms[0].useLocation
            .options[i].value;
         break;
```

```
    }
}
for (var j = 0; j < 4; ++j) {
    if (document.forms[0]
        .purchaseLocation[j].checked == true) {
        parent.topframe.document
            .forms[0].purchasedWhere.value
            = document.forms[0]
            .purchaseLocation[j].value;
        break;
    }
}
for (var k = 0; k < 4; ++k) {
    if (document.forms[0].platform[k]
        .checked == true) {
        parent.topframe.document
            .forms[0].platform.value
            = document.forms[0]
            .platform[k].value;
        break;
    }
}
```

```
    if (document.forms[0].quality.checked == true)
       parent.topframe.document
          .forms[0].quality.value = “true”;
    if (document.forms[0].speed.checked == true)
       parent.topframe.document
          .forms[0].speed.value = “true”;
    if (document.forms[0].functions.checked == true)
       parent.topframe.document
          .forms[0].functions.value = “true”;
    if (document.forms[0].price.checked == true)
       parent.topframe.document
          .forms[0].price.value = “true”;
    if (document.forms[0].design.checked == true)
       parent.topframe.document
          .forms[0].design.value = “true”;
    parent.topframe.document.forms[0].comments.value
       = document.forms[0].comments.value;
}
```

6. The form in the ProductInfo.html document includes a Previous button with an onclick event handler that calls a function named previousForm(). The previousForm() function contains a single statement that opens the Customer Information document using the href property of the Location object. To call the saveProductData() function when you click the Previous button, add the following statement above the single statement in the previousForm() function.

```
function previousForm() {
   saveProductData();
   location.href = “CustomerInfo.html”;
}
```

7. Save the **ProductInfo.html** document.

Next, you will add functions to the CustomerInfo.html and ProductInfo.html files that populate the form fields when you navigate between the pages. The functions will be called with an onload event handler in the opening <body> tag.

To add code that populates the CustomerInfo.html and ProductInfo.html files:

1. Return to the **CustomerInfo.html** document in your text editor.
2. Add the following populateCustomerInfo() function to the end of the script section. The statements in the function simply copy the values from the form in the top frameset to the corresponding fields in the bottom frameset.

```
function populateCustomerInfo() {
   document.forms[0].name.value
      = parent.topframe.document
      .forms[0].name.value;
   document.forms[0].address1.value
      = parent.topframe.document
      .forms[0].address1.value;
   document.forms[0].address2.value
      = parent.topframe.document
      .forms[0].address2.value;
   document.forms[0].city.value
```

```
        = parent.topframe.document
        .forms[0].city.value;
    document.forms[0].state.value
        = parent.topframe.document
        .forms[0].state.value;
    document.forms[0].zip.value
        = parent.topframe.document
        .forms[0].zip.value;
    document.forms[0].company.value
        = parent.topframe.document
        .forms[0].company.value;
    document.forms[0].email.value
        = parent.topframe.document
        .forms[0].email.value;
    document.forms[0].telephone.value
        = parent.topframe.document
        .forms[0].telephone.value;
}
```

3. Add an onload event handler to the opening <body> tag to call the populateCustomerInfo() function, as follows:

   ```
   <body onload=”populateCustomerInfo()”>
   ```

4. Save the **CustomerInfo.html** document, and then close it in your text editor.
5. Return to the **ProductInfo.html** document in your text editor.

6. Add the following populateProductData()function to the end of the script section. The statements in the function are a little more complicated than the ones found in the populateCustomerInfo() function. These new functions need to evaluate the values found in the fields in the top frame in order to select the correct values in the select list and radio buttons lists in the bottom frame.

```
function populateProductData() {
   document.forms[0].serial.value
      = parent.topframe.document
      .forms[0].serial.value;
   document.forms[0].date.value
      = parent.topframe.document
      .forms[0].date.value;
   if (parent.topframe.document
      .forms[0].usedWhere.value == “work”)
      document.forms[0].useLocation
         .options[0].selected = true;
   else if (parent.topframe.document
      .forms[0].usedWhere.value == “school”)
      document.forms[0].useLocation
         .options[1].selected = true;
   else if (parent.topframe.document
      .forms[0].usedWhere.value == “home”)
      document.forms[0].useLocation
         .options[2].selected = true;
```

```
else if (parent.topframe.document
   .forms[0].usedWhere.value == “home_office”)
   document.forms[0].useLocation.
      options[3].selected = true;
if (parent.topframe.document
   .forms[0].purchasedWhere.value == “retail”)
   document.forms[0].purchaseLocation[0]
      .checked = true;
else if (parent.topframe.document
   .forms[0].purchasedWhere.value
   == “catalog_mail”)
   document.forms[0].purchaseLocation[1]
      .checked = true;
else if (parent.topframe.document
   .forms[0].purchasedWhere.value
   == “internet”)
   document.forms[0].purchaseLocation[2]
      .checked = true;
else if (parent.topframe.document.forms[0]
   .purchasedWhere.value == “other”)
   document.forms[0].purchaseLocation[3]
      .checked = true;
```

```
if (parent.topframe.document
   .forms[0].platform.value == “windows”)
   document.forms[0].platform[0].checked
      = true;
else if (parent.topframe.document
   .forms[0].platform.value == “linux”)
   document.forms[0].platform[1].checked
      = true;
else if (parent.topframe.document
   .forms[0].platform.value == “unix”)
   document.forms[0].platform[2].checked
      = true;
else if (parent.topframe.document
   .forms[0].platform.value == “mac”)
   document.forms[0].platform[3].checked
       = true;
if (parent.topframe.document
   .forms[0].quality.value == “true”)
   document.forms[0].quality.checked = true;
if (parent.topframe.document
   .forms[0].speed.value == “true”)
   document.forms[0].speed.checked = true;
```

```
    if (parent.topframe.document
       .forms[0].functions.value == “true”)
       document.forms[0].functions.checked = true;
    if (parent.topframe.document
       .forms[0].price.value == “true”)
       document.forms[0].price.checked = true;
    if (parent.topframe.document
       .forms[0].design.value == “true”)
       document.forms[0].design.checked = true;
    document.forms[0].comments.value
       = parent.topframe.document
       .forms[0].comments.value;
}
```

7. Add an onload event handler to the opening <body> tag to call the populateProductData() function, as follows:

   ```
   <body onload=”populateProductData()”>
   ```

8. Save the **ProductInfo.html** document, and then open the **ProductRegistration.html** file in your Web browser. Enter some data into the customer information form, click the **Next** button, and then enter some data into the product information form. Test the Previous and Next buttons to ensure that the data is still visible as you navigate between the two pages.
9. Close your Web browser window.

Next, you add code to the Product Information document that submits the Printer Product Registration to the FormProcessor.html document.

To add code to the Product Information document that submits the Printer Product Registration to the FormProcessor.html document:

1. Return to the **ProductInfo.html** document in your text editor.
2. Add a submit button to the end of the form, immediately after the Previous button, as follows:

   ```
   <p><input type="button" name="previous" value="
   Previous " onclick="previousForm()" />
   <input type="submit" value="Register Product" /></p>
   ```

3. Add to the opening <form> tag the following onsubmit event handler, which calls a function named submitForm():

   ```
   onsubmit="return submitForm()"
   ```

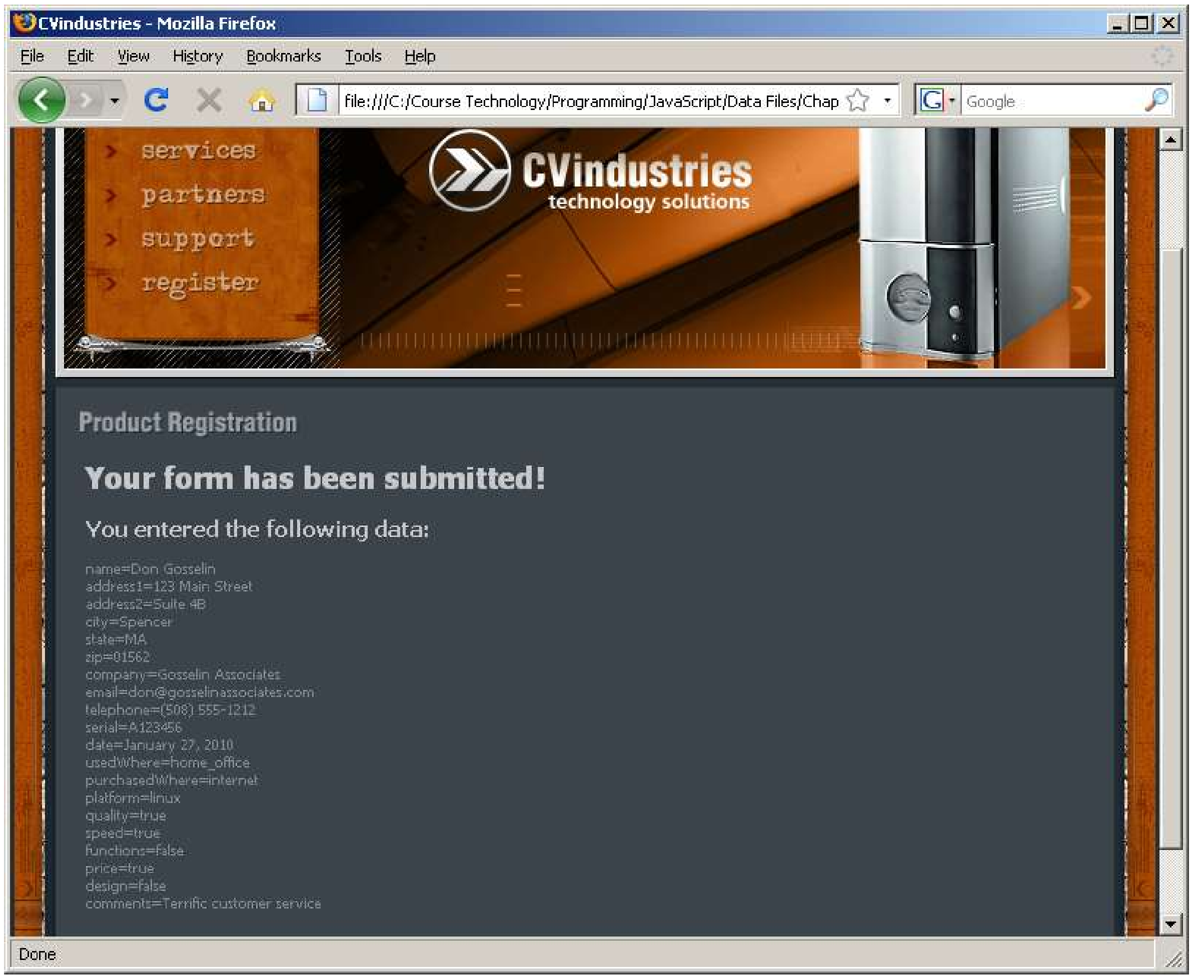
4. Now add the following submitForm() function to the end of the script section in the document head. The first statement calls the saveProductData() function to copy the values in the Product Information form to the corresponding hidden form fields in the top frame. The second statement uses the Form object's submit() function to submit the form in the top frame to the FormProcessor.html document. (The action attribute in the <form> element in the top frame is assigned "FormProcessor.html", which submits the top frame's form to the FormProcessor.html document.) Notice that the last statement returns a value of false, which prevents the form in the ProductIn-fo.html document from submitting its data.

   ```
   function submitForm() {
      saveProductData();
      parent.topframe.document.forms[0].submit();
      return false;
   }
   ```

5. Save the **ProductInfo.html** document, and then open the **ProductRegistration.html** document in a Web browser. Enter some data into the Customer Information form

fields, and click the **Next** button. Then, enter some data into the Product Information form fields, and click the **Register Product** button. The FormProcessor.html document should open and display the data you entered, as shown in Figure 9-5.

Figure 9-5: FormProcessor.html document


CVindustries - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/Course Technology/Programming/JavaScript/Data Files/Chap
Google
services
partners
support
register
CVindustries
technology solutions
Done

Product Registration

Your form has been submitted!

You entered the following data:

