FULLY UPDATED AND EXPANDED BESTSELLER

VISUAL QUICKPRO GUIDE

LARRY ULLMAN

PHP AND MySQL FOR DYNAMIC WEB SITES

Second Edition

Learn PHP and MySQL the fast, efficient way! This Visual QuickPro Guide uses

Covers
- PHP 5 and
- MySQL 4.1

Sample Chapter

Save 35% on this title with coupon code:
EE-FLY3-MM22
www.peachpit.com/title/0321336577
offer expires 12/31/06

9

Cookies and Sessions

The Hypertext Transfer Protocol (HTTP) is a stateless technology, meaning that each individual HTML page is an unrelated entity. HTTP has no method for tracking users or retaining variables as a person traverses a site. Using a Web scripting language like PHP, you can overcome the statelessness of the Web. You have a few options to choose from, the most popular two being cookies and sessions.

Prior to the existence of cookies, surfing a Web site was a trip without a history. Although your browser tracked the pages you visited, allowing you to use the back button to return to previously visited pages, the server kept no record of who had seen what. Without the server being able to track a user, there can be no shopping carts or custom Web site personalization.

Sessions improve upon cookies, allowing the Web application to store and retrieve far more information than cookies alone can. Both technologies are easy to use with PHP and are worth knowing. In this chapter I'll explain each, using a login system, based upon the existing *users* database, as my example.

Using Cookies

Cookies are a way for a server to store information on the user's machine. This is one way that a site can remember or track a user over the course of a visit. Think of a cookie like a name tag: you tell the server your name and it gives you a sticker to wear. Then it can know who you are by referring back to that name tag.

Some people are suspicious of cookies because they believe that cookies allow a server to know too much about them. However, a cookie can only be used to store information that the server is given, so it's no less secure than most anything else online. Unfortunately, many people still have misconceptions about the technology, which is a problem as those misconceptions can undermine the functionality of your Web application.

In this section you will learn how to set a cookie, retrieve information from a stored cookie, alter a cookie's settings, and then delete a cookie.

Testing for Cookies

To effectively program using cookies, you need to be able to accurately test for their presence. The best way to do so is to have your Web browser ask what to do when receiving a cookie. In such a case, the browser will prompt you with the cookie information each time PHP attempts to send a cookie.

Different versions of different browsers on different platforms all define their cookie handling policies in different places. I'll quickly run through a couple of options for popular Web browsers.

To set this up using Internet Explorer on Windows XP, choose Tools > Internet Options. Then click the Privacy tab, followed by the Advanced button under Settings. Click "Override automatic cookie handling" and then choose "Prompt" for both First- and Third-party Cookies.

Using Firefox on Windows, choose Tools > Options. Then click Privacy and expand the Cookies section. Finally, select "ask me every time" in the Keep Cookies drop-down menu. If you are using Firefox on Mac OS X, the steps are the same, but you must start by choosing Firefox > Preferences.

Unfortunately, Safari on Mac OS X does not have a cookie prompting option, but it will allow you to view existing cookies, which is still a useful debugging tool. This option can be found under the Security Preferences panel.

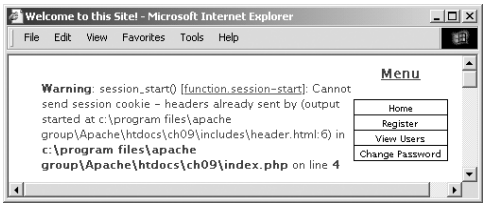

Figure 9.1 The *headers already sent...* error message is all too common when creating cookies. Pay attention to what the error message says in order to find and fix the problem.

Figure 9.2 If I have my browser set to ask for permission when receiving cookies, I'll see a message like this when a site attempts to send one.

Setting cookies

The most important thing to understand about cookies is that they must be sent from the server to the client prior to *any other information*. 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 will not be sent (**Figure 9.1**). This is by far the most common cookie-related error.

Cookies are sent via the `setcookie()` function:

```
setcookie (name, value);
```

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

The second line of code will send a cookie to the browser with a name of *first_name* and a value of *Larry* (**Figure 9.2**).

You can continue to send more cookies to the browser with subsequent uses of the `setcookie()` function:

```
setcookie ('ID', 263);
```

```
setcookie ('email', 'phpmysql2@
→ dmcinsights.com');
```

As when using any variable in PHP, when naming your cookies, do not use white spaces or punctuation, but do pay attention to the exact case used.

