

What Are Cookies?

Prior to the existence of cookies, traversing a Web site was a trip without a history. Although the browser tracks the pages you visit, allowing you to use the Back button to return to previously visited pages and indicating visited links in a different color, the server does not follow what individual users see and do. This is still true for sites that don't use cookies, as well as for users who have disabled cookies in their Web browsers **A**.

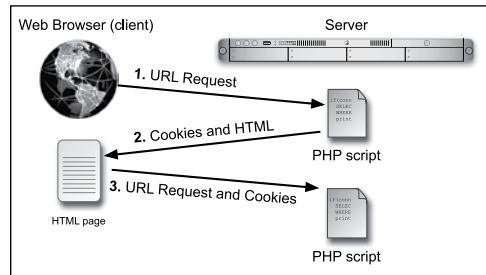
Why is that a problem? If the server can't track a user, there can be no shopping carts for making purchases online. If cookies didn't exist (or if they're disabled in the Web browser), people wouldn't be able to use popular sites that require user registration. In short, without cookies, there would be no Amazon or Facebook or any of the other most popular or useful sites (not in their current incarnations, at least).

Cookies are simply a way for a server to store information on the user's computer. By doing so, the server can remember the user over the course of a visit or through several visits. Think of a cookie like a name tag: You tell the server your name, and it gives you a name tag. Then it can know who you are by referring back to the name tag.

This brings up another point about the security issues involved with cookies. Cookies have gotten a bad rap because users believe cookies allow a server to know too much about them. However, a cookie can only be used to store information that you give it, so it's as secure as you want it to be. And, as previously mentioned, it's very easy in modern browsers to customize the cookie handling as desired.



A Most Web browsers let users set the cookie-handling preferences. This is Internet Explorer 8's Advanced Privacy Settings tab.



B How cookies are sent back and forth between the server and the client.

PHP has very good support for cookies. In this chapter, you'll learn how to set a cookie, retrieve information from a cookie, and then delete the cookie. You'll also see some of the optional parameters you can use to apply limits to a cookie's existence.

Before moving on, there are two more things you ought to know about cookies. The first is how to debug cookie-related problems. You'll inevitably need to know how to do that, so the topic is discussed in the sidebar "Debugging Cookies." The second is how a cookie is transmitted and received **B**. Cookies are stored in the Web

browser, but only the site that originally sent a cookie can read it. Also, the cookies are read by the site when the page on that site is requested by the Web browser. In other words, when the user enters a URL in the address bar and clicks Go (or whatever), the site reads any cookies it has access to and then serves up the requested page. This order is important because it dictates when and how cookies can be accessed.

TIP The ability to send, read, and delete cookies is one of the few overlaps between server-side PHP and browser-side JavaScript.

Debugging Cookies

When you begin working with cookies in PHP, you'll need to know how to debug your cookie-related scripts when difficulties arise. Three areas might cause you problems:

- Sending the cookie with PHP
- Receiving the cookie in your Web browser
- Accessing a cookie in a PHP script

The first and last issues can be debugged by printing out the variable values in your PHP scripts (as you'll soon learn). The second issue requires that you know how to work with cookies in your Web browser. For debugging purposes, you'll want your Web browser to notify you when a cookie is being sent.

With Internet Explorer on Windows, you can do this by choosing Internet Options under the Tools menu. Then click the Privacy tab, followed by the Advanced button under Settings. Click "Override automatic cookie handling," **A** and then choose Prompt for both First-party and Third-party Cookies (you can actually block the third-party cookies, if you'd rather). Other versions of Internet Explorer may use different variations on this process. Internet Explorer also has a Developer Tools window (linked under the Tools menu) that can be useful.

The best way to debug cookies when using Firefox on any platform is to install one of the many cookie-related extensions, like *Firecookie*. The *Firebug* extension, which every developer must use, also shows cookie-related information. But at the very least, if you select "Use custom settings for history" on the Privacy panel (in the Options/Preferences window), you'll be able to establish custom cookie behavior, such as being prompted.

Safari on Mac OS X and Windows doesn't give you as many cookie choices; you can find the available options on the Security tab of the Preferences window.

Some browsers also let you browse through the existing cookies to see their names and values. Doing so is a great asset in the debugging war.

Creating Cookies

An important thing to understand about cookies is that *they must be sent from the server to the client prior to any other information*. This means a script should send cookies before any **print** statement, before including an external file that contains HTML, and so forth.

Should the server attempt to send a cookie after the Web browser has already received HTML—even an extraneous white space—an error message will result and the cookie won't be sent **A**. This is by far the most common cookie-related error.

Cookies are sent using the **setcookie()** function:

```
setcookie(name, value);  
setcookie('CookieName', 'This is the  
→ cookie value.');
```

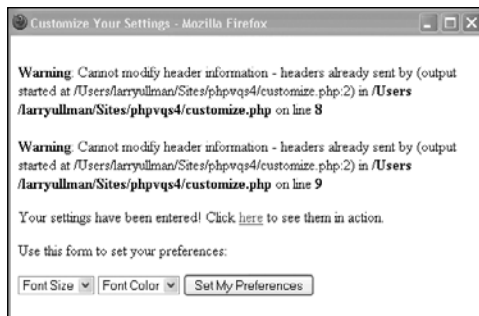
That line of code sends to the browser a cookie with the name *CookieName* and the value *This is the cookie value.* **B**.

You can continue to send more cookies to the browser with subsequent uses of the **setcookie()** function, although you're limited by the Web browser as to how many cookies can be sent from the same site:

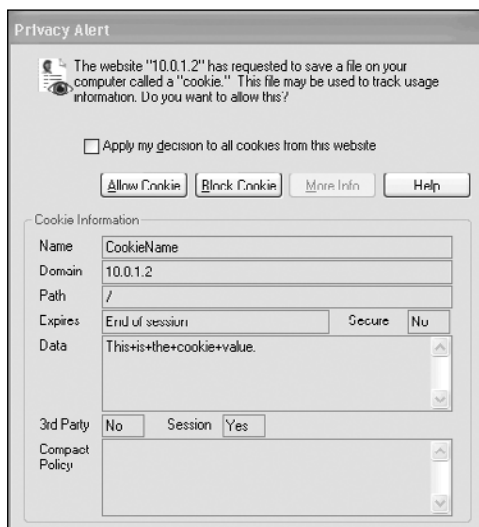
```
setcookie('name2', 'some value');  
setcookie('name3', 'another value');
```

Finally, when creating cookies, you can—as you'll see in this example—use a variable for the name or value attribute of your cookies:

```
setcookie($cookie_name, $cookie_value);
```



A A message like this is what you'll see if the **setcookie()** function is called after anything, even a blank line or space, has already been sent to the Web browser.