```
name=Don Gosselin
address1=123 Main Street
address2=Suite 4B
city=Spencer
state=MA
zip=01562
company=Gosselin Associates
email=don@gosselinassociates.com
telephone=(508) 555-1212
serial=A123456
date=January 27, 2010
usedWhere=home_office
purchasedWhere=internet
platform=linux
quality=true
speed=true
functions=false
price=true
design=false
comments=Terrific customer service
```

6. Close your Web browser window and the ProductInfo.html file in your text editor.

Saving State Information with Query Strings

One way to preserve information following a user's visit to a Web page is to append a query string to the end of a URL. A **query string** is a set of name=value pairs appended to a target URL. It consists of a single text string containing one or more pieces of information. You can use a query string to pass information, such as search criteria, from one Web page to another.

Passing Data with a Query String

To pass data from one Web page to another using a query string, add a question mark (?) immediately after a URL, followed by the query string (in name=value pairs) for the information you want to preserve. In this manner, you are passing information to another Web page, similarly to the way you can pass arguments to a function or method. You separate individual name=value pairs within the query string by using ampersands (&). The following code provides an example of an <a> element that contains a query string consisting of three name=value pairs:

```
<a href="http://www.URL.com/TargetPage.html? ↵
firstName=Don&lastName=Gosselin&occupation=writer">
Link Text</a>
```

The passed query string is then assigned to the search property of the target Web page Location object. The search property of the Location object contains a URL's query or search parameters. For the preceding example, after the TargetPage.html document opens, the query string "?firstName=Don&lastName=Gosselin&occupation=writer" is available as the value of the search property of the Location object.

Next, you will begin to modify the Printer Product Registration pages so that the registration information is passed as query strings instead of being stored in hidden form fields.

To begin modifying the Printer Product Registration pages so that the registration information is passed as query strings:

1. First, copy the CVindustries_hiddenfields folder to a folder named **CVindustries_querystrings**.
2. Open the **CustomerInfo.html** document in the -CVindustries_querystrings folder in your text editor.
3. Replace all of the statements in the nextForm() function with the following code, which builds the query string using each form element name and value in a variable named savedData. Notice that the if statement checks to see if the savedData variable, which is assigned the search string, contains a value. If it does, that means the page was opened from the ProductInfo.html page and contains product information

fields. If the variable does contain product information, then the query string fields for the ProductInfo.html page are extracted to the productData variable by searching for "serial", which is the first field on the Product Information form, and then by using the substring() method to retrieve all of the data to the end of the string. The query string is then built for the customer information fields and assigned to the savedData variable. The name of each form element is entered as a literal string and concatenated with the value property of each element, using the + and += assignment operators. Then, the product information productData variable is appended to the savedData variable. The last statement appends the savedData query string to the ProductInfo.html URL that is assigned to the href property of the Location object.

```
var savedData = location.search;
var productData = "";
if (savedData != "")
   productData = savedData.substring(
      savedData.search("&serial"), savedData.length);
savedData = "?name=" + document.forms[0].name.value;
savedData += "&address1=" +
document.forms[0].address1.value;
savedData += "&address2=" +
document.forms[0].address2.value;
savedData += "&city=" +
document.forms[0].city.value;
savedData += "&state=" +
document.forms[0].state.value;
savedData += "&zip=" +
document.forms[0].zip.value;
savedData += “&company=” +
document.forms[0].company.value;
savedData += “&email=” +
document.forms[0].email.value;
savedData += “&telephone=” +
document.forms[0].telephone.value;
savedData += productData;
location.href = “ProductInfo.html” +
savedData;
```

4. Save the **CustomerInfo.html** document.
5. Open the **ProductInfo.html** document in the CVindustries_querystrings folder in your text editor.
6. Replace the statements in the saveProductData() function with the following code, which builds and returns a query string. The first if statement uses the substring() method to return only the customer information fields in the query string by extracting the characters up to a value of “serial”, which is the first field in the product information form. Although they appear complicated, the remaining statements are very similar to the statements in the previous version of the function that copied the values of the Product Information form to the corresponding hidden form fields in the top frame. In this version, they instead copy the values to the savedData variable.

```
var savedData = location.search;
if (savedData.search(“serial”) != -1)
   savedData = savedData.substring(0,
   savedData.search(“serial”));
```

```
}

savedData += “&serial=” +
document.forms[0].serial.value;
savedData += “&date=” +
document.forms[0].date.value;
for (var i = 0; i <
document.forms[0].useLocation.length; ++i) {
   if (document.forms[0].useLocation.options[i]
      .selected == true) {
      savedData += “&useLocation=”
         + document.forms[0].useLocation
         .options[i].value;
      break;
   }
}
for (var j = 0; j < 4; ++j) {
   if (document.forms[0].purchaseLocation[j]
      .checked == true) {
      savedData += “&purchaseLocation=”
         + document.forms[0].purchaseLocation[j]
         .value;
      break;
   }
```

```
}
for (var k = 0; k < 4; ++k) {
   if (document.forms[0].platform[k]
      .checked == true) {
      savedData += “&platform=”
         + document.forms[0].platform[k].value;
      break;
   }
}
if (document.forms[0].quality.checked == true)
   savedData += “&quality=true”;
else
   savedData += “&quality=false”;
if (document.forms[0].speed.checked == true)
   savedData += “&speed=true”;
else
   savedData += “&speed=false”;
if (document.forms[0].functions.checked == true)
   savedData += “&functions=true”;
else
   savedData += “&functions=false”;
if (document.forms[0].price.checked == true)
```

```
   savedData += “&price=true”;
else
   savedData += “&price=false”;
if (document.forms[0].design.checked == true)
   savedData += “&design=true”;
else
   savedData += “&design=false”;
savedData += “&comments=”
   + document.forms[0].comments.value;
return savedData;
```

7. Modify the previousForm() function so that the first statement assigns the query value returned from the saveProductData() function to a variable named queryString. Then, append the queryString variable to the CustomerInfo.html file and assign the combined value to the location.href property. The modified previousForm() function should appear as follows:

```
function previousForm() {
   var queryString = saveProductData();
   location.href = “CustomerInfo.html” + queryString;
}
```

8. Modify the submitForm() function as follows. The first statement assigns the query value returned from the saveProductData() function to a variable named savedData. The second statement appends the savedData variable to the FormProcessor.html file and assigns it to the top.location.href property.

```
function submitForm() {
   var savedData = saveProductData();
```

```
    top.location.href = "FormProcessor.html"
        + savedData;
    return false;
}
```

9. Save the **ProductInfo.html** document.

Before you can test the new code, you need learn how to manipulate query strings.

Parsing Data from a Query String

For a Web page to use the information in a query string, your JavaScript program must first parse the string, using a combination of several methods and the length property of the String object. (This is also true when you want to use data contained in a cookie, as you'll learn later in this chapter.) The first parsing task is to remove the question mark at the start of the query string, using the substring() method combined with the length property. As you recall from Chapter 7, the substring() method takes two arguments: a starting index number and an ending index number. The first character in a string has an index number of 0, similar to the first element in an array. Because you want to exclude the first character of the string (the question mark), which has an index of 0, you use a starting index of 1. For the ending index number you use the length property, which tells the substring() method to include the rest, or length, of the string. The following code assigns the search property of the Location object to a variable named queryData and uses the substring() method and length property to remove the starting question mark:

```
// Assigns the query string to the queryData variable
var queryData = location.search;
// Removes the opening question mark from the string
queryData = queryData.substring(1,
   queryData.length);
```

The next step is to convert the individual pieces of information in the queryData variable into array elements, using the split() method. You pass to the split() method the character that separates each individual piece of information in a string. In this case, you will pass the ampersand character, because that is the character that separates the name=value pairs in the query string. However, keep in mind that you can split a string at any character. The code to convert the information in the queryData variable into an array named queryArray[] is as follows:

```
// splits queryData into an array
var queryArray = queryData.split("&");
```

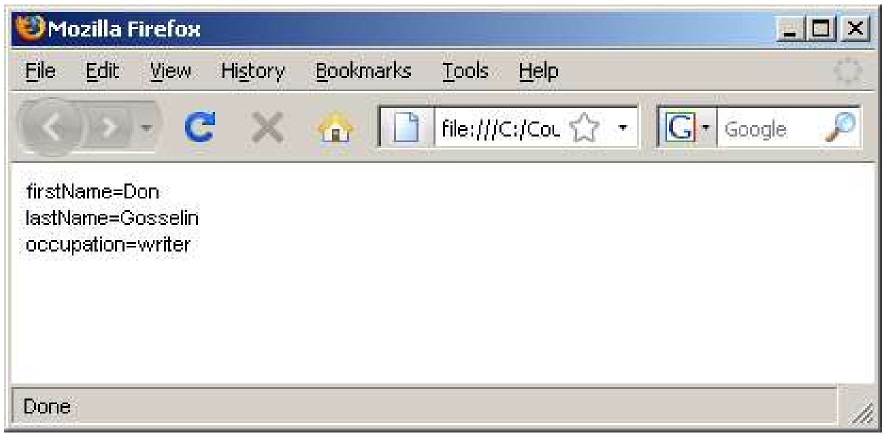
The following code shows a completed version of the parsing script that uses a for loop to print the values in queryArray[]:

```
// Assigns the query string to queryData
var queryData = location.search;
// Removes the opening question mark from the string
queryData = queryData.substring(1, queryData.length);
// splits queryData into an array
var queryArray = queryData.split("&");
for (var i=0; i<queryArray.length; ++i) {
    document.write(queryArray[i] + "<br />");
}
```

Figure 9-6 shows the output in a Web browser when the location.search property in the preceding code contains the -following string value:

```
?firstName=Don&lastName=Gosselin&occupation=writer
```

Figure 9-6: Parsing script in a Web browser



```
firstName=Don
lastName=Gosselin
occupation=writer
```

Next, you will modify the populateCustomerInfo() and populateProductData() functions so they extract and display the data in the query strings that are passed between the CustomerInfo.html and ProductInfo.html files.

To modify the populateCustomerInfo() and populateProductData() functions so that they extract and display the data in the query strings that are passed between the CustomerInfo.html and ProductInfo.html files:

1. Return to the CustomerInfo.html file in your text editor.
2. Modify the populateCustomerInfo() function as follows. The first few statements retrieve the value assigned to the location.search property and then split the data into an array named queryArray[]. Each subsequent statement then uses the substring() method of the String object to extract and display the value portion of each name=value pair, based on its index in the array. For example, the name value is located at queryArray[0], while the telephone value is located at queryArray[8].