To send a cookie:

1. Create a new PHP document in your text editor (**Script 9.1**).

```
<?php # Script 9.1 - login.php
```

For this example, I'll make a new `login.php` script (which works in conjunction with the scripts from Chapter 7, "Using PHP with MySQL").

2. Validate the form.

```
if (isset($_POST['submitted'])) {
  require_once ('../mysql_connect.
  → php');
  $errors = array();
  if (empty($_POST['email'])) {
    $errors[] = 'You forgot to enter
    → your email address.';
  } else {
    $e = escape_data($_POST
    → ['email']);
  }
  if (empty($_POST['password'])) {
    $errors[] = 'You forgot to enter
    → your password.';
  } else {
    $p = escape_data($_POST
    → ['password']);
  }
```

These steps are very similar to those in Chapter 7. The main conditional checks if the form has been submitted. Then the database connection is made by including the connection script (which also defines the `escape_data()` function as of Chapter 8, "Web Application Development"). Finally, the email address and password inputs are checked for values.

3. Retrieve the `user_id` and `first_name` for this user from the database.

```
if (empty($errors)) {
  $query = "SELECT user_id,
  → first_name FROM users WHERE
  → email='$e' AND password=
  → SHA('$p')";
  $result = @mysql_query ($query);
  $row = mysql_fetch_array
  → ($result, MYSQL_NUM);
```

If both validation tests were passed, the database will be queried, retrieving the `user_id` and `first_name` values for the record where the `email` column matches the submitted email address and the password matches an encrypted version of the submitted password.

4. If the user entered the correct information, log the user in.

```
if ($row) {
  setcookie ('user_id', $row[0]);
  setcookie ('first_name', $row[1]);
```

The `$row` variable will have a value only if the preceding query returned at least one record (indicating the submitted email address and password match those on file). In this case, two cookies will be created.

continues on page 328

Script 9.1 The `login.php` script creates cookies upon a successful login.

script

```
1   <?php # Script 9.1 - login.php
2   // Send NOTHING to the Web browser prior to the setcookie() lines!
3
4   // Check if the form has been submitted.
5   if (isset($_POST['submitted'])) {
6
7      require_once ('../mysql_connect.php'); // Connect to the db.
8
9      $errors = array(); // Initialize error array.
10
11     // Check for an email address.
12     if (empty($_POST['email'])) {
13        $errors[] = 'You forgot to enter your email address.';
14     } else {
15        $e = escape_data($_POST['email']);
16     }
17
18     // Check for a password.
19     if (empty($_POST['password'])) {
20        $errors[] = 'You forgot to enter your password.';
21     } else {
22        $p = escape_data($_POST['password']);
23     }
24
25     if (empty($errors)) { // If everything's OK.
26
27        /* Retrieve the user_id and first_name for
28        that email/password combination. */
29        $query = "SELECT user_id, first_name FROM users WHERE email='$e' AND password=SHA('$p')";
30        $result = @mysql_query ($query); // Run the query.
31        $row = mysql_fetch_array ($result, MYSQL_NUM); // Return a record, if applicable.
32
33        if ($row) { // A record was pulled from the database.
34
35           // Set the cookies & redirect.
36           setcookie ('user_id', $row[0]);
37           setcookie ('first_name', $row[1]);
38
```

(script continues on page 333)

5. Redirect the user to another page.

```
$url = 'http://' . $_SERVER
→['HTTP_HOST'] . dirname($_SERVER
→['PHP_SELF']);
if ((substr($url, -1) == '/') OR
→(substr($url, -1) == '\\') ) {
  $url = substr ($url, 0, -1);
}
$url .= '/loggedin.php';
header("Location: $url");
exit();
```

Using the steps outlined in Chapter 8, the redirection URL is first dynamically generated. To do so, various `$_SERVER` values are referenced, along with the `dirname()` function. Any trailing slashes are also chopped off should this script be within a subdirectory (this is all covered in Chapter 8).

Finally the `header()` function is called to redirect the user and the script's execution is terminated with `exit()`.

6. Complete the `$row` conditional (started in Step 4) and the `$errors` conditional, and then close the database connection.

```
  } else {
    $errors[] = 'The email address
    →and password entered do not
    →match those on file.';
    $errors[] = mysql_error() .
    →'<br /><br />Query: ' .
$query;
  }
}
mysql_close();
```

The error management in this script is much like that in the `register.php` script in Chapter 8. Because nothing can be sent to the Web browser before calling the `setcookie()` and `header()` lines, the errors have to be saved and printed later. The second error message here is for debugging purposes only and shouldn't be used on a live site.