B If the browser is set to prompt the user for cookies, a message like this will appear for each cookie sent. (Note that the window, from Internet Explorer 8, shows the value in a URL-encoded format.)

Use this form to set your preferences:

Font Size Font Color

C This form is used to select the font size and color for use on another PHP page.

Your settings have been entered! Click [here](#) to see them in action.

Use this form to set your preferences:

Font Size Font Color

D After submitting the form, the page shows a message and a link to another page (where the user's preferences will be used). That page will be created next.

Script 9.1 Two cookies will be used to store the user's choices for the font size and color. This page both displays and handles the form.

```

1  <?php // Script 9.1 - customize.php
2
3  // Handle the form if it has been
   submitted:
4  if (isset($_POST['font_size'],
   $_POST['font_color'])) {
5
6      // Send the cookies:
7      setcookie('font_size',
   $_POST['font_size']);
8      setcookie('font_color',
   $_POST['font_color']);
9
10     // Message to be printed later:
11     $msg = '<p>Your settings have
   been entered! Click <a href=
   "view_settings.php">here</a>
   to see them in action.</p>';
12
13 } // End of submitted IF.
14 ?><!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
   1.0 Transitional//EN"
15     "http://www.w3.org/TR/xhtml1/DTD/
   xhtml1-transitional.dtd">
16 <html xmlns="http://www.w3.org/1999/
   xhtml" xml:lang="en" lang="en">
```

code continues on next page

For an example of setting cookies, you'll create a script that allows the user to specify the default font size and color for a page. The page displays a form for choosing these values **C** and then handles the form submission **D**. A separate page, created in the next section of this chapter, will use these settings.

To send cookies:

1. Create a new PHP document in your text editor or IDE, to be named **customize.php (Script 9.1)**:

<?php // Script 9.1 - customize.php

The most critical issue with cookies is that they're created before anything is sent to the Web browser. To accomplish this, the script begins with a PHP section that handles the sending of cookies.

Also be certain not to have any extraneous spaces or lines before the initial PHP tag.

2. Check whether the form has been submitted:

```
if (isset($_POST['font_size'],
→ $_POST['font_color'])) {
```

This page will both display and handle the form. It could use the same method explained in the previous chapter—checking if the **\$_SERVER['REQUEST_METHOD']** variable has a value of **POST**, but as an alternative approach, the script will perform basic, minimal validation as the test for a form submission. The conditional checks for the existence of two variables: **\$_POST['font_size']** and **\$_POST['font_color']**. If both are set, the form submission will be addressed.

continues on next page

3. Create the cookies:

```
setcookie('font_size',
→ $_POST['font_size']);
setcookie('font_color',
→ $_POST['font_color']);
```

These two lines create two separate cookies. One is named **font_size** and the other **font_color**. Their values will be based on the selected values from the HTML form, which are stored in the `$_POST['font_size']` and `$_POST['font_color']` variables.

In a more fully developed application, you should first confirm that the variables both have acceptable values.

4. Create a message and complete the conditional and the PHP section:

```
$msg = '<p>Your settings
→ have been entered! Click
→ <a href="view_settings.
→ php">here</a> to see them in
→ action.</p>';
} // End of submitted IF.
?>
```

When the form has been submitted, the cookies will be sent and the `$msg` variable will be assigned a string value. This variable will be used later in the script to print a message. This approach is necessary, as you can't print the message at this juncture (because not even the HTML head has been created).

5. Create the HTML head and opening body tag:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD
→ XHTML 1.0 Transitional//EN"
→ "http://www.w3.org/TR/xhtml1/
→ DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/
→ 1999/xhtml" xml:lang="en"
→ lang="en">
```

Script 9.1 continued

```
17 <head>
18 <meta http-equiv="Content-Type"
content="text/html; charset=utf-8"/>
19 <title>Customize Your Settings
</title>
20 </head>
21 <body>
22 <?php // If the cookies were sent, print
a message.
23 if (isset($msg)) {
24 print $msg;
25 }
26 ?>
27
28 <p>Use this form to set your
preferences:</p>
29
30 <form action="customize.php"
method="post">
31 <select name="font_size">
32 <option value="">Font Size</option>
33 <option value="xx-small">xx-small
</option>
34 <option value="x-small">x-small
</option>
35 <option value="small">small
</option>
36 <option value="medium">medium
</option>
37 <option value="large">large
</option>
38 <option value="x-large">x-large
</option>
39 <option value="xx-large">xx-large
</option>
40 </select>
41 <select name="font_color">
42 <option value="">Font Color
</option>
43 <option value="999">Gray</option>
44 <option value="0c0">Green</option>
45 <option value="00f">Blue</option>
46 <option value="c00">Red</option>
47 <option value="000">Black</option>
48 </select>
49 <input type="submit" name="submit"
value="Set My Preferences" />
50 </form>
51
52 </body>
53 </html>
```

```

<head>
  <meta http-equiv="Content-Type"
    → content="text/html;
    → charset=utf-8"/>
  <title>Customize Your Settings
    → </title>
</head>
<body>

```

All of this code must come after the `setcookie()` lines. Not to overstate the fact, but no text, HTML, or blank spaces can be sent to the Web browser prior to the `setcookie()` calls.

6. Create another PHP section to report on the cookies being sent:

```

<?php
if (isset($msg)) {
  print $msg;
}
?>

```

This code prints out a message if the cookies have been sent. The first time the user comes to the page, the cookies haven't been sent, so `$msg` is not set, making this conditional **FALSE**, and this `print` invocation never runs. Once the form has been submitted, `$msg` has been set by this point, so this conditional is **TRUE** ^D.

7. Begin the HTML form:

```

<p>Use this form to set your
→ preferences:</p>
<form action="customize.php"
→ method="post">
  <select name="font_size">
    <option value="">Font Size
    → </option>
    <option value="xx-small">
    → xx-small</option>
    <option value="x-small">
    → x-small</option>
    <option value="small">small
    → </option>

```

```

    <option value="medium">medium
    → </option>
    <option value="large">large
    → </option>
    <option value="x-large">x-large
    → </option>
    <option value="xx-large">
    → xx-large</option>
  </select>

```

The HTML form itself is very simple ^C. The user is given one drop-down menu to select the font size. The value for each corresponds to the CSS code used to set the document's font size: from *xx-small* to *xx-large*.

Because this script both displays and handles the form, the form's **action** attribute points to the same file.

8. Complete the HTML form:

```

  <select name="font_color">
    <option value="">Font Color
    → </option>
    <option value="999">Gray</option>
    <option value="0c0">Green</option>
    <option value="00f">Blue</option>
    <option value="c00">Red</option>
    <option value="000">Black</option>
  </select>
  <input type="submit" name=
    → "submit" value="Set My
    → Preferences" />
</form>

```

The second drop-down menu is used to select the font color. The menu displays the colors in text form, but the values are HTML color values. Normally such values are written using six characters plus a pound sign (e.g., `#00cc00`), but CSS allows you to use just a three-character version and the pound sign will be added on the page that uses these values.

continues on next page

9. Complete the HTML page:

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

10. Save the file as **customize.php** and place it in the proper directory for your PHP-enabled server.

11. Make sure you've set your Web browser to prompt for each cookie, if applicable.

To guarantee that the script is working, you want the browser to prompt you for each cookie, if you can. See the "Debugging Cookies" sidebar.

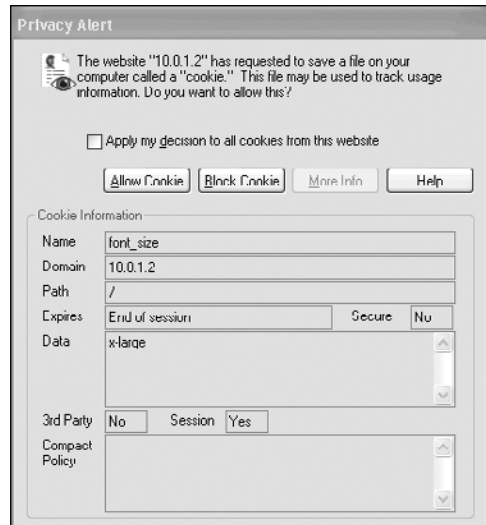
12. Run the script in your Web browser **E** and **F**.

TIP Cookies are one of the few areas in PHP that can behave differently from browser to browser or operating system to operating system. You should test your cookie-based applications on as many browsers and operating systems as you can.

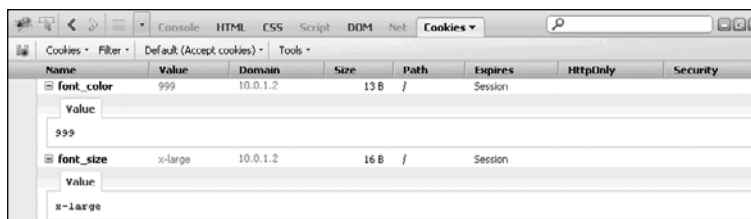
TIP If you use the output buffering technique taught in Chapter 8, then you can place your `setcookie()` calls anywhere within the script (because the Web browser won't receive the data until the `ob_end_flush()` function is called).

TIP Cookies are limited to approximately 4 KB of total data. This is more than sufficient for most applications.

TIP To test whether it's safe to send a cookie, use the `headers_sent()` function. It reports on whether HTTP headers have already been sent to the Web browser.



E The user sees this message when the first `setcookie()` call is made, if they've opted to be prompted before accepting a cookie. This cookie is storing the value of *x-large* in a cookie named *font_size*.



F The Firebug extension for Firefox shows the cookies received by the browser. The second cookie that's sent by the PHP script is called *font_color* and has a value of *999*, representing the color gray.

Script 9.2 This script sets the font size and color in CSS, using the values stored in the cookies.

```
1  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
1.0 Transitional//EN"
2    "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
3  <html xmlns="http://www.w3.org/1999/
xhtml" xml:lang="en" lang="en">
4  <head>
5    <meta http-equiv="content-type"
content="text/html; charset=utf-8" />
6    <title>View Your Settings</title>
7    <style type="text/css">
8      body {
9  <?php // Script 9.2 - view_settings.php
10
11    // Check for a font size value:
12    if (isset($_COOKIE['font_size'])) {
13      print "\t\tfont-size: " .
htmlentities($_COOKIE['font_
size']) . ";\n";
14    } else {
15      print "\t\tfont-size: medium;";
16    }
17
18    // Check for a font_color value:
19    if (isset($_COOKIE['font_color'])) {
20      print "\t\tcolor: #" .
htmlentities($_COOKIE['font_
color']) . ";\n";
21    } else {
22      print "\t\tcolor: #000;";
23    }
24
25    ?>
26      }
27    </style>
28  </head>
29  <body>
30    <p><a href="customize.php">Customize Your
Settings</a></p>
31    <p><a href="reset.php">Reset Your
Settings</a></p>
32
33    <p>yadda yadda yadda yadda yadda
34    yadda yadda yadda yadda yadda
35    yadda yadda yadda yadda yadda
36    yadda yadda yadda yadda yadda
37    yadda yadda yadda yadda yadda</p>
38
39  </body>
40  </html>
```

Reading from Cookies

Just as form data is stored in the **\$_POST** array (assuming it used the POST method) and values passed to a script in the URL are stored in the **\$_GET** array, the **setcookie()** function places cookie data in the **\$_COOKIE** array. To retrieve a value from a cookie, you only need to refer to the cookie name as the index of this array. For example, to retrieve the value of the cookie established with the line

```
setcookie('user', 'trout');
```

you would use the variable **\$_COOKIE['user']**.

Unless you change the cookie's parameters (as you'll see later in this chapter), the cookie will automatically be accessible to every other page in your Web application. You should understand, however, that a cookie is never accessible to a script immediately after it's been sent. You can't do this:

```
setcookie('user', 'trout');
print $_COOKIE['user']; // No value.
```

The reason for this is the order in which cookies are read and sent (see **B** in the first section of this chapter).

To see how simple it is to access cookie values, let's write a script that uses the preferences set in **customize.php** to specify the page's text size and color. The script relies on CSS to achieve this effect.

To retrieve cookie data with PHP:

1. Begin a new PHP document in your text editor or IDE, to be named **view_settings.php** (Script 9.2):

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

continues on next page


```

<html xmlns="http://www.w3.org/
→ 1999/xhtml" xml:lang="en"
→ lang="en">
<head>
  <meta http-equiv="content-type"
  → content="text/html;
  → charset=utf-8" />
  <title>View Your Settings
  → </title>

```

2. Start the CSS section:

```

<style type="text/css">
  body {

```

The page will use CSS to enact the user's preferences. The aim is to create code like

```

body {
  font-size: x-large;
  color: #999;
}

```

The two values will differ based on what the user selected in the `customize.php` page. In this step, you create the initial CSS tag.

3. Open a section of PHP code:

```

<?php // Script 9.2 -
→ view_settings.php

```

The script will now use PHP to print out the remaining CSS, based on the cookies.

4. Use the font size cookie value, if it exists:

```

if (isset($_COOKIE['font_size'])) {
  print "\t\tfont-size: " .
  → htmlentities($_COOKIE
  → ['font_size']) . ";\n";
} else {
  print "\t\tfont-size: medium;";
}

```

If the script can access a cookie with a name of `font_size`, it will print out that cookie's value as the CSS `font-size` value. The `isset()` function is

sufficient to see if the cookie exists. If no such cookie exists, PHP will print out a default size, *medium*.

For security purposes, the cookie's value is not directly printed. Instead, it's run through the `htmlentities()` function, discussed in Chapter 5, "Using Strings." This function will prevent bad things from happening should the user manipulate the value of the cookie (which is easy to do).

Also note that two tabs (`\t`) and a newline (`\n`) are added to the `print` statements so that the resulting CSS code is formatted properly. Not that this affects the functionality of the page, but...

5. Repeat this process for the font color cookie:

```

if (isset($_COOKIE['font_color'])) {
  print "\t\tcolor: #" .
  → htmlentities($_COOKIE
  → ['font_color']) . ";\n";
} else {
  print "\t\tcolor: #000;";
}

```

Here the CSS's `color` attribute is being assigned a value. The cookie itself is used the same as in Step 4.

6. Close the PHP section, complete the CSS code, and finish the HTML head:

```

?>
  }
  </style>
</head>

```

7. Start the HTML body and create links to two other pages:

```

<body>
<p><a href="customize.php">
→ Customize Your Settings</a></p>
<p><a href="reset.php">Reset Your
→ Settings</a></p>

```



A This page reflects the customized font choices made using the other PHP script.



B By viewing the source code of the page, you can also track how the CSS values change.

These two links take the user to two other PHP pages. The first, **customize.php**, has already been written and lets the user define their settings. The second, **reset.php**, will be written later in the chapter and lets the user delete their customized settings.

8. Add some text:

```
<p>yadda yadda yadda yadda yadda
yadda yadda yadda yadda yadda
yadda yadda yadda yadda yadda
yadda yadda yadda yadda yadda
yadda yadda yadda yadda yadda</p>
```

This text exists simply to show the effects of the cookie changes.

9. Complete the HTML page:

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

10. Save the file as **view_settings.php**, place it in the same directory as **customize.php**, and test it in your Web browser **A** by clicking the link on **customize.php**.

11. View the source of the page to see the resulting CSS code **B**.

12. Use the customize page to change your settings and return to this script.

Each submission of the form will create two new cookies storing the form values, thereby replacing the existing cookies.

TIP The value of a cookie is automatically encoded when it's sent and decoded on being received by the PHP page. The same is true of values sent by HTML forms.

Adding Parameters to a Cookie

Although passing just the **name** and **value** arguments to the **setcookie()** function will suffice for most of your cookie uses, you ought to be aware of the other arguments available. The function can take up to five more parameters, each of which limits the operation of the cookie:

setcookie(name, value, expiration,
→ **path, domain, secure, httponly);**

The **expiration** argument is used to set a specific length of time for a cookie to exist. If it isn't specified, the cookie will continue to be functional until the user closes the browser. Normally, you set the expiration time by adding a particular number of minutes or hours to the current time. You can find the current time in PHP by using the **time()** function (it returns a timestamp; see Chapter 8). Therefore, this line of code sets the expiration time of the cookie to be

one hour (60 seconds times 60 minutes) from the current moment:

setcookie(name, value, time()+3600);

Because the expiration time will be calculated as the value of **time()** plus 3600, that argument isn't put in quotes (you don't want to literally pass **time() + 3600** as the expiration but rather the result of that calculation).

The **path** and **domain** arguments are used to limit a cookie to a specific folder in a Web site (the path) or to a specific domain. Using the **path** option, you could limit a cookie to exist only while a user is in a specific subfolder of the domain:

setcookie(name, value, time()+3600,
→ **'/subfolder/');**

Cookies are already specific to a domain, so the **domain** argument might be used to limit a cookie to a subdomain, such as **forum.example.com**.

setcookie(name, value, time()+3600, '',
→ **'forum.example.com');**

Script 9.3 When you add the **expiration** arguments to the two cookies, the cookies will persist even after the user has closed out of and later returned to their browser.