```
function populateCustomerInfo() {
    if (location.search) {
        var queryData = location.search;
        queryData = queryData.substring(1,
            queryData.length);
```

```
var queryArray = queryData.split("&");
document.forms[0].name.value = queryArray[0]
   .substring(queryArray[0]
   .lastIndexOf("=") + 1);
document.forms[0].address1.value
   = queryArray[1].substring(queryArray[1]
   .lastIndexOf("=") + 1);
document.forms[0].address2.value
   = queryArray[2].substring(queryArray[2]
   .lastIndexOf("=") + 1);
document.forms[0].city.value = queryArray[3]
   .substring(queryArray[3]
   .lastIndexOf("=") + 1);
document.forms[0].state.value
    = queryArray[4].substring(queryArray[4]
   .lastIndexOf("=") + 1);
document.forms[0].zip.value = queryArray[5]
   .substring(queryArray[5]
   .lastIndexOf("=") + 1);
document.forms[0].company.value
    = queryArray[6].substring(queryArray[6]
   .lastIndexOf("=") + 1);
```

```
        document.forms[0].email.value
            = queryArray[7].substring(queryArray[7]
            .lastIndexOf("=") + 1);
        document.forms[0].telephone.value
            = queryArray[8].substring(queryArray[8]
            .lastIndexOf("=") + 1);
    }
}
```

3. Save the **CustomerInfo.html** file, and then close it in your text editor.
4. Return to the **ProductInfo.html** file in your text editor.
5. Modify the populateProductData() function as follows. The first statement retrieves the value of the location.search property and assigns it to the queryData variable. The if statement then uses the search() method of the String object to search the queryData variable for "serial". If the method returns a value other than -1, then data for the product information page is stored in the query string. In that case, the statements within the if statement use the substring() method of the String object to extract and display the value portion of each name=value pair, based on its index in the array. For example, the serial value is located at queryArray[9] and the date value is located at queryArray[10].

```
function populateProductData() {
    var queryData = location.search;
    if (queryData.search("serial") != -1) {
        queryData = queryData.substring(1,
          queryData.length);
        var queryArray = queryData.split("&");
```

```
document.forms[0].serial.value
   = queryArray[9].substring(
   queryArray[9].lastIndexOf(“=”) + 1);
document.forms[0].date.value
   = queryArray[10].substring(
   queryArray[10].lastIndexOf(“=”) + 1);
if (queryArray[11].substring(
   queryArray[11].lastIndexOf(“=”) + 1)
   == “work”)
   document.forms[0].useLocation.options[0]
      .selected = true;
else if (queryArray[11].substring(
   queryArray[11].lastIndexOf(“=”) + 1)
   == “school”)
   document.forms[0].useLocation.options[1]
      .selected = true;
else if (queryArray[11].substring(
   queryArray[11].lastIndexOf(“=”) + 1)
    == “home”)
   document.forms[0].useLocation.options[2]
      .selected = true;
else if (queryArray[11].substring(
   queryArray[10].lastIndexOf(“=”) + 1)
   == “home_office”)
   document.forms[0].useLocation.options[3]
      .selected = true;
if (queryArray[12].substring(
   queryArray[12].lastIndexOf(“=”) + 1)
   == “retail”)
   document.forms[0].purchaseLocation[0]
      .checked = true;
else if (queryArray[12].substring(
   queryArray[12].lastIndexOf(“=”) + 1)
    == “catalog_mail”)
   document.forms[0].purchaseLocation[1]
      .checked = true;
else if (queryArray[12].substring(
   queryArray[12].lastIndexOf(“=”) + 1)
   == “internet”)
   document.forms[0].purchaseLocation[2]
      .checked = true;
else if (queryArray[12].substring(
   queryArray[12].lastIndexOf(“=”) + 1)
   == “other”)
   document.forms[0].purchaseLocation[3]
      .checked = true;
if (queryArray[13].substring(
   queryArray[13].lastIndexOf("=") + 1)
   == "windows")
   document.forms[0].platform[0].checked
      = true;
else if (queryArray[13].substring(
   queryArray[13].lastIndexOf("=") + 1)
   == "linux")
   document.forms[0].platform[1].checked
      = true;
else if (queryArray[13].substring(
   queryArray[13].lastIndexOf("=") + 1)
   == "unix")
   document.forms[0].platform[2].checked
      = true;
else if (queryArray[13].substring(
   queryArray[13].lastIndexOf("=") + 1)
   == "mac")
   document.forms[0].platform[3].checked
      = true;
```

```
if (queryArray[14].substring(
   queryArray[14].lastIndexOf("=") + 1)
   == "true")
   document.forms[0].quality.checked
      = true;
if (queryArray[15].substring(
   queryArray[15].lastIndexOf("=") + 1)
   == "true")
   document.forms[0].speed.checked = true;
if (queryArray[16].substring(
   queryArray[16].lastIndexOf("=") + 1)
   == "true")
   document.forms[0].functions.checked
      = true;
if (queryArray[17].substring(
   queryArray[17].lastIndexOf("=") + 1)
   == "true")
   document.forms[0].price.checked = true;
if (queryArray[18].substring(
   queryArray[18].lastIndexOf("=") + 1)
   == "true")
   document.forms[0].design.checked = true;
```

```
        document.forms[0].comments.value
           = queryArray[19].substring(
        queryArray[19].lastIndexOf("=") + 1);
     }
  }
```

6. Save the **ProductInfo.html** document, and then open the **ProductRegistration.html** document in a Web browser. Enter some data into the Customer Information form fields, and click the **Next** button. Next, enter some data into the Product Information form fields, and click the **Register Product** button. The FormProcessor.html document should open and display the data you entered, just as it did with the hidden forms version of the script.
7. Close your Web browser window and the **ProductInfo.html** file in your text editor.

Short Quiz 1

1. Explain how to use hidden form fields to maintain state information.
2. Explain how to pass state information with query strings.
3. Explain how to parse query strings.

Saving State Information with Cookies

Query strings do not permanently maintain state information. The information contained in a query string is available only during the current session of a Web page. Once a Web page that reads a query string closes, the query string is lost. Hidden form fields maintain state information between Web pages, but the data they contain are also lost once the Web page that reads the hidden fields closes. You can save the contents of a query string or hidden form fields by submitting the form data by using a server-side scripting language, but that method requires a separate, server-based application. To make it possible to store state information beyond the current Web page session, Netscape created cookies. **Cookies** are small pieces of information about a user that are stored by a Web server in text files on the user's computer. The W3C DOM defines cookie specifications.

Each time the Web client visits a Web server, saved cookies for the requested Web page are sent from the client to the server. The server then uses the cookies to customize the Web page for the client. Cookies were originally created for use with CGI scripts but are now commonly used by JavaScript and other scripting languages.

You have probably seen cookies in action if you have ever visited a Web site where you entered a username in a prompt dialog box or in a text field and then found that you were greeted by that username the next time you visited the Web site. This could occur with each subsequent visit to the same Web site, whether during the same browser session or during a different browser session days or weeks later. The Web page remembers this information by storing it locally on your computer in a cookie. Another example of a cookie is a counter that counts the number of times an individual user has visited a Web site.

Cookies can be temporary or persistent. **Temporary cookies** remain available only for the current browser session. **Persistent cookies** remain available beyond the current browser session and are stored in a text file on a client computer. In this section, you will create both persistent and temporary cookies.

There are a number of limitations on the use of cookies that are enforced by Web browsers. Each individual server or domain can store only a maximum of 20 cookies on a user's computer. In addition, the total cookies per browser cannot exceed 300, and the largest cookie size is 4 KB. If these limits are exceeded, a Web browser may start discarding older cookies.

Creating and Modifying Cookies

You use the cookie property of the Document object to create cookies in name=value pairs, the same way you used name=value pairs with a query string. The syntax for the cookie property is as follows:

```
document.cookie = name + "=" + value;
```

The cookie property is created with a required name attribute and four optional attributes: expires, path, domain, and secure.

The name Attribute

The only required parameter of the cookie property is the name attribute, which specifies the cookie's name=value pair. Cookies created with only the name attribute are temporary cookies, because they are available for only the current browser session. The following code creates a cookie with a name=value pair of "firstName=Don":

```
document.cookie = "firstName=" + "Don";
```

The cookie property of the Document object can be confusing. For other JavaScript properties, assigning a new value to a property *replaces* the old value. In contrast, assigning a new value to the cookie property adds another entry to a list of cookies, rather than simply replacing the last value. The following example builds a list of cookies:

```
document.cookie = "firstName=" + "Don";

document.cookie = "lastName=" + "Gosselin";

document.cookie = "occupation=" + "writer";
```

A Web browser automatically separates each name=value pair in the cookie property with a semicolon and a space. Therefore, the value assigned to the cookie property for the preceding cookies contains the following value:

```
firstName=Don; lastName=Gosselin; occupation=writer
```

By default, cookies themselves cannot include semicolons or other special characters, such as commas or spaces. Cookies cannot include special characters because they are transmitted between Web browsers and Web servers using HTTP, which does not allow certain nonalphanumeric characters to be transmitted in their native format. However, you can use special characters in your cookies if you use **encoding**, which involves converting special characters in a text string to their corresponding hexadecimal ASCII value, preceded by a percent sign. For example, 20 is the hexadecimal ASCII equivalent of a space character, and 25 is the hexadecimal ASCII equivalent of a percent sign (%). In URL encoded format, each space character is represented by %20, and each percent sign is represented by %25. After encoding, the contents of the string "tip=A standard tip is 15%" would read as follows:

```
tip=A%20standard%20tip%20is%2015%25
```

The built-in **encodeURIComponent() function** is used in JavaScript for encoding the individual parts of a URI. More specifically, the encodeURIComponent() function converts special characters in the individual parts of a URI to their corresponding hexadecimal ASCII value, preceded by a percent sign. The syntax for the encodeURIComponent() function is encodeURIComponent(*text*);. The encodeURIComponent() function does not encode standard alphanumeric characters such as A, B, C, or 1, 2, 3, or any of the following special characters: - _ . ! ~ * ‘ ( ). It also does not encode the following characters, which have a special meaning in a URI: ; / ? : % @ & = + $,. For example, the / character is not encoded because it is used for designating a path on a file system. When you read a cookie or other text string encoded with the encodeURIComponent() function, you must first decode it with the **decodeURIComponent() function**. The syntax for the decodeURIComponent() function is decodeURIComponent(*text*);. The following code encodes several cookies with the encodeURIComponent() function and assigns them to the cookie property of the Document object:

```
document.cookie = “firstName=”
   + encodeURIComponent(“Don”);
document.cookie = “lastName=”
   + encodeURIComponent(“Gosselin”);
document.cookie = “occupation=”
   + encodeURIComponent(“writer”);
```

If you transmit a URI containing spaces from current Web browsers (including Firefox and Internet Explorer), the Web browser automatically encodes the spaces for you before transmitting the cookie. However, special characters, such as the percent sign, are not automatically encoded. This can cause problems with older browsers and Web servers that do not recognize certain special characters unless they are encoded. Additionally, older Web browsers do not automatically encode spaces in URIs. For these reasons, you should manually encode and decode cookies using the encodeURIComponent() and decodeURIComponent() functions if you anticipate that your scripts will run in older Web browsers.

Next, you will modify the Customer Information and Product Information forms so that the fields are saved in temporary cookies instead of in query strings.

To modify the Customer Information and Product Information forms so that the fields are saved in temporary cookies instead of in query strings:

1. Copy the CVindustries_querystrings folder to a folder named **CVindustries_cookies**.
2. Open the **CustomerInfo.html** document in the CVindustries_cookies folder in your text editor.
3. Delete the following statements from the nextForm() function:

```
var savedData = location.search;
var productData = "";
if (savedData != "")
   productData = savedData.substring(
      savedData.search("&serial"),
      savedData.length);
```

4. Next, in each of the lines that build the savedData variable, replace savedData with **document.cookie**. Remove the question mark from the statement that stores the name field, and change the += assignment operators to standard assignment operators (=). Also remove the ampersands (&) from the name portion of each name=value pair, and encode each of the values that are assigned as cookies using the encodeURIComponent() method. Finally, delete the savedData += productData; statement and the portions of the location.href statement that append the query string, so that it reads **location.href = "ProductInfo.html";**. The statements in the modified nextForm() function should appear as follows:

```
document.cookie = "name="
   + encodeURIComponent(
   document.forms[0].name.value);
document.cookie = "address1="
   + encodeURIComponent(
   document.forms[0].address1.value);
document.cookie = "address2="
```

```
    + encodeURIComponent(
    document.forms[0].address2.value);
document.cookie = “city=”
    + encodeURIComponent(
    document.forms[0].city.value);
document.cookie = “state=”
    + encodeURIComponent(
    document.forms[0].state.value);
document.cookie = “zip=”
    + encodeURIComponent(
    document.forms[0].zip.value);
document.cookie = “company=”
    + encodeURIComponent(
    document.forms[0].company.value);
document.cookie = “email=”
    + encodeURIComponent(
    document.forms[0].email.value);
document.cookie = “telephone=”
    + encodeURIComponent(
    document.forms[0].telephone.value);
location.href = “ProductInfo.html”;
```

5. Save the **CustomerInfo.html** document.

6. Open the **ProductInfo.html** document in the CVindustries_cookies folder in your text editor.
7. Delete the following statements from the saveProductData() function:

```
var savedData = location.search;
if (savedData.search("serial") != -1)
   savedData = savedData.substring(0,
      savedData.search("serial"));
```

8. Next, in each of the lines that build the savedData variable, replace savedData with **document.cookie** and change the += assignment operators to standard assignment operators (=). Also, remove the ampersands (&) from the name portion of each name=value pair and encode each of the values that are assigned as cookies using the encodeURIComponent() method. Finally, delete the return statement at the end of the function. The statements in the modified saveProductData() function should appear as follows:

```
document.cookie = "serial=" + encodeURIComponent(
   document.forms[0].serial.value);
document.cookie = "date="
   + encodeURIComponent(
   document.forms[0].date.value);
for (var i = 0; i < document.forms[0].useLocation
   .length; ++i) {
   if (document.forms[0].useLocation.options[i]
      .selected == true) {
      document.cookie = "useLocation="
         + encodeURIComponent(
    }

          document.forms[0].useLocation
             .options[i].value);
          break;
    }
}
for (var j = 0; j < 4; ++j) {
    if (document.forms[0].purchaseLocation[j]
        .checked == true) {
        document.cookie = “purchaseLocation=”
            + encodeURIComponent(
            document.forms[0].purchaseLocation[j].value);
        break;
    }
}
for (var k = 0; k < 4; ++k) {
    if (document.forms[0].platform[k].checked
        == true) {
        document.cookie = “platform=”
            + encodeURIComponent(
            document.forms[0].platform[k].value);
        break;
    }
```

```
}
if (document.forms[0].quality.checked == true)
   document.cookie = “quality=true”;
else
   document.cookie = “quality=false”;
if (document.forms[0].speed.checked == true)
   document.cookie = “speed=true”;
else
   document.cookie = “speed=false”;
if (document.forms[0].functions.checked == true)
   document.cookie = “functions=true”;
else
   document.cookie = “functions=false”;
if (document.forms[0].price.checked == true)
   document.cookie = “price=true”;
else
   document.cookie = “price=false”;
if (document.forms[0].design.checked == true)
   document.cookie = “design=true”;
else
   document.cookie = “design=false”;
document.cookie = “comments=” + encodeURIComponent(
    document.forms[0].comments.value);
```

9. Modify the previousForm() function so it no longer appends the query string to the href property of the Location object. The modified form should appear as follows:

```
function previousForm(){
    saveProductData();
    location.href = "CustomerInfo.html";
}
```

10. Save the **ProductInfo.html** document.

Before you can open the Printer Product Registration document, you need to learn how to read cookies, as explained later in this chapter. Before you learn how to read cookies, you will learn about other cookie parameters.

Setting Cookie Expiration Dates

For a cookie to persist beyond the current browser session, you must use the expires attribute of the cookie property. The **expires attribute** of the cookie property determines how long a cookie can remain on a client system before it is deleted. Cookies created without an expires attribute are available for only the current browser session. The syntax for assigning the expires attribute to the cookie property, along with an associated name=value pair, is expires=*date*. The name=value pair and the expires=*date* pair are separated by a semicolon. The date portion of the expires attribute must be a text string in Coordinated Universal Time (usually abbreviated as UTC) format, which looks like this:

```
Weekday Mon DD HH:MM:SS Time Zone YYYY
```

The following is an example of Coordinated Universal Time:

```
Mon Dec 27 14:15:18 PST 2010
```

You can manually type a string in UTC format, or you can create the string with the Date object, which automatically creates the string in UTC format. (You first learned about the Date object in Chapter 6.) To use a Date object with the expires attribute, you specify the amount of time you want a cookie to be valid by using a combination of the set and get methods of the Date object. The following statement declares a Date object named cookieDate, and then changes the date portion of the new object by using the setDate() and getDate() methods. Notice that you can nest Date object methods inside other Date object methods. In the example, the setDate() method sets the date portion of cookieDate by using the get-Date() method to retrieve the date, and adding seven to increase the date by one week. You might use a cookie that expires after one week (or less) to store data that needs to be maintained for a limited amount of time. For example, a travel agency may store data in a cookie that temporarily holds a travel reservation that expires after a week.

```
cookieDate.setDate(myDate.getDate() + 7);
```

After you create a Date object and specify the date you want the cookie to expire, you must use the toUTCString() method to convert the Date object to a string, formatting it in Coordinated Universal Time. The following code creates a new cookie and assigns an expiration date one year from now. Before the expires attribute is assigned to the cookie property, the Date object uses the toUTCString() method to convert the date to a string in Coordinated Universal Time.

```
var expiresDate = new Date();
expiresDate.setFullYear(expiresDate.getFullYear() + 1);
document.cookie = “firstName=” + encodeURIComponent(“Don”)
   + “; expires=” + expiresDate.toUTCString();
```

Deleting Cookies from your Browser

When developing a JavaScript program, you may accidentally create, but not delete, persistent cookies that your program does not need. Unused persistent cookies can sometimes interfere with the execution of a JavaScript cookie program. For this reason, it’s a good idea to delete your browser cookies periodically, especially while developing a JavaScript program that uses cookies. To delete cookies in Firefox, select the Tools menu and then select Clear Recent History. In the Clear Recent History dialog box, select Everything in the Time range to clear box (if necessary), click the Details button (if necessary), and then select the Cookies button. Be sure to deselect any items in the Details section that you do not want to clear, and then click the Clear Now button. To delete cookies in Internet Explorer, select Internet Options from the Tools menu, click the General tab of the Internet Options dialog box, and then click the Delete button. In the Delete Browsing History dialog box, select Cookies, along with any other items you want to delete, and then click Delete.

Configuring Availability of Cookies to Other Web Pages on the Server

The **path attribute** determines the availability of a cookie to other Web pages on a server. The path attribute is assigned to the cookie property, along with an associated name=value pair, using the syntax path=*path name*. By default, a cookie is available to all Web pages in the same directory. However, if you specify a path, then a cookie is available to all Web pages in the specified path as well as to all Web pages in all subdirectories in the specified path. For example, the following statement makes the cookie named firstName available to all Web pages located in the /marketing directory or any of its subdirectories:

```
document.cookie = “firstName=”
   + encodeURIComponent(“Don”
   + “;path=/marketing”);
```

To make a cookie available to all directories on a server, use a slash to indicate the root directory, as in the following example:

```
document.cookie = “firstName=”
   + encodeURIComponent(“Don” + “;path=/”);
```

When you are developing JavaScript programs that create cookies, your programs may not function correctly if the directory containing your Web page contains other programs that create cookies. Cookies from other programs that are stored in the same directory along with unused cookies you created during development can cause your JavaScript cookie program to run erratically. Therefore, you should always place JavaScript cookie programs in their own directory and use the path attribute to specify any subdirectories your program requires.

Sharing Cookies Across a Domain

Using the path attribute allows cookies to be shared across a server. Some Web sites, however, are very large and use a number of servers. The **domain attribute** is used for sharing cookies across multiple servers in the same domain. Note that you cannot share cookies outside of a domain. The domain attribute is assigned to the cookie property, along with an associated name=value pair, using the syntax domain=*domain name*. For example, if the Web server programming.gosselin.com needs to share cookies with the Web server writing.gosselin.com, the domain attribute for cookies set by programming.gosselin.com should be set to .gosselin.com. That way, cookies created by programming.gosselin.com are available to writing.gosselin.com and to all other servers in the domain gosselin.com.

The following code shows how to make a cookie at programming.gosselin.com available to all servers in the gosselin.com domain:

```
document.cookie = “firstName=”
   + encodeURIComponent(“Don”
   + “;domain=.gosselin.com”);
```

Securing Cookie Transmissions

Internet connections are not always considered safe for transmitting sensitive information. It is possible for unscrupulous people to steal personal information, such as credit card numbers, passwords, Social Security numbers, and other types of private information online. To protect private data transferred across the Internet, Netscape developed Secure Sockets Layer, or SSL, to encrypt data and transfer it across a secure connection. The URLs for Web sites that support SSL usually start with the HTTPS protocol instead of HTTP. The **secure attribute** indicates that a cookie can only be transmitted across a secure Internet connection using HTTPS or another security protocol. Generally, when working with client-side JavaScript, the secure attribute should be omitted. However, if you wish to use this attribute, you assign it to the cookie property with a Boolean value of true or false, along with an associated name=value pair, using the syntax secure=*boolean value*. For example, to activate the secure attribute for a cookie, you use a statement similar to the following:

```
document.cookie = “firstName=”
   + encodeURIComponent(“Don”
   + “;secure=true”);
```

Reading Cookies with JavaScript

So far, you have stored both temporary and persistent cookies. Next, you need to learn how to retrieve stored cookie values—in other words, how to read cookies. The cookies for a particular Web page are available in the cookie property of the Document object. Cookies consist of one continuous string that must be parsed before the data they contain can be used. To parse a cookie, you must:

1. Decode it using the decodeURIComponent() function.
2. Use the methods of the String object to extract individual name=value pairs.

Parsing cookie data is very similar to parsing query strings, except that you do not need to remove the question mark at the beginning of the string; also, the individual cookies are separated by a semicolon and a space instead of ampersands. To give you an idea of what is involved in extracting data from cookies, the following code creates three encoded cookies, then reads them from the cookie property and decodes them. The split() method is then used to copy each name=value pair into the elements of an array named cookieArray[].