7. Complete the main submit conditional, include the HTML header, and print any error messages.

```
} else {
  $errors = NULL;
}
$page_title = 'Login';
include ('./includes/header.html');
if (!empty($errors)) {
  echo '<h1 id="mainhead">Error!
  →</h1>
  <p class="error">The following
  →error(s) occurred:<br />';
  foreach ($errors as $msg) {
  →echo " - $msg<br />\n";
  }
  echo '</p><p>Please try again.
  →</p>';
}
?>
```

Again, this and the previous steps are like those in Chapter 8's `register.php` script. The first `else` conditional sets the `$errors` variable to `NULL`, indicating that no errors need to be printed out when this page is first run. Then the page's title is set and the template's header file is included (this application uses the same template as those in Chapters 7 and 8). Finally, any existing errors—from the form's submission—are printed.

continues on page 330

Script 9.1 *continued*

script

```
39            // Redirect the user to the loggedin.php page.
40            // Start defining the URL.
41            $url = 'http://' . $_SERVER['HTTP_HOST'] . dirname($_SERVER['PHP_SELF']);
42            // Check for a trailing slash.
43            if ((substr($url, -1) == '/') OR (substr($url, -1) == '\\') ) {
44               $url = substr ($url, 0, -1); // Chop off the slash.
45            }
46            // Add the page.
47            $url .= '/loggedin.php';
48
49            header("Location: $url");
50            exit(); // Quit the script.
51
52         } else { // No record matched the query.
53            $errors[] = 'The email address and password entered do not match those on file.';
              // Public message.
54            $errors[] = mysql_error() . '<br /><br />Query: ' . $query; // Debugging message.
55         }
56
57      } // End of if (empty($errors)) IF.
58
59      mysql_close(); // Close the database connection.
60
61   } else { // Form has not been submitted.
62
63      $errors = NULL;
64
65   } // End of the main Submit conditional.
66
67   // Begin the page now.
68   $page_title = 'Login';
69   include ('./includes/header.html');
70
71   if (!empty($errors)) { // Print any error messages.
72      echo '<h1 id="mainhead">Error!</h1>
73      <p class="error">The following error(s) occurred:<br />';
74      foreach ($errors as $msg) { // Print each error.
75         echo " - $msg<br />\n";
76      }
77      echo '</p><p>Please try again.</p>';
78   }
79
```

(script continues on next page)

8. Display the HTML form.

```
<h2>Login</h2>
<form action="login.php"
→ method="post">
  <p>Email Address: <input type=
  → "text" name="email" size="20"
  → maxlength="40" /> </p>
  <p>Password: <input type=
  → "password" name="password"
  → size="20" maxlength="20" /></p>
  <p><input type="submit" name=
  → "submit" value="Login" /></p>
  <input type="hidden" name=
  → "submitted" value="TRUE" />
</form>
```

The HTML form takes two inputs—an email address and a password—and submits the data back to this same page. You can make the email address input sticky by presetting a value attribute, if you'd like.

9. Include the PHP footer.

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

10. Save the file as `login.php`, upload it to your Web server in the same directory as the files from Chapter 7, and load the form in your Web browser (**Figure 9.3**).

✔ Tips

- Cookies are limited to about 4 KB of total data, and each Web browser can remember only 20 cookies from any one server.
- Because cookies rely upon the HTTP header, you can set them in PHP using the `header()` function. It's very important to remember that the `setcookie()` and `header()` functions must be called before any data is sent to the Web browser.

Script 9.1 *continued*

script

```
80  // Create the form.
81  ?>
82  <h2>Login</h2>
83  <form action="login.php" method="post">
84     <p>Email Address: <input type="text"
       name="email" size="20" maxlength="40"
       /> </p>
85     <p>Password: <input type="password"
       name="password" size="20" maxlength="20"
       /></p>
86     <p><input type="submit" name="submit"
       value="Login" /></p>
87     <input type="hidden" name="submitted"
       value="TRUE" />
88  </form>
89  <?php
90  include ('./includes/footer.html');
91  ?>
```

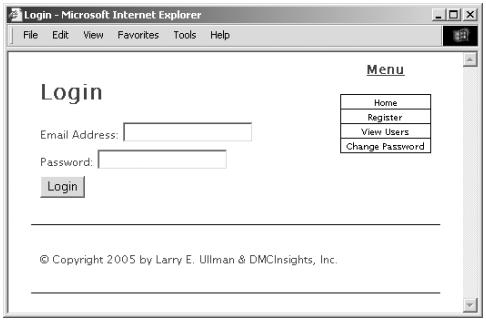

Figure 9.3 The login form.