```
1  <?php // Script 9.3 - customize.php #2
2
3  // Handle the form if it has been submitted:
4  if (isset($_POST['font_size'], $_POST['font_color'])) {
5
6      // Send the cookies:
7      setcookie('font_size', $_POST['font_size'], time()+10000000, '/');
8      setcookie('font_color', $_POST['font_color'], time()+10000000, '/');
9
10     // Message to be printed later:
11     $msg = '<p>Your settings have been entered! Click <a href="view_settings.php">here</a> to see
        them in action.</p>';
12
13 } // End of submitted IF.
14 ?><!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
15     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
16 <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
```

code continues on next page

Script 9.3 *continued*

```
17 <head>
18   <meta http-equiv="Content-Type"
19     content="text/html; charset=utf-8"/>
20   <title>Customize Your Settings
21   </title>
22 </head>
23 <body>
24 <?php // If the cookies were sent, print
25   a message.
26   if (isset($msg)) {
27     print $msg;
28   }
29   ?>
30 <p>Use this form to set your
31   preferences:</p>
32 <form action="customize.php"
33   method="post">
34   <select name="font_size">
35     <option value="">Font Size</option>
36     <option value="xx-small">xx-small
37     </option>
38     <option value="x-small">x-small
39     </option>
40     <option value="small">small
41     </option>
42     <option value="medium">medium
43     </option>
44     <option value="large">large
45     </option>
46     <option value="x-large">x-large
47     </option>
48     <option value="xx-large">xx-large
49     </option>
50   </select>
51   <select name="font_color">
52     <option value="">Font Color</option>
53     <option value="999">Gray</option>
54     <option value="0c0">Green</option>
55     <option value="00f">Blue</option>
56     <option value="c00">Red</option>
57     <option value="000">Black</option>
58   </select>
59   <input type="submit" name="submit"
60     value="Set My Preferences" />
61 </form>
62 </body>
63 </html>
```

The **secure** value dictates that a cookie should only be sent over a secure HTTPS connection. A value of 1 indicates that a secure connection must be used, whereas 0 indicates that a secure connection isn't necessary. You could ensure a secure cookie transmission for e-commerce sites:

```
setcookie('cart', '82ABC3012',
→ time()+3600, ' ', 'shop.example.com', 1);
```

As with all functions that take arguments, you must pass all the values in order. In the preceding example, if there's no need to specify (or limit) the path, you use empty quotes. With the **path** argument, you can also use a single slash (/) to indicate the root folder (i.e., no path restriction). By doing so, you maintain the proper number of arguments and can still indicate that an HTTPS connection is necessary.

The final argument—**httponly**—was added in PHP 5.2. It can be used to restrict access to the cookie (for example, preventing a cookie from being read using JavaScript) but isn't supported by all browsers.

Let's add an expiration date to the existing **customize.php** page so that the user's preferences will remain even after they've closed their browser and then returned to the site later.

To set a cookie's expiration date:

1. Open **customize.php** (Script 9.1) in your text editor or IDE.
2. Change the two **setcookie()** lines to read as follows (Script 9.3):

```
setcookie('font_size', $_POST
→ ['font_size'], time()+10000000,
→ '/', '', 0);
setcookie('font_color', $_POST
→ ['font_color'], time()+10000000,
→ '/', '', 0);
```

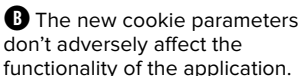
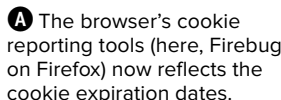
continues on next page

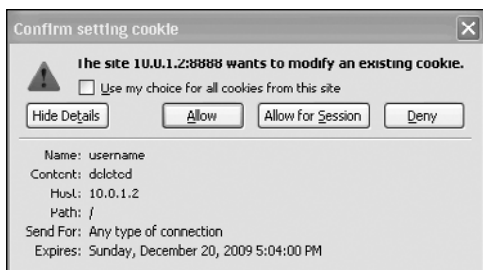
Because the expiration date of the cookies is set months into the future, the user's preferences, which are stored in the cookies, will be valid even after the user has closed and reopened the browser. Without this expiration date, users would see the default font size and color and have to reassign their preferences with every new browser session.

- A** and **B**.

TIP Here are some general guidelines for what kind of expiration date to use with your cookies: If the cookie should last as long as the user browses through the site, don't set an expiration time. If the cookie should continue to exist after the user has closed and reopened the browser, set an expiration time that's weeks or months in the future. And if the cookie can constitute a security risk, set an expiration time of an hour or a fraction thereof so that the cookie doesn't continue to exist too long after a user has left the browser.

the user's last action.





A How Firefox displays the cookie information when a deletion cookie is sent.

Deleting a Cookie

The final thing to know about cookies is how to delete them. Although a cookie automatically expires when the user's browser is closed or when the expiration date/time is met, sometimes you'll want to manually delete the cookie as well. For example, Web sites that have registered users and login capabilities generally delete any cookies when the user logs out.

The `setcookie()` function can take up to seven arguments, but only one is required—the name. If you send a cookie that consists of a name without a value, it will have the same effect as deleting the existing cookie of the same name. For example, to create the cookie `username`, you use this line:

```
setcookie('username', 'Larry');
```

To delete the `username` cookie, you code

```
setcookie('username', '');
```

or

```
setcookie('username', FALSE);
```

As an added precaution, you can also set an expiration date that's in the past **A**:

```
setcookie('username', FALSE,  
→ time() - 600);
```

The only caveat when it comes to deleting a cookie is that you must use the same argument values that were used to set the cookie in the first place (aside from the value and expiration). For example, if you set a cookie while providing a `domain` value, you must also provide that value when deleting the cookie:

```
setcookie('user', 'larry', time() +  
→ 3600, '', 'forums.example.com');  
setcookie('user', '', time() -  
→ 600, '', 'forums.example.com');
```

continues on next page

To demonstrate this feature, let's add a *reset* page to the Web application. This PHP script will destroy the sent cookies, so that the user's preferences are forgotten.

To delete a cookie:

1. Begin a new PHP script in your text editor or IDE, to be named **reset.php** (Script 9.4):

```
<?php // Script 9.4 - reset.php
```

2. Delete the existing cookies by sending blank cookies. Then complete the PHP code:

```
setcookie('font_size', '',
→ time() - 600, '/');
setcookie('font_color', '',
→ time() - 600, '/');
?>
```

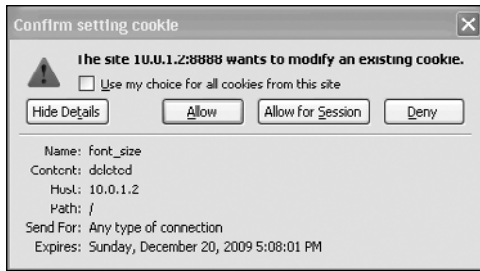
These two lines send cookies named *font_size* and *font_color*, each with no value and an expiration time of 10 minutes ago. As you did when creating cookies, you must call the **setcookie()** function before anything else is sent to the Web browser.

3. Create the HTML head:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD
→ XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/
    → DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/
→ 1999/xhtml" xml:lang="en"
→ lang="en">
<head>
    <meta http-equiv="Content-Type"
    → content="text/html;
    → charset=utf-8"/>
    <title>Reset Your Settings
    → </title>
</head>
```