```
document.cookie = “city=” + encodeURIComponent(“Boston”);
document.cookie = “team=”
   + encodeURIComponent(“Red Sox”);
document.cookie = “sport=”
   + encodeURIComponent(“baseball”);
var cookieString = decodeURIComponent(
   document.cookie);
var cookieArray = cookieString.split(“; “);
```

Notice that the split() method in the preceding code splits the cookies by using two characters: a semicolon and a space. If you do not include the space in the split() method, then the name portion of each name=value pair in the new array has an extra space before it. Once you split the cookies into separate array elements, you still need to determine which cookie holds the value you need. The following for loop cycles through each element in the array, using an if statement and several string methods to check if the name portion of each name=value pair is equal to *team*. The conditional expression in the if statement uses the substring() method to return the name portion of the name=value pair in the variable named yourTeam. The first argument in the substring() method specifies the starting point of the substring as the first character (0). The second argument in the substring() method is the indexOf() method appended to the yourTeam variable, which returns the index number of the equal sign. If the substring is equal to *team*, then the for loop ends using a break statement, and the text *Your team is the* is written to the browser along with the value portion of the name=value pair. The statements that return the value portion of the name=value pair also use the substring() method along with the indexOf() method. However, this time the first argument starts the substring at the index number of the equal sign plus one, which is the character following the equal sign. The second argument in the substring() method specifies that the ending point of the substring is the length of the data variable.

```
var yourTeam;
for (var count = 0; count < 3; ++count) {
    yourTeam = cookieArray[count];
    if (yourTeam.substring(0,yourTeam.indexOf("="))
         == "team") {
        document.writeln("Your team is the "
           + yourTeam.substring(
           yourTeam.indexOf("=")+ 1,
           yourTeam.length));
        break;
    }
}
```

The preceding code is a little difficult to understand at first. If you are having trouble understanding how to manipulate strings, try experimenting with different string methods and see what you come up with.

Next, you will modify the populateCustomerInfo() and populateProductData() functions in the CustomerInfo.html and the ProductInfo.html files so that they read the stored cookies instead of query strings. The code in each function is almost identical to the query string versions, except that the cookie string is split with a semicolon and a space instead of an ampersand.

To modify the populateCustomerInfo() and populateProductData() functions in the CustomerInfo.html and the ProductInfo.html files so that they read the stored cookies instead of query strings:

1. Return to the **CustomerInfo.html** file in your text editor.
2. In the populateCustomerInfo() function, modify the conditional expression in the if statement so that it checks if the document.cookie exists instead of the location.search string.
3. Modify the statement that declares the queryData variable so that it is assigned document.cookie instead of the location.search string.
4. Modify the statement that declares the queryArray[] array so that it splits the array with a semicolon and a space instead of an ampersand. The modified function should appear as follows:

```
function populateCustomerInfo() {
   if (document.cookie) {
      var queryData = decodeURIComponent(
         document.cookie);
      var queryArray = queryData.split("; ");
      document.forms[0].name.value = queryArray[0]
         .substring(queryArray[0]
         .lastIndexOf("=") + 1);
```

```
document.forms[0].address1.value
   = queryArray[1].substring(
   queryArray[1].lastIndexOf(“=”) + 1);
document.forms[0].address2.value
   = queryArray[2].substring(
   queryArray[2].lastIndexOf(“=”) + 1);
document.forms[0].city.value
   = queryArray[3].substring(
   queryArray[3].lastIndexOf(“=”) + 1);
document.forms[0].state.value
   = queryArray[4].substring(
   queryArray[4].lastIndexOf(“=”) + 1);
document.forms[0].zip.value
   = queryArray[5].substring(
   queryArray[5].lastIndexOf(“=”) + 1);
document.forms[0].company.value
   = queryArray[6].substring(
   queryArray[6].lastIndexOf(“=”) + 1);
document.forms[0].email.value
   = queryArray[7].substring(
   queryArray[7].lastIndexOf(“=”) + 1);
document.forms[0].telephone.value
```

```
            = queryArray[8].substring(
            queryArray[8].lastIndexOf("=") + 1);
    }
}
```

5. Save the **CustomerInfo.html** document.
6. Return to the **ProductInfo.html** file in your text editor.
7. Replace the first statement in the populateProductData() function with the following statement, which assigns the document.cookie value to the queryData variable:

   ```
   var queryData = decodeURIComponent(document.cookie);
   ```

8. Modify the second statement in the if statement so that the queryData variable is split with a semicolon and space instead of an ampersand, as follows:

   ```
   var queryArray = queryData.split("; ");
   ```

9. Finally, modify the submitForm() function so that the first statement does not assign the returned value to the savedData variable. Then, add the following two statements after the statement that calls the saveProductData() function. The first statement creates a string variable named savedData and assigns to it a question mark and the contents of the document.cookie property. The second statement then uses a regular expression to replace all instances of "; " with ampersands so that the string can be passed as a query string.

   ```
   var savedData = "?" +
   decodeURIComponent(document.cookie);
   savedData = savedData.replace(/; /g, "&");
   ```

10. Save the **ProductInfo.html** document and then open the **ProductRegistration.html** document in a Web browser. The script should function the same as the query string version.
11. Close your Web browser window and the **CustomerInfo.html** and **ProductInfo.html** files in your text editor.

Deleting Cookies with JavaScript

You can delete cookies, although the way in which you delete them is not intuitive. To delete a cookie, you must set its expiration to a date in the past. The following code deletes the firstName cookie by setting its expires attribute to one week ago:

```
var expiresDate = new Date();

expiresDate.setDate(expiresDate.getDate() - 7);

document.cookie = "firstName=don" + "; expires="

    + expiresDate.toUTCString();
```

Short Quiz 2

1. Explain the difference between temporary and persistent cookies. How do you configure a cookie to be persistent?
2. How do you configure cookies to be available to other Web pages on the server?
3. How do you secure cookie transmissions?
4. How do you determine if a cookie exists?
5. How do you delete cookies?

Understanding Security Issues

Viruses, worms, data theft by hackers, and other types of security threats are now a fact of life when it comes to Web-based applications. If you put an application into a production environment without considering security issues, you are asking for trouble. To combat security violations, you need to consider both Web server security issues and secure coding issues. Web server security involves technologies such as firewalls, which combine software and hardware to prevent access to private networks connected to the Internet. One very important technology is the Secure Sockets Layer (SSL) protocol, which encrypts data and transfers it across a secure connection. These types of security technologies work well in the realm of the Internet. However, JavaScript programs are downloaded and execute locally within the Web browser of a client computer, and are not governed by security technologies such as firewalls and Secure Sockets Layer.

This section discusses security issues that relate to Web browsers and JavaScript.

Secure Coding with JavaScript

To provide even stronger software security, many technology companies, including Microsoft and Oracle, now require their developers and other technical staff to adhere to secure coding practices and principles. **Secure coding**, or **defensive coding**, refers to the writing of code in such a way that minimizes any intentional or accidental security issues. Secure coding has become a major goal for many information technology companies, primarily because of the exorbitant cost of fixing security flaws in commercial software. According to one study, it is 100 times more expensive to fix security flaws in released software than it is to apply secure coding techniques during the development phase. The National Institute of Standards & Technology estimates that $60 billion a year is spent identifying and correcting software errors. In addition, politicians have recently shown a great deal of interest in regulating software security. Tom Ridge, former Secretary of the U.S. Department of Homeland Security, recently said, "A few lines of code can wreak more havoc than a bomb." Intense government scrutiny gives information technology companies a strong incentive to voluntarily improve the security of software products before state and federal governments pass legislation that requires security certification of commercial software.

Basically, all code is insecure unless proven otherwise. Unfortunately, there is no magic formula for writing secure code, although there are various techniques that you can use to minimize security threats in your scripts. Your first line of defense in securing your JavaScript programs is to validate all user input. You have studied various techniques in this book for validating user input, including how to validate data with regular expressions and how to use exceptions to handle errors as they occur in your scripts. Be sure to use these techniques in your scripts, especially scripts that run on commercial Web sites. The remainder of this section discusses security issues that relate to Web browsers and JavaScript.

JavaScript Security Concerns

The Web was originally designed to be read-only, which is to say its primary purpose was to locate and display documents that existed on other areas of the Web. With the development of programming languages such as JavaScript, Web pages can now contain programs in addition to static content. This ability to execute programs within a Web page raises several security concerns. The security areas of most concern to JavaScript programmers are:

- Protection of a Web page and JavaScript program against malicious tampering
- Privacy of individual client information
- Protection of the local file system of the client or Web site from theft or tampering

Another security concern is the privacy of individual client information in the Web browser window. Your e-mail address, bookmarks, and history list are valuable pieces of information that many direct marketers would love to get their hands on in order to bombard you with advertising geared toward your likes and dislikes. Without security restrictions, a JavaScript program could read this information from your Web browser. One of the most important JavaScript security features is its *lack* of certain types of functionality. For example, many programming languages include objects and methods that make it possible for a program to read, write, and delete files. To prevent mischievous scripts from stealing information or causing damage by changing or deleting files, JavaScript does not allow any file manipulation whatsoever. Similarly, JavaScript does not include any sort of mechanism for creating a network connection. This limitation prevents JavaScript programs from infiltrating a private network or intranet from which information may be stolen or damaged. Another helpful limitation is the fact that JavaScript cannot run system commands or execute programs on a client. The ability to read and write cookies is the only type of access to a client that JavaScript has. Web browsers, however, strictly govern cookies and do not allow access to cookies from outside the domain that created them.

The Same Origin Policy

Another JavaScript security feature has to do with the **same origin policy**, which restricts how JavaScript code in one window or frame accesses a Web page in another window or frame on a client computer. For windows and frames to view and modify the elements and properties of documents displayed in other windows and frames, they must have the same protocol (such as HTTP) and exist on the same Web server. For example, documents from the following two domains cannot access each other's elements and properties because they use different protocols. The first domain's protocol is HTTP and the second domain's protocol is HTTPS, which, as mentioned earlier, is used on secure networks (that is, networks that run SSL).

```
http://www.gosselin.com
```

https://www.gosselin.com

The same origin policy applies not only to the domain name but also to the server on which a document is located. Therefore, documents from the following two domains cannot access each other's elements and properties, since they are located on different servers, even though they exist in the same domain of gosselin.com:

http://www.programming.gosselin.com

http://www.writing.gosselin.com

The same origin policy prevents malicious scripts from modifying the content of other windows and frames and prevents the theft of private browser information and information displayed on secure Web pages. How crucial is the same origin policy? Consider the src attribute of the Document object, which determines the URL displayed in a window or frame. If a client has multiple windows or frames open on its system and the same origin policy did not exist, then a Web page in one window or frame could change the Web pages displayed in other windows or frames. There are plenty of unscrupulous or simply malicious advertisers who would try to force you to view only their Web pages. The security of private networks and intranets would also be at risk without the same origin policy. Consider a user who has one Web browser open to a page on the Internet and another Web browser open to a secure page from his or her private network or intranet. Without the same origin policy, the Internet Web page would have access to the information displayed on the private Web page.

The same origin policy also protects the integrity of the design of your Web page. For example, without the same origin policy, a frame in one window or frame could modify the elements and properties of JavaScript objects and XHTML code in other windows and frames. To give you an idea of how the same origin policy prevents this type of scenario from occurring, you will now create a frame set in which one frame uses JavaScript code to try to change the status bar text of another frame, using the status property of the Document object.

To test the same origin policy:

1. Create a new document in your text editor.
2. Type the following code to create a frameset document. The code causes the Yahoo! Web page to appear in the second frame.