- The `setcookie()` function is one of the few functions in PHP that could generate different results in different browsers, since browsers will treat cookies differently. Be sure to test your Web sites in multiple browsers on different platforms to ensure consistency.
- In Chapter 11, "Extended Topics," I'll show how to control browser output so that cookies can be sent at nearly any point in a script.

Script 9.2 The `loggedin.php` script prints a greeting to a user based upon a stored cookie.

script

```
1   <?php # Script 9.2 - loggedin.php
2   # User is redirected here from login.php.
3
4   // If no cookie is present, redirect the
    user.
5   if (!isset($_COOKIE['user_id'])) {
6
7       // Start defining the URL.
8       $url = 'http://' . $_SERVER['HTTP_HOST']
        . dirname($_SERVER['PHP_SELF']);
9       // Check for a trailing slash.
10      if ((substr($url, -1) == '/') OR
        (substr($url, -1) == '\\') ) {
11          $url = substr ($url, 0, -1); // Chop
            off the slash.
12      }
13      $url .= '/index.php'; // Add the page.
14      header("Location: $url");
15      exit(); // Quit the script.
16  }
17
18  // Set the page title and include the HTML
    header.
19  $page_title = 'Logged In!';
20  include ('./includes/header.html');
21
22  // Print a customized message.
23  echo "<h1>Logged In!</h1>
24  <p>You are now logged in, {$_COOKIE
    ['first_name']}!</p>
25  <p><br /><br /></p>";
26
27  include ('./includes/footer.html');
28  ?>
```

Accessing cookies

To retrieve a value from a cookie, you only need to refer to the `$_COOKIE` superglobal, using the appropriate cookie name as the key (as you would with any array). For example, to retrieve the value of the cookie established with the line

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

you would use `$_COOKIE['username']`.

In the following example, the cookies set by the `login.php` script will be accessed in two ways. First a check will be made that the user is logged in (otherwise, they shouldn't be accessing this page). Next, the user will be greeted by their first name, which was stored in a cookie.

To access a cookie:

1. Create a new PHP document in your text editor (**Script 9.2**).

   ```
   <?php # Script 9.2 - loggedin.php
   ```

   The user will be redirected to this page after successfully logging in. It will print a user-specific greeting.

2. Check for the presence of a cookie.

   ```
   if (!isset($_COOKIE['user_id'])) {
   ```

   Since I don't want a user to access this page unless that user is logged in, I first check for the cookie that should have been set (in `login.php`).

continues on next page

3. Complete the `if` conditional.

```
$url = 'http://' . $_SERVER
→['HTTP_HOST'] . dirname($_SERVER
→['PHP_SELF']);
if ((substr($url, -1) == '/') OR
→(substr($url, -1) == '\\') ) {
   $url = substr ($url, 0, -1);
}
$url .= '/index.php';
header("Location: $url");
exit();
}
```

   If the user is not logged in, they will be automatically redirected to the main page. This is a simple way to limit access to logged-in users.

4. Include the page header.

```
$page_title = 'Logged In!';
include ('./includes/header.html');
```

5. Welcome the user, using the cookie.

```
echo "<h1>Logged In!</h1>
<p>You are now logged in, {$_COOKIE
→['first_name']}!</p>
<p><br /><br /></p>";
```

   To greet the user by name, I refer to the `$_COOKIE['first_name']` variable (enclosed within curly braces to avoid parse errors).

6. Complete the HTML page.

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

7. Save the file as `loggedin.php`, upload to your Web server (in the same directory as `login.php`), and test in your Web browser by logging in through `login.php` (**Figure 9.4**).

   Since these examples use the same database as those in Chapter 7, you should be able to log in using the registered username and password submitted at that time.

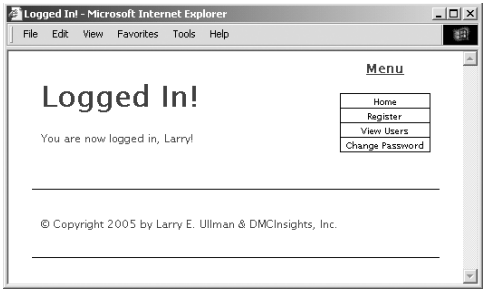

Figure 9.4 If you used the correct username and password, you'll be redirected here after logging in.