Script 9.4 To delete the existing cookies, send new cookies with the same names, empty values, and expirations in the past.

```
1  <?php // Script 9.4 - reset.php
2
3  // Delete the cookies:
4  setcookie('font_size', '', time()
    - 600, '/');
5  setcookie('font_color', '', time()
    - 600, '/');
6
7  ?><!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
    1.0 Transitional//EN"
8      "http://www.w3.org/TR/xhtml1/DTD/
        xhtml1-transitional.dtd">
9  <html xmlns="http://www.w3.org/1999/
        xhtml" xml:lang="en" lang="en">
10 <head>
11     <meta http-equiv="Content-Type"
        content="text/html; charset=utf-8"/>
12     <title>Reset Your Settings</title>
13 </head>
14 <body>
15
16 <p>Your settings have been reset! Click
    <a href="view_settings.php">here</a> to
    go back to the main page.</p>
17
18 </body>
19 </html>
```



B When the `setcookie()` function is used with a name but no value, the existing cookie of that name is deleted. The expiration date in the past also guarantees proper destruction of the existing cookie.



C The reset page sends two blank cookies and then displays this message.



D After accessing the reset page, PHP destroys the cookies **B**, which will have the effect of resetting the `view_settings.php` page to its default formatting.

4. Add the page's body:

```
<body>
<p>Your settings have been reset!
→ Click <a href="view_settings.php">
→ here</a> to go back to the main
→ page.</p>
</body>
```

The body of this script merely tells users that their settings have been reset. A link is then provided to return to the main page.

5. Complete the HTML:

```
</html>
```

6. Save the page as `reset.php`, place it in the proper directory for your PHP-enabled server, and test it in your Web browser **B**, **C**, and **D**.

To test this page, either click the appropriate link in `view_settings.php` (Script 9.2) or just go to this page directly.

TIP Just as creating a cookie doesn't take effect until another page is loaded, deleting a cookie doesn't take effect until another page. This is to say that you can delete a cookie on a page but still access that cookie on it (because the cookie was received by the page before the delete cookie was sent).

TIP Just as creating cookies has mixed results using different browsers, the same applies to deleting them. Test your scripts on many browsers and play with the `setcookie()` settings to ensure the best all-around compatibility.

What Are Sessions?

A session, like a cookie, provides a way for you to track data for a user over a series of pages. The difference between the two—and this is significant—is that a cookie stores the data on the client (in the Web browser), whereas the session data is stored on the server. Because of this difference, sessions have numerous benefits over cookies:

- Sessions are generally more secure, because the data isn't transmitted back and forth between the client and server repeatedly.
- Sessions allow you to store more information than you can in a cookie.
- Sessions can be made to work even if the user doesn't accept cookies in their browser.

When you start a session, PHP generates a random session ID. Each user's session will have its own session ID, corresponding to the name of the text file on the server that stores the user's session data (**Script 9.5**). So that every PHP script on a site can associate the same session data with a particular user, the session ID must be tracked as well. By default, this session ID is sent to the Web browser as a cookie **A**. Subsequent PHP pages will use this cookie to retrieve the session ID and access the session information.

Over the next few pages, you'll see just how easy sessions are to work with in PHP.



A A session cookie being sent to the Web browser.

Script 9.5 How session data is stored in a file on the server.

```
1  email[s:14:"me@example.com";  
    loggedin|i:1292883103;
```

Choosing Between Sessions and Cookies

Sessions have many benefits over cookies, but there are still reasons why you would use the latter. Cookies have these advantages over sessions:


- Marginally easier to create and retrieve
- Require slightly less work from the server
- Normally persist over a longer period of time

As a rule of thumb, you should use cookies in situations where security is less of an issue and only a minimum of data is being stored. If security's a concern and there will be oodles of information to remember, you're best off with sessions. Understand, though, that using sessions may require a little more effort in writing your scripts.

Creating a Session

Creating, accessing, or deleting a session begins with the `session_start()` function. This function will attempt to send a cookie the first time a session is started, so it absolutely must be called prior to any HTML or white space being sent to the Web browser. Therefore, on pages that use sessions, you should call the `session_start()` function as one of the very first lines in your script:

```
<?php
session_start();
```

The first time a session is started, a random session ID is generated and a cookie is sent to the Web browser with a name of **PHPSESSID** (the session name) and a value like **4bcc48dc87cb4b54d63f99da23fb41e1** (see  in the previous section).

Once the session has been started, you can record data to it by assigning values to the `$_SESSION` array:

```
$_SESSION['first_name'] = 'Sam';
$_SESSION['age'] = 4;
```

Unlike with other arrays you might use in PHP, you should always treat this array as an associative array. In other words, you should explicitly use strings for the keys, such as *first_name* and *age*.

Each time a value is assigned to the `$_SESSION` array, PHP writes that data to a temporary file stored on the server (see Script 9.5).

To begin, you'll rewrite the login script from Chapter 8, this time storing the email address in a session.

To create a session:

1. Open **login.php** (Script 8.13) in your text editor or IDE.
2. Before the **ob_end_clean()** line, add the following (Script 9.6):

```
session_start();
$_SESSION['email'] = $_POST['email'];
$_SESSION['loggedin'] = time();
```

To store values in a session, begin by calling the **session_start()** function. Although you normally have to call this function first thing in a script (because it may attempt to send a cookie), that's not required here because the header file for this script begins output buffering (see Chapter 8).

The session first stores the user's submitted email address in **\$_SESSION['email']**. Then the timestamp of when the user logged in is assigned to **\$_SESSION['loggedin']**. This value is determined by calling the **time()** function, which returns the number of seconds that have elapsed since the *epoch* (midnight on January 1, 1970).

3. Save the file as **login.php** and place it in the appropriate directory on your PHP-enabled computer.

This script should be placed in the same directory used in Chapter 8, as it requires some of those other files.