```
<!DOCTYPE html PUBLIC
"-//W3C//DTD XHTML 1.0 Frameset//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1- ↵
frameset.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Same Origin Policy</title>
<meta http-equiv="content-type"
   content="text/html; charset=iso-8859-1" />
</head>
<frameset cols="20%, *">
    <frame src="WrongOrigin.html"
       name="wrongFrame" />
    <frame src="http://www.yahoo.com"
       name="yahooFrame" />
</frameset>
</html>
```

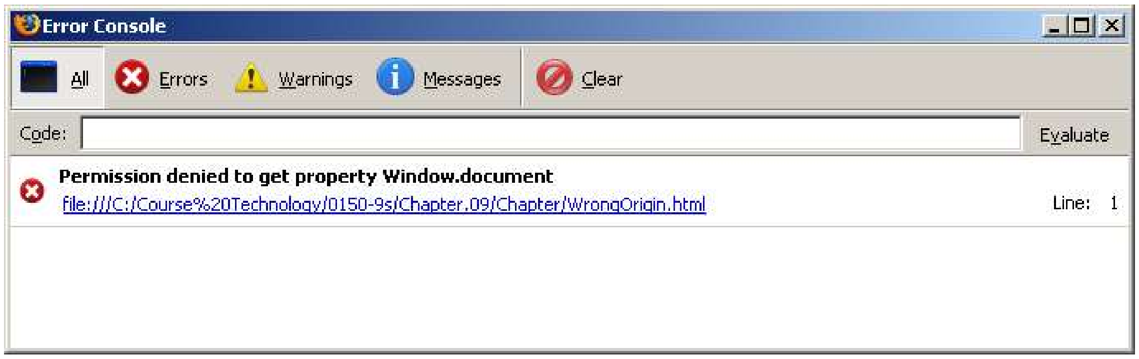
3. Save the document as **MainFrame.html** in your Chapter folder for Chapter 9, and then validate it with the W3C Markup Validation Service. Once the MainFrame.html document is valid, close it in your text editor.
4. Create another document in your text editor. Type the <!DOCTYPE> declaration, <html> element, document head, and document body. Use the strict DTD and "Same Origin Policy" as the content of the <title> element.

5. Add the following simple form, which contains a single button, which is called Change Status. The button uses an onclick event that tries to change the status bar text of the frame containing the Yahoo! Web page.

```
<form action="">
<p><input type="button" value="Change Status"
onclick="parent.yahooFrame.document.status='Visit ↵
   Don\'s Bait and Tackle Shop!'" /></p>
</form>
```

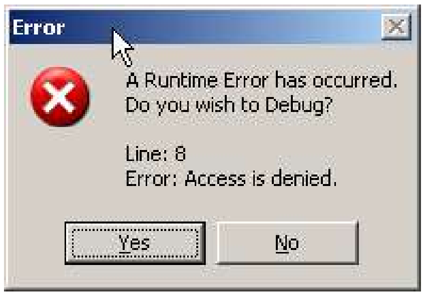
6. Save the document as **WrongOrigin.html** in your Chapter folder for Chapter 9, and then validate it with the W3C Markup Validation Service. Once the WrongOrigin.html document is valid, close it in your text editor.
7. Open the MainFrame.html document in your Web browser, and click the **Change Status** button. If you are using Firefox and the Error Console is open, you should receive an error message similar to the one shown in Figure 9-7. If you are using Internet Explorer and script debugging is enabled, you will see the error message shown in Figure 9-8.

Figure 9-7: Error message demonstrating the same origin policy in Firefox



Permission denied to get property Window.document
file:///C:/Course%20Technology/0150-9s/Chapter.09/Chapter/WrongOrigin.html Line: 1

Figure 9-8: Error message demonstrating the same origin policy in Internet Explorer


Error

A Runtime Error has occurred.
Do you wish to Debug?

Line: 8
Error: Access is denied.

8. Close your Web browser window.

In some circumstances, you will want two documents from related Web sites on different servers to be able to access each other's elements and properties. Consider a situation in which a document in the programming.gosselin.com domain needs to access content, such as form data, from a document in the writing.gosselin.com domain. To allow documents from different origins in the same domain to access each other's elements and properties, you use the domain property of the Document object. The **domain property** of the Document object changes the origin of a document to its root domain name by using the statement document.domain = "*domain*";. Adding the statement document.domain = "gosselin.com"; to documents from both programming.gosselin.com and writing.gosselin.com allows the documents to access each other's elements and properties, even though they are located on different servers.

Short Quiz 3

1. What is secure coding, and why is it so important?
2. What are some of the security areas of most concern to JavaScript?
3. What is the same origin policy?

Summing Up

- Information about individual visits to a Web site is called state information.
- HTTP was originally designed to be stateless, which means that Web browsers stored no persistent data about a visit to a Web site.

- A special type of form element, called a hidden form field, is not displayed by the Web browser. You can hide information from users in a hidden form field.
- A query string is a set of name=value pairs appended to a target URL. A query string consists of a single text string containing one or more pieces of information.
- Cookies are small pieces of information about a user that are stored by a Web server in text files on the user's computer.
- Cookies can be temporary or persistent. Temporary cookies remain available only for the current browser session. Persistent cookies remain available beyond the current browser session and are stored in a text file on a client computer.
- The cookie property is created with a required name attribute and four optional attributes: expires, path, domain, and secure.
- You can use special characters in your cookies if you use encoding, which involves converting special characters in a text string to their corresponding hexadecimal ASCII value, preceded by a percent sign.
- The built-in encodeURIComponent() function encodes the individual parts of a URI.
- When you read a cookie or other text string encoded with the encodeURIComponent() function, you must first decode it with the decodeURIComponent() function.
- Cookies consist of one continuous string that must be parsed before the data they contain can be used.
- To delete a cookie, you must set its expiration to a date in the past.
- The term "secure coding," or "defensive coding," refers to the writing of code in a way that minimizes any intentional or accidental security issues.
- The same origin policy restricts how JavaScript code in one window or frame accesses a Web page in another window or frame on a client computer.
- The domain property of the Document object changes the origin of a document to its root domain name using the statement document.domain = "*domain*";.

Comprehension Check

1. What is state information as it applies to Web sites?
2. Which of the following attributes can you use with an <input type="hidden"> element? (Choose all that apply.)
   a. visible

   b. name
   c. type
   d. value
3. The data stored in a hidden form field is not sent to a server along with the rest of the form. True or false?
4. What character is used for appending a query string to a URL?
   a. *
   b. $
   c. ?
   d. %
5. From where can you access a query string that is passed to a Web page? (Choose all that apply.)
   a. at the end of the URL in the Address box
   b. the search property of the Window object
   c. the search property of the Document object
   d. the search property of the Location object
6. What character separates entries in a query string?
   a. @
   b. ^
   c. &
   d. ~
7. What is the first task in parsing data from a query string?
   a. Use the split() method to convert the individual pieces of information in the query string into array elements.
   b. Encode the string with the encodeURIComponent() function.
   c. Use the toString() method to convert the query string to a JavaScript text string.
   d. Remove the question mark from the beginning of the string.
8. What is the difference between temporary and persistent cookies?
9. What is the correct syntax for creating a temporary cookie?

a. document.cookie = “language=” + “french”;

b. document.cookie = “language” + “french”;

c. cookie.temporary = “language” + “&” + “french”;

d. document.cookie = “?language” + “; “ + “french”;

10. Explain why you should use encoding with cookie values. What methods and procedures do you use to encode and decode cookie values?

11. In URL encoded format, what character is represented by %20?

    a. a space character

    b. an ampersand (&)

    c. an uppercase letter ‘A’

    d. a dollar sign ($)

12. You should always encode the value assigned to the expires attribute. True or false?

13. The availability of a cookie to other Web pages on a server is determined by the ___ attribute.

    a. system

    b. path

    c. directory

    d. server

14. Why should you store your JavaScript programs that create cookies in separate directories?

15. Which attribute is used for sharing cookies outside of a domain?

    a. share

    b. secure

    c. domain

    d. You cannot share cookies outside of a domain.

16. To delete a cookie, you must set its expiration to a date in the past. True or false?

17. Explain some of the steps you can take to write secure JavaScript code.

18. What are some of the ways in which JavaScript enforces the privacy of individual client information in the Web browser window?

19. Which of the following statements best describes the same origin policy?
    a. The same origin policy determines if and how a user allows cookies to be set on his or her computer.
    b. The same origin policy restricts how JavaScript code in one window or frame accesses a Web page in another window or frame on a client computer.
    c. The same origin policy allows Web sites to access e-mail addresses, bookmarks, history lists, and other types of client information that are stored in a user's Web browser.
    d. The same origin policy is a security protocol that verifies whether JavaScript code is running secure or unsecure mode as determined by the Web site's domain protocol.
20. To allow documents from different origins in the same domain to access each other's elements and properties, you use the ___.
    a. path attribute of the cookie property
    b. domain attribute of the cookie property
    c. domain property of the Document object
    d. origin property of the Window object

Reinforcement Exercises

Exercise 9-1

In this project, you will create a cookies program that stores the date and time of your last visit.

1. Create a new document in your text editor.
2. Type the <!DOCTYPE> declaration, <html> element, header information, and <body> element. Use the strict DTD and "Last Visit" as the content of the <title> element.
3. Add the following script section to the document body:

```
<script type="text/javascript">

/* <![CDATA[ */

/* ]]> */

</script>
```

4. Add the following if statement to the script section, which checks to see if a cookie exists for the current Web page. If it does, then statements within the if statement will extract and display the date and time of the last visit.

```
if (document.cookie) {
   var cookieString = decodeURIComponent(
      document.cookie);
   var cookieArray = cookieString.split("; ");
   var lastVisit =
        cookieArray[0].substring(
        cookieArray[0].indexOf("=")
        + 1, cookieArray[0].length);
    document.write("<p>Your last visit was "
       + lastVisit + "</p>");
}
else
    document.write("<p>This is your first
       visit.</p>");
```

5. Next, to the end of the script section, add the following statements, which use a Date object to assign the date and time of the current visit to the document's cookie:

```
var now = new Date();
var day = now.getDay();
var date = now.getDate();
var year = now.getFullYear();
var month = now.getMonth();
var hours = now.getHours();
var minutes = now.getMinutes();

var seconds = now.getSeconds();

var days = new Array();

days[0] = “Sunday”; days[1] = “Monday”;

days[2] = “Tuesday”; days[3] = “Wednesday”;

days[4] = “Thursday”; days[5] = “Friday”;

days[6]=”Saturday”;

var thisVisit = days[day] + “ “ + month + “/”

   + date + “/” + year + “ at “ + hours + “:”

   + minutes + “:” + seconds;

document.cookie = encodeURIComponent(thisVisit);
```

6. Save the document as **LastVisit.html** in a folder named LastVisit in your Exercises folder for Chapter 9, and then validate the document with the W3C Markup Validation Service. Once the document is valid, close it in your text editor.
7. Open the LastVisit.html document in your Web browser. The first time you open the document, you should see the text “This is your first visit.” Refresh your Web browser and you should see the date and time of your last visit.
8. Close your Web browser window.

Exercise 9-2

In the next few projects, you will create a Web site that simulates online banking for a company named Forestville Funding. Each customer’s user information will be stored in cookies. When a user visits the Web page again, he or she will be prompted to enter the stored user name and password. If the user does not enter the correct information within three tries, the script will prompt him or her to reregister. Note that for security reasons, browser cookies should never be the primary repository for a user’s login name and password. The purpose of this exercise is to demonstrate how you can use cookies to remember user data, including login information.

In this project, you will create the registration page for the Forestville Funding online banking site.

1. Create a new document in your text editor.

2. Type the <!DOCTYPE> declaration, <html> element, document head, and document body. Use the strict DTD and “Forestville Funding Online Banking” as the content of the <title> element.
3. Add the following text and elements to the document body. The form gathers each customer’s first name, last name, account number, user ID, and password. Clicking the Register button will call a function named registerForm() that stores the form values in cookies. You will create the registerForm() function next.