Figure 9.5 To see the effect of the `setcookie()` function, set your Web browser to ask before storing a cookie.

Figure 9.6 The `user_id` cookie with a value of *1*.

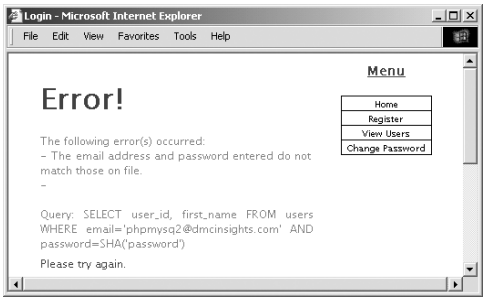

Figure 9.7 If no record was returned from the database, this will be the result. The blank space after the second dash is for the `mysql_error()`, which doesn't exist (since the query ran fine).

8. If you like, change the cookie settings for your browser (**Figure 9.5**) and test again (**Figure 9.6**).

✔ Tips

- If the submitted email address and username do not match those on file, a public message is displayed, followed by the query (**Figure 9.7**). Remember to delete the debugging message (the MySQL error plus the query) before using this code on a live site.
- If your `mysql_connect.php` file sends anything to the Web browser or even has blank lines or spaces after the closing PHP tag, you'll see a *headers already sent* error.
- With *register_globals* enabled, PHP will load variables in a specific order (depending upon the setting in the `php.ini` file), normally: get, post, cookie, session. If you do not use the superglobal arrays to refer to variables, then the value of a `$username` variable in a form could be overridden by the value of a `$username` variable stored in a cookie. This is one reason why you should program without relying upon *register_globals*.
- A cookie is not accessible until the setting page (e.g., `login.php`) has been reloaded or another page has been accessed (in other words, you cannot set and access a cookie in the same page).
- If users decline a cookie or have their Web browser set not to accept them, they will automatically be redirected to the home page in this example, even if they successfully logged in. For this reason you may want to let the user know when cookies are required.

Setting cookie parameters

Although passing just the name and value arguments to the `setcookie()` function will suffice, you ought to be aware of the other arguments available. The function can take up to four more parameters, each of which will alter the definition of the cookie.

```
setcookie ('name', 'value', expiration,
→ 'path', 'domain', secure);
```

The expiration argument is used to set a definitive length of time for a cookie to exist, specified in seconds since the *epoch* (the epoch is midnight on January 1, 1970). If it is not set, the cookie will continue to be functional until the user closes his or her browser. Normally, the expiration time is determined by adding a particular number of minutes or hours to the current moment, retrieved using the `time()` function. The following line will set the expiration time of the cookie to be 1 hour (60 seconds times 60 minutes) from the current moment:

```
setcookie ('name', 'value', time()+
→ 3600);
```

The path and domain arguments are used to limit a cookie to a specific folder within a Web site (the path) or to a specific host. For example, you could restrict a cookie to exist only while a user is within the *admin* folder of a domain (and the *admin* folder's subfolders):

```
setcookie ('name', 'value', time()+
→ 3600, '/admin/');
```

Finally, the secure value dictates that a cookie should only be sent over a secure HTTPS connection. A *1* indicates that a secure connection must be used, and a *0* says that a standard connection is fine.

```
setcookie ('name', 'value', time()+
→ 3600, '/admin/', '', 1);
```

As with all functions that take arguments, you must pass the `setcookie()` values in order. To skip any parameter, use `NULL` or an empty string. The expiration and secure values are both integers and are therefore not quoted.

To demonstrate this information, I'll add an expiration setting to the login cookies so that they last for only one hour.

Script 9.3 The `login.php` script now uses every argument the `setcookie()` function can take.

script

```
1   <?php # Script 9.3 - login.php (2nd version after Script 9.1)
2   // Send NOTHING to the Web browser prior to the setcookie() lines!
3
4   // Check if the form has been submitted.
5   if (isset($_POST['submitted'])) {
6
7       require_once ('../mysql_connect.php');
        // Connect to the db.
8
9       $errors = array(); // Initialize error array.
10
11      // Check for an email address.
12      if (empty($_POST['email'])) {
13          $errors[] = 'You forgot to enter your email address.';
14      } else {
15          $e = escape_data($_POST['email']);
16      }
17
18      // Check for a password.
19      if (empty($_POST['password'])) {
20          $errors[] = 'You forgot to enter your password.';
21      } else {
```

(script continues on next page)

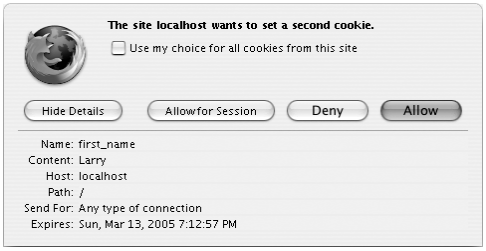

Figure 9.8 Once an expiration date or time has been set, it will be reflected in the cookie sent to the Web browser.