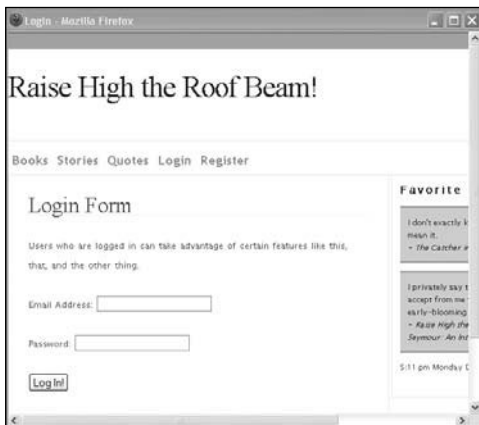
Script 9.6 This script stores two values in the session and then redirects the user to another page, where the session values can be accessed.

```
1  <?php // Script 9.6 - login.php #3
2  /* This page lets people log into the
   site (almost!). */
3
4  // Set the page title and include the
   header file:
5  define('TITLE', 'Login');
6  include('templates/header.html');
7
8  // Print some introductory text:
9  print '<h2>Login Form</h2>
10     <p>Users who are logged in can
       take advantage of certain features
       like this, that, and the other
       thing.</p>';
11
12 // Check if the form has been
   submitted:
13 if ($_SERVER['REQUEST_METHOD'] == 'POST') {
14
15     // Handle the form:
16     if ( !empty($_POST['email'])) &&
        (!empty($_POST['password'])) ) {
17
18         if ( strtolower($_POST
19            ['email']) == 'me@example.com')
20             && ($_POST['password'] ==
21                'testpass') ) { // Correct!
22
23             // Do session stuff:
24             session_start();
25             $_SESSION['email'] =
26                 $_POST['email'];
27             $_SESSION['loggedin'] =
28                 time();
29
30             // Redirect the user to the
31             welcome page!
32             ob_end_clean();
33             // Destroy the buffer!
34             header ('Location: welcome.php');
35             exit();
36
37         } else { // Incorrect!
38
39             print '<p>The submitted email
40                address and password do not
41                match those on file!<br />Go
42                back and try again.</p>';
43         }
44     }
45 }
```

code continues on next page

Script 9.6 *continued*

```
33
34     }
35
36     } else { // Forgot a field.
37
38         print '<p>Please make sure you
           enter both an email address and
           a password!<br />Go back and try
           again.</p>';
39
40     }
41
42 } else { // Display the form.
43
44     print '<form action="login.php"
           method="post">
45     <p>Email Address: <input type="text"
           name="email" size="20" /></p>
46     <p>Password: <input type="password"
           name="password" size="20" /></p>
47     <p><input type="submit" name=
           "submit" value="Log In!" /></p>
48     </form>';
49
50 }
51
52 include('templates/footer.html');
53 // Need the footer.
54 ?>
```



A The login form.

4. Load the form in your Web browser to ensure that it has no errors **A**.

Don't complete and submit the login form yet, as the welcome page needs to be updated prior to logging in.

TIP The `php.ini` configuration file includes many session-related settings that you can tinker with if you have administrative-level control over your server. Open the `php.ini` file in a text editor and see the manual for more information.

TIP You can also alter some of the session settings using the `ini_set()` function.

TIP The `session_name()` function lets you change the name of the session (instead of using the default `PHPSESSID`). It must be used before every `session_start()` call, like so:

```
session_name('YourVisit');
session_start();
```

TIP The `session_set_cookie_params()` function alters the session cookie settings, like the expiration time, the path, and the domain.

TIP The constant `SID`, short for *Session ID*, stores a string in the format `name=ID`. An example is `PHPSESSID=4bcc48dc87cb4b54d63f99da23fb41e1`.

TIP You can store any type of value—number, string, array, or object—or any combination thereof in your sessions.

Accessing Session Variables

Now that you've stored values in a session, you need to know how to access them. The first step is to invoke the **session_start()** function. This is necessary on every page that will make use of sessions, whether it's creating a new session or accessing an existing one.

From there it's simply a matter of referencing the **\$_SESSION** variable as you would any other array. With this in mind, you'll write another welcome page—similar to the one from Chapter 8—that accesses the stored *email* and *loggedin* values.

To access session variables:

1. Create a new PHP document in your text editor or IDE, to be named **welcome.php** (Script 9.7):

```
<?php // Script 9.7 - welcome.php
```

2. Begin the session:

```
session_start();
```

Even when you're accessing session values, you should call the **session_start()** function before any data is sent to the Web browser.

3. Define a page title, and include the HTML header:

```
define('TITLE', 'Welcome to the  
→ J.D. Salinger Fan Club!');  
include('templates/header.html');
```

Because this page uses the same template system developed in Chapter 8, it also uses the same header system.

Script 9.7 You can access stored session values using the **\$_SESSION** array, as long as your script uses **session_start()** first.

```
1 <?php // Script 9.7 - welcome.php #2  
2 /* This is the welcome page. The user is  
   redirected here  
3 after they successfully log in. */  
4  
5 // Need the session:  
6 session_start();  
7  
8 // Set the page title and include the  
   header file:  
9 define('TITLE', 'Welcome to the J.D.  
   Salinger Fan Club!');  
10 include('templates/header.html');  
11  
12 // Print a greeting:  
13 print '<h2>Welcome to the J.D.  
   Salinger Fan Club!</h2>';  
14 print '<p>Hello, ' . $_SESSION  
   ['email'] . '!</p>';  
15  
16 // Print how long they've been  
   logged in:  
17 date_default_timezone_set('America/  
   New_York');  
18 print '<p>You have been logged  
   in since: ' . date('g:i a',  
   $_SESSION['loggedin']) . '</p>';  
19  
20 // Make a logout link:  
21 print '<p><a href="logout.php">Click here  
   to logout.</a></p>';  
22  
23 include('templates/footer.html');  
   // Need the footer.  
24 ?>
```

4. Greet the user by email address:

```
print '<h2>Welcome to the J.D.
→ Salinger Fan Club!</h2>';
print '<p>Hello, ' . $_SESSION
→ ['email'] . '!</p>';
```

To access the stored user's address, refer to `$_SESSION['email']`. Here, that value is concatenated to the rest of the string that's being printed out.