```
<h1>Forestville Funding</h1><hr />

<h2>Online Banking Registration</h2>

<form action=”” method=”get”

   enctype=”application/x-www-form-urlencoded”>

<p><strong>First Name</strong><br />

<input type=”text” name=”firstname” /></p>

<p><strong>Last Name</strong><br />

<input type=”text” name=”lastname” /></p>

<p><strong>Account Number</strong><br />

<input type=”text” name=”acctnum” /></p>

<p><strong>User ID</strong><br />

<input type=”text” name=”username” /></p>

<p><strong>Password</strong><br />

<input type=”password” name=”userpassword” /></p>

<p><input type=”button” value=”Register”

onclick=”registerForm();” /></p>

</form>
```

4. Add the following script section to the document head:

```
<script type=”text/javascript”>
```

```
/* <![CDATA[ */

/* ]]> */

</script>
```

5. Start creating the registerForm() function in the script section, as follows:

```
function registerForm() {

}
```

6. Add the following statements to the registerForm() function. These statements declare and initialize variables with the contents of the form fields.

```
var firstName = document.forms[0].firstname.value;

var lastName = document.forms[0].lastname.value;

var acctnum = document.forms[0].acctnum.value;

var userName = document.forms[0].username.value;

var userPassword = document.forms[0]

   .userpassword.value;
```

7. Add the following statements to the end of the registerForm() function. These statements declare a new Date object and set the year to one year from the current date. You will use the Date object to make the user information cookies persistent.

```
var myDate = new Date();

myDate.setFullYear(myDate.getFullYear() + 1);
```

8. Next, add the following statements to the end of the registerForm() function. These statements create persistent cookies out of the variables containing the values from the form fields.

```
document.cookie = “firstname=” +

encodeURIComponent(firstName)

   + “; expires=” + myDate.toUTCString();

document.cookie = “lastname=”

   + encodeURIComponent(lastName)
   + “; expires=” + myDate.toUTCString();
document.cookie = “acctnum=”
   + encodeURIComponent(acctnum)
   + “; expires=” + myDate.toUTCString();
document.cookie = “name=”
   + encodeURIComponent(userName)
   + “; expires=” + myDate.toUTCString();
document.cookie = “password=”
   + encodeURIComponent(userPassword)
   + “; expires=” + myDate.toUTCString();
```

9. Finally, add the following statements to the end of the function. The first statement displays an alert dialog box after the cookies are successfully created. The last statement opens the Forestville Funding online banking login page, which you will create in the next project.

```
window.alert(“Thank you for registering!”);
location.href = “ForestvilleFundingLogin.html”;
```

10. Save the document as **ForestvilleFundingRegistration.html** in a folder named ForestvilleFunding in your Exercises folder for Chapter 9, and then validate the document with the W3C Markup Validation Service. Once the document is valid, close it in your text editor.

Exercise 9-3

In this project, you create the login page for the Forestville Funding online banking Web site.

1. Create a new document in your text editor.
2. Type the <!DOCTYPE> declaration, <html> element, document head, and document body. Use the strict DTD and “Forestville Funding Online Banking” as the content of the <title> element.
3. Add the following text and elements to the document body. The form allows customers to enter their user IDs and passwords. Clicking the Log In button will call a function named checkUser() that determines whether the user entered a valid ID and password. You will create the checkUser() function next.

```
<h1>Forestville Funding</h1><hr />

<h2>Online Banking</h2>

<form action=”” method=”get”

   enctype=”application/x-www-form-urlencoded”>

<p><strong>User ID</strong><br />

<input type=”text” name=”username” /></p>

<p><strong>Password</strong><br />

<input type=”password” name=”userpassword” /></p>

<p><input type=”button” value=”Log In”

   onclick=”checkUser();” /></p>

</form>

<p><a href=”ForestvilleFundingRegistration.html”>

   Register</a></p>

<hr />

<p>Forestville Funding. Member FDIC. Equal Housing

Lender.<br />

&copy; 2008 Forestville Funding. All rights

reserved.</p>
```

4. Add the following script section to the document head:

```
<script type=”text/javascript”>

/* <![CDATA[ */

/* ]]> */

</script>
```

5. Start creating the checkUser() function in the script section, as follows:

```
function checkUser() {

}
```

6. Add the following statements to the checkUser() function. The if statement checks whether the cookie property exists. If it doesn't, then the ForestvilleFundingRegistration.html document opens. The attempts variable will track the number of times the user has attempted to log in.

```
if (document.cookie.length == 0){

    location.href = "ForestvilleFundingRegistration.html";

   return false;

}

var attempts = 0;
```

7. Add the following statements to the end of the checkUser() function. These statements declare and initialize variables with the contents of the form fields. The first statement decodes the contents of the document cookie and assigns its value to a variable named savedData. The second statement declares two variables, storedName and storedPassword, which will store the user ID and password from the cookies. The third and fourth statements assign the user ID and password values that the user entered into the form to variables named userName and userPassword. The final statement uses the split() method to create a variable named dataArray[] that contains the contents of the cookie, split into array elements.

```
var savedData = decodeURIComponent(document.cookie);

var storedName, storedPassword;

var userName = document.forms[0].username.value;

var userPassword = document.forms[0]

   .userpassword.value;

var dataArray = savedData.split("; ");
```

8. Add to the end of the checkUser() function the following for statement, which retrieves and assigns the user name and password values from dataArray[] to the storedName and storedPassword variables.

```
for (var i = 0; i < dataArray.length; ++i) {
    if (dataArray[i].substring(0,dataArray[i]
       .indexOf("="))== "name") {
       storedName = dataArray[i]
          .substring(dataArray[i].indexOf("=")
          + 1,dataArray[i].length);
    }
    if (dataArray[i].substring(0,dataArray[i]
       .indexOf("=")) == "password") {
       storedPassword = dataArray[i]
          .substring(dataArray[i].indexOf("=")
          + 1,dataArray[i].length);
    }
 }
```

9. Finally, add the following statements to the end of the checkUser() function. The first statement increments the attempts variable by a value of one. The if statement determines whether the user ID and password entered by the user match the values stored in the cookies. If so, a new temporary "login=successful" cookie is created and the ForestvilleFunding.html page opens. (You will create the ForestvilleFunding.html page in the next project.) The else statement displays an alert dialog box informing the user that his or her login attempt was unsuccessful. If the user has made three attempts to log in (as determined by the attempts variable), the ForestvilleFundingRegistration.html page opens.

```
++attempts;
if (userName != "" && userPassword != "") {
    if (userName == storedName && userPassword ==
     storedPassword) {
      document.cookie = "login=" +
         encodeURIComponent("successful");
        location.href = "ForestvilleFunding.html";
     }
     else {
        window.alert("Incorrect login or password.
         Please try again.");
        if (attempts == 3)
           location.href =
           "ForestvilleFundingRegistration.html";
     }
}
```

10. Save the document as **ForestvilleFundingLogin.html** in the ForestvilleFunding folder in your Exercises folder for Chapter 9, and then validate the document with the W3C Markup Validation Service. Once the document is valid, close it in your text editor.

Exercise 9-4

In this project, you create the main Forestville Funding online banking page that users see after they log in successfully.

1. Create a new document in your text editor.
2. Type the <!DOCTYPE> declaration, <html> element, document head, and document body. Use the strict DTD and "Forestville Funding Online Banking" as the content of the <title> element.
3. Add the following text and elements to the document body:

```
<h1>Forestville Funding</h1><hr />
<h2>Online Banking</h2>
<hr />
<p>Forestville Funding. Member FDIC. Equal
```

```
Housing Lender.<br />

&copy; 2008 Forestville Funding. All rights

reserved.</p>
```

4. Add the following script section to the document head:

```
<script type=”text/javascript”>

/* <![CDATA[ */

/* ]]> */

</script>
```

5. Add the following statements to the end of the script section. The first statement assigns the cookie value to the savedData variable. The if statement uses the search() method of the String object to determine whether the cookie value that is assigned to the savedData variable contains the value “login=successful”. If it does not, then the user has not logged in during the current browser session and the ForestvilleFundingLogin.html document opens. If the “login=successful” cookie is found, then the split() method creates a variable named dataArray[] that contains the contents of the cookie, split into array elements. The final statement declares firstName, lastName, and acctNum variables that will be assigned the values stored in the cookies.

```
var savedData = decodeURIComponent(document.cookie);

if (savedData.search(“login=successful”) == -1)

   location.href = “ForestvilleFundingLogin.html”;

var dataArray = savedData.split(“; “);

var firstName, lastName, acctNum;
```

6. Add to the end of the script section the following for statement, which retrieves and assigns the user’s first name, last name, and account number from dataArray[] to the firstName, lastName, and acctNum variables:

```
for (var i = 0; i < dataArray.length; ++i) {

   if (dataArray[i].substring(0,dataArray[i]

      .indexOf(“=”)) == “firstname”) {

      firstName = dataArray[i]
```

```
            .substring(dataArray[i].indexOf(“=”)
            + 1,dataArray[i].length);
    }
    if (dataArray[i].substring(0,dataArray[i]
        .indexOf(“=”)) == “lastname”) {
        lastName = dataArray[i]
           .substring(dataArray[i].indexOf(“=”)
           + 1,dataArray[i].length);
    }
    if (dataArray[i].substring(0,dataArray[i]
        .indexOf(“=”)) == “acctnum”) {
        acctNum = dataArray[i]
            .substring(dataArray[i].indexOf(“=”)
            + 1,dataArray[i].length);
    }
}
```

7. Finally, add the following script section to the document body, immediately after the <h2> element. The script section prints the values assigned to the firstName, lastName, and acctNum variables:

```
<script type=”text/javascript”>
/* <![CDATA[ */
document.write(“<p>You are currently logged in as “
    + firstName + “ “ + lastName + “.<br />”);
document.write(“Your account number is “ + acctNum
    + ".</p>");
/* ]]> */
</script>
```

8. Save the document as **ForestvilleFunding.html** in the ForestvilleFunding folder in your Exercises folder for Chapter 9, and then validate the document with the W3C Markup Validation Service. Once the document is valid, close it in your text editor and open it in your Web browser. Because you have not yet logged in, the ForestvilleFunding.html document should open the ForestvilleFundingLogin.html document. The ForestvilleFundingLogin.html document should in turn open the ForestvilleFundingRegistration.html document because you have not yet entered any registration information.
9. Enter information into the fields on the registration page, and click **Register**. An alert dialog box should appear, thanking you for registering. Click the **OK** button in the alert dialog box. This displays the login page.
10. Enter the user ID and password for the account that you just created into the fields on the login page, and then click **Log In**. The main Forestville Funding online banking page should appear, displaying the name and account number you entered.
11. Close your Web browser window.

Exercise 9-5

Many Web sites require cookies to be enabled in order to support certain types of Web page functionality, especially when it comes to logging in to a Web site. For example, if you at-tempt to log in to American Express at *https://www.americanexpress.com* when cookies are disabled in your browser, the login attempt will fail because the American Express Web site requires cookies to be enabled on client browsers to store security information and other types of data. Cookies are also required for the Forestville Funding online banking page. You will also add functionality that "remembers" login names and passwords so users won't need to log in every time they visit the Forestville Funding Web site. Note that with commercial applications, a user's login name and password are stored in cookies and are then retrieved by a Web server. Because you already stored the login name and password in cookies, you will just create another cookie named remember that is assigned a value of true.

1. Create a new document in your text editor.
2. Type the <!DOCTYPE> declaration, <html> element, document head, and document body. Use the strict DTD and "Forestville Funding Online Banking" as the content of the <title> element.
3. Add the following text and elements to the document body:

```
<h1>Forestville Funding</h1><hr />

<h2>Online Banking</h2>

<p>This Web site requires that your browser

accept cookies.</p>

<hr />

<p>Forestville Funding. Member FDIC. Equal

Housing Lender.<br />

&copy; 2008 Forestville Funding. All rights

reserved.</p>
```

4. Save the document as **ForestvilleFundingNoCookies.html** in the ForestvilleFunding folder in your Exercises folder for Chapter 9, and then validate the document with the W3C Markup Validation Service. Once the document is valid, close it in your text editor.
5. Open in your text editor the **ForestvilleFundingLogin.html** document from the ForestvilleFunding folder in your Exercises folder for Chapter 9. Modify the first if statement in the script section as follows. If no document cookie exists, then the first statement in the if statement writes a test cookie. The nested if statement then checks again to see if a document cookie exists, indicating that the test cookie was written successfully. If the cookie exists, the page is redirected to the ForestvilleFundingRegistration.html and the test cookie is deleted. If the cookie does not exist, it indicates that the browser is blocking cookies and the else clause redirects the page to ForestvilleFundingNoCookies.html.

```
if (!document.cookie) {

   document.cookie = “test”;

   if (document.cookie.length = 0) {

      location.href

         = “ForestvilleFundingRegistration.html”;

      var expiresDate = new Date();

      expiresDate.setDate(
            expiresDate.getDate() - 7);
        document.cookie = "test" + "; expires="
            + expiresDate.toUTCString();
    }
    else
        location.href
            = "ForestvilleFundingNoCookies.html";
}
```

6. Add the following text and elements immediately above the paragraph in the document body that contains the Log In button:

```
<p><input type="checkbox" name="remember_me" />
Remember my login information</p>
```

7. Modify the if statement at the end of the checkUser() function so that it includes the following bolded if statement, which creates a "remember=true" cookie if the remember_me check box is selected:

```
if (userName == storedName && userPassword
    == storedPassword) {
    document.cookie = "login="
        + encodeURIComponent("successful");
    if (document.forms[0].elements[2].checked) {
        var myDate = new Date();
        myDate.setFullYear(myDate.getFullYear()
            + 1);
        document.cookie = "remember="
            + encodeURIComponent("true")
        + "; expires=" + myDate.toUTCString();
    }
    location.href = "ForestvilleFunding.html";
}
```

8. Save the **ForestvilleFundingLogin.html** document, and validate it with the W3C Markup Validation Service. Once the document is valid, close it in your text editor.
9. Open in your text editor the **ForestvilleFunding.html** document from the ForestvilleFunding folder in your Exercises folder for Chapter 9, and modify the statement that checks for the "login=successful" cookie so that it also checks for the "remember=true" cookie, as shown with the following bolded code. If both search() methods return a value of -1 for both cookies, then the ForestvilleFundingLogin.html document opens because the user: (1) is not logged in for the current session, and (2) has not selected the remember_me button on the ForestvilleFundingLogin.html page.

```
if (savedData.search("login=successful") == -1
    && savedData.search("remember=true") == -1)
    location.href = "ForestvilleFundingLogin.html";
```

10. Save the **ForestvilleFunding.html** document, and validate it with the W3C Markup Validation Service. Once the document is valid, close it in your text editor.
11. Open your Web browser and disable cookies.

- To disable cookies in Firefox, select **Options** from the Tools menu and click the **Privacy** tab. On the Privacy tab, select **Use custom settings for history** from the **Firefox will** box, and then deselect the **Accept cookies from sites** box and click **OK**.
- To disable cookies in Internet Explorer, select **Internet Options** from the Tools menu and click the **Privacy** tab. On the Privacy tab, click the **Advanced** button to display the Advanced Privacy Settings dialog box. In the Advanced Privacy Settings dialog box, select the **Override automatic cookie handling** box and then select the **Block** radio buttons in the First-party Cookies and Third-party Cookies sections. Click **OK** to close the Advanced Privacy Settings dialog box and click **OK** again to close the Internet Options dialog box.

12. Close your Web browser after you have disabled cookies.
13. Open the **ForestvilleFunding.html** document in your Web browser. Because you have disabled cookies, the ForestvilleFundingNoCookies.html document should open. Follow the same procedures listed in Step 11 to reenable cookies, and then close your Web browser.
14. Open the **ForestvilleFunding.html** document again in your Web browser. Because you are starting a new brower session, the temporary "login=successful" cookie has ceased to exist, so the ForestvilleFundingLogin.html document should open. Enter

your user ID and password, click the **Remember my login information** box, and then click **Log In**. Your account information should display on the ForestvilleFunding.html page.

15. Close your Web browser and then reopen the **ForestvilleFunding.html** document again in your Web browser. Because you selected the Remember my login information box, your account information should appear -immediately on the ForestvilleFunding.html page; you will not be redirected to the ForestvilleFundingLogin.html page.
16. Close your Web browser window.

Exercise 9-6

In this project, you will correct errors in a cookie program.

1. Create a new document in your text editor.
2. Type the <!DOCTYPE> declaration, <html> element, header information, and the <body> element. Use the strict DTD and "Cookie Errors" as the content of the <title> element.
3. Add the following script section to the document body:

```
<script type=”text/javascript”>

/* <![CDATA[ */

var visitData = decodeURIComponent(document.cookie);

if (visitData.length = 0)

    document.write(“<p>You have visited ↵

        before.</p>”);

else

    document.write(“<p>This is your first ↵

        visit.</p>”);

var expiresDate = new Date();

expiresdate.setFullYear(expiresDate.getFullYear()

   - 1);

document.cookie = encodeURIComponent(“expires=”
    + expiresDate.toUTCString());
/* ]]> */
</script>
```

4. Save the document as **CookieErrors.html** in a folder named CookieErrors in your Exercises folder for Chapter 9, and validate it with the W3C Markup Validation Service. Once the CookieErrors.html document is valid, open it in your Web browser. The first time you open the document, you should see the text "This is your first visit." If you close and then reopen your Web browser (rather than refreshing your Web browser window), you will continue to receive the message "This is your first visit." Fix the errors in the document. (*Hint*: There is more than one error in the program.)
5. Close your Web browser window.

Discovery Projects

For the following projects, save the documents you create in your Projects folder for Chapter 9. Be sure to validate the documents you create with the W3C Markup Validation Service. Also, be sure to create each document in its own folder in order to avoid conflicts with cookies that are set by other Web pages.

Project 9-1

Create a document that stores and reads cookies that track the number of times a user has visited your Web site and the date of his or her last visit. The first time the user visits, display a message welcoming him or her to your Web site and reminding him or her to bookmark the page. Whenever a user visits the site, display the cookies using `document.write()` statements, increment the counter cookie by one, and then reset the counter cookie expiration date to one year from the current date. Save the document as Counter.html.

Project 9-2

Create a document with a "nag" counter that reminds users to register. Save the counter in a cookie and display a message reminding users to register every fifth time they visit your site. Create a form in the body of the document that includes text boxes for a user's name and e-mail address along with a Register button. Once a user fills in the text boxes and clicks the Register button, delete the nag counter cookie and replace it with cookies containing the user's name and e-mail address. After registering, display the name and e-mail address cookies in an alert message whenever the user revisits the site. Save the document as NagCounter.html.

Project 9-3

Create a document with a form that registers users for a marketing seminar. When a user submits the registration form, store cookies containing the user's information such as name, company, and so on. If a user attempts to register a second time with the same name, display a confirm dialog box asking if he or she wants to register again. Save the document as MarketingSeminar.html.

Project 9-4

Create a document with a form for reserving a rental car. As a user creates a reservation, store cookies containing the user's reservation information, including name and address, telephone, pickup and return dates, and car type. Also, create a button that redisplay a user's reservation information with an alert message. Set the cookies so that they expire one day after a visit. Save the document as CarRentals.html.

You can also use a global JavaScript variable to add storage functionality to the calculator script.

You will not add validation code to the Printer Product Registration forms; this way, you can focus on the techniques presented in this chapter.

The search property of the Location object gets its name from the fact that many Internet search engines use the query string it contains to store search criteria.

To modify an existing cookie, you simply assign a new name=value pair to the document.cookie property. If the name=value pair already exists it will be overwritten.

JavaScript also includes the encodeURI() and decodeURI() functions, which can be used to encode and decode entire URIs. Be sure to distinguish these functions from the encodeURIComponent() and decodeURIComponent() functions, which encode and decode the individual parts of a URI.

Older versions of JavaScript use the deprecated escape() and unescape() methods for encoding and decoding text strings.

Coordinated Universal Time is also known as Greenwich Mean Time (GMT), Zulu time, and world time.

Take care not to encode the expires attribute using the encodeURIComponent() method. JavaScript does not recognize a UTC date when it is in URI-encoded format. If you use the encodeURIComponent() method with the expires attribute, JavaScript is not able to set the cookie expiration date.

Using string methods to parse a cookie is the only way to extract individual pieces of information from a long cookie string, so it is important that you understand how they work.

Although Web server security issues are critical, they are properly covered in books on Apache, Internet Information Services, and other types of Web servers. Be sure to research security issues for your Web server and operating system before activating a production Web site.

To view errors in Firefox Web browsers, you need to select Error Console from the Tools menu in Firefox 2.0 or later, or the JavaScript Console in Firefox versions earlier than 2.0. If you are using a version of Internet Explorer higher than 4.0, you need to turn on error notification. To verify that error notification is turned on in Internet Explorer, click Tools on the menu bar, click Internet Options, click the Advanced tab, in the Browsing category click the

Display a notification about every script error check box to select it (if necessary), and then click OK. Other Web browsers may also require you to turn on error notification.

The Forestville Funding site will not include validation functionality in order to allow you to focus on the cookie techniques.

An easier way to determine whether cookies are enabled in a Web browsers is to use the cookieEnabled property of the Navigator object. However, current versions of Internet Explorer contain a bug that results in the cookieEnabled property always being set to true, even when cookies are disabled in Internet Explorer. For this reason, you need to use the preceding longer code.