To set a cookie's expiration date:

1. Open `login.php` in your text editor (refer to Script 9.1).
2. Change the two `setcookie()` lines to include an expiration date that's 60 minutes away (**Script 9.3**):

   ```
   setcookie ('user_id', $row[0], →time()+3600, '/', '', 0);
   setcookie ('first_name', $row[1], →time()+3600, '/', '', 0);
   ```

   With the expiration date set to `time() + 3600` (60 minutes times 60 seconds), the cookie will continue to exist for an hour after it is set. While I'm at it, I explicitly state the other cookie parameters.
3. Save the script, upload to your Web server, and test in your Web browser (**Figure 9.8**).

Script 9.3 *continued*

script

```
22          $p = escape_data($_POST['password']);
23      }
24
25      if (empty($errors)) { // If everything's OK.
26
27          /* Retrieve the user_id and first_name for
28          that email/password combination. */
29          $query = "SELECT user_id, first_name FROM users WHERE email='$e' AND password=SHA('$p')";
30          $result = @mysql_query ($query); // Run the query.
31          $row = mysql_fetch_array ($result, MYSQL_NUM); // Return a record, if applicable.
32
33          if ($row) { // A record was pulled from the database.
34
35              // Set the cookies & redirect.
36              setcookie ('user_id', $row[0], time()+3600, '/', '', 0);
37              setcookie ('first_name', $row[1], time()+3600, '/', '', 0);
38
39              // Redirect the user to the loggedin.php page.
40              // Start defining the URL.
41              $url = 'http://' . $_SERVER['HTTP_HOST'] . dirname($_SERVER['PHP_SELF']);
42              // Check for a trailing slash.
43              if ((substr($url, -1) == '/') OR (substr($url, -1) == '\\') ) {
44                  $url = substr ($url, 0, -1); // Chop off the slash.
45              }
46              // Add the page.
47              $url .= '/loggedin.php';
48
49              header("Location: $url");
50              exit(); // Quit the script.
51
52          } else { // No record matched the query.
53              $errors[] = 'The email address and password entered do not match those on file.';
                // Public message.
54              $errors[] = mysql_error() . '<br /><br />Query: ' . $query; // Debugging message.
55          }
56
57      } // End of if (empty($errors)) IF.
58
59      mysql_close(); // Close the database connection.
60
```

(script continues on next page)

Script 9.3 *continued*

script

```
} else { // Form has not been submitted.

    $errors = NULL;

} // End of the main Submit conditional.

// Begin the page now.
$page_title = 'Login';
include ('./includes/header.html');

if (!empty($errors)) { // Print any error messages.
    echo '<h1 id="mainhead">Error!</h1>
    <p class="error">The following error(s) occurred:<br />';
    foreach ($errors as $msg) { // Print each error.
        echo " - $msg<br />\n";
    }
    echo '</p><p>Please try again.</p>';
}

// Create the form.
?>
<h2>Login</h2>
<form action="login.php" method="post">
    <p>Email Address: <input type="text" name="email" size="20" maxlength="40" /> </p>
    <p>Password: <input type="password" name="password" size="20" maxlength="20" /></p>
    <p><input type="submit" name="submit" value="Login" /></p>
    <input type="hidden" name="submitted" value="TRUE" />
</form>
<?php
include ('./includes/footer.html');
?>
```

✔ Tips

- Some browsers have difficulties with cookies that do not list every argument. Explicitly stating every parameter—even as an empty string, as I did here—will achieve more reliable results across all browsers.
- Here are some general guidelines for cookie expirations: If the cookie should last as long as the session, do not set an expiration time; if the cookie should continue to exist after the user has closed and reopened his or her browser, set an expiration time months ahead; and if the cookie can constitute a security risk, set an expiration time of an hour or fraction thereof so that the cookie does not continue to exist too long after a user has left his or her browser.
- For security purposes, you could set a five- or ten-minute expiration time on a cookie and have the cookie resent with every new page the user visits (assuming that the cookie exists). This way