5. Show how long the user has been logged in:

```
date_default_timezone_set
→ ('America/New_York');
print '<p>You have been logged
→ in since: ' . date('g:i a',
→ $_SESSION['loggedin']) . '</p>';
```

To show how long the user has been logged in, refer to the `$_SESSION['loggedin']` variable. By using this as the second argument sent to the `date()` function, along with the appropriate formatting parameters, you make the PHP script create text like *11:22 pm*.

Before using the `date()` function, however, you need to set the default time zone (this is also discussed in Chapter 8). If you want, after setting the time zone here, you can remove the use of the same function from the footer file.

6. Complete the content:

```
print '<p><a href="logout.php">
→ Click here to logout.</a></p>';
```

The next script will provide logout functionality, so a link to it is added here.

7. Include the HTML footer, and complete the HTML page:

```
include('templates/footer.html');
?>
```

8. Save the file as `welcome.php`, place it in the proper directory for your PHP-enabled server, and test it (starting with `login.php`, Script 9.6) in your Web browser **A**.

TIP To see whether a particular session variable exists, use `isset($_SESSION['var'])` as you would to check if any other variable is set.

TIP Always remember that the data stored in a session is being stored as plain text in an openly readable text file. Don't be cavalier about what gets stored in a session and never store really sensitive information, such as credit card data, there.

TIP For added security, data can be encrypted prior to storing it in a session and decrypted upon retrieval. Doing so requires the Mcrypt library and more advanced PHP knowledge, however.



A After successfully logging in (using *me@example.com* and *testpass* in the form), the user is redirected to this page, which greets them using the session values.

Deleting a Session

It's important to know how to delete a session, just as it's important to know how to delete a cookie: Eventually you'll want to get rid of the data you've stored. Session data exists in two places—in an array during the execution of the script and in a text file, so you'll need to delete both. But first you must begin with the **session_start()** function, as always:

```
session_start();
```

Then, you clear the session variables by resetting the **\$_SESSION** array:

```
$_SESSION = array();
```

Finally, remove the session data from the server (where it's stored in temporary files). To do this, use

```
session_destroy();
```

With that in mind, let's write **logout.php**, which will delete the session, effectively logging out the user.

To delete a session:

1. Start a new PHP script in your text editor or IDE, to be named **logout.php** (Script 9.8).

```
<?php // Script 9.8 - logout.php
```

2. Begin the session:

```
session_start();
```

Remember that you can't delete a session until you activate the session using this function.

3. Reset the session array:

```
$_SESSION = array();
```

As explained in Chapter 7, "Using Arrays," the **array()** function creates a new, empty array. By assigning the result of this function call to **\$_SESSION**, all the existing *key-value* pairs in **\$_SESSION** will be erased.

Script 9.8 Deleting a session is a three-step process: start the session, reset the array, and destroy the session data.

```
1 <?php // Script 9.8 - logout.php
2 /* This is the logout page. It
   destroys the session information. */
3
4 // Need the session:
5 session_start();
6
7 // Delete the session variable:
8 unset($_SESSION);
9
10 // Reset the session array:
11 $_SESSION = array();
12
13 // Define a page title and include the
   header:
14 define('TITLE', 'Logout');
15 include('templates/header.html');
16
17 ?>
18
19 <h2>Welcome to the J.D. Salinger Fan
   Club!</h2>
20 <p>You are now logged out.</p>
21 <p>Thank you for using this site. We
   hope that you liked it.<br />
22 Blah, blah, blah...
23 Blah, blah, blah...</p>
24
25 <?php include('templates/
   footer.html'); ?>
```



A The logout page destroys the session data.

4. Destroy the session data on the server:

```
session_destroy();
```

This step tells PHP to remove the actual session file on the server.

5. Include the HTML header, and complete this PHP section:

```
define('TITLE', 'Logout');  
include('templates/header.html');  
?>
```

6. Make the page content:

```
<h2>Welcome to the J.D. Salinger  
Fan Club!</h2>  
<p>You are now logged out.</p>  
<p>Thank you for using this site.  
We hope that you liked it.<br />  
Blah, blah, blah...  
Blah, blah, blah...</p>
```

7. Include the HTML footer:

```
<?php include('templates/footer.  
html'); ?>
```

8. Save the file as **logout.php**, place it in the proper directory for your PHP-enabled server, and test it in your Web browser by clicking the link in **welcome.php** **A**.

TIP To delete an individual session value, use:

```
unset($_SESSION['var']);
```

TIP The PHP module on the server will automatically perform *garbage collection* based on settings in its configuration. PHP uses garbage collection to manually delete session files from the server, with the assumption that they're no longer needed.

TIP You can have PHP use sessions without cookies, in which case the session ID must be appended to every link in your site (so that each page receives the session ID). If you enable the `use_trans_sid` setting (in the `php.ini` file), PHP will handle this for you.

Review and Pursue

If you have any problems with the review questions or the pursue prompts, turn to the book's supporting forum (www.LarryUllman.com/forum/).

Review

- Where does a cookie store data? Where does a session store data? Which is generally more secure?
- Name two debugging techniques when trying to solve issues involving cookies.
- How do the **path** and **domain** arguments to the **setcookie()** function affect the accessibility of the cookie?
- How do you delete a cookie?
- What function must every page call if it needs to assign or access session data?
- Why do sessions also use cookies (by default)?

Pursue

- Install the Firebug extension for Firefox, if you have not already. Install Firefox first, if you haven't!
- Look up the PHP manual page for the **setcookie()** function. Review the information and user comments there for added instructions on cookies.
- Rewrite **customize.php** so that the script also applies the user's preferences. Hint: You need to take into account the fact that the cookies aren't available immediately after they've been set. Instead, you would write the CSS code using the **\$_GET** values after the form has been submitted, the **\$_COOKIE** values upon first arriving at the page (if the cookies exist), and the default values otherwise.
- Make the form in **customize.php** *sticky*, so that it reflects the user's current choices.
- Rewrite **welcome.php** so that the **print** statement that greets the user by email address uses double quotation marks.
- For an added challenge, rewrite **welcome.php** so that the **print** statement that indicates how long the user has been logged in also uses double quotation marks. Hint: You'll need to use a variable.
- Rewrite the last three scripts so that the session uses a custom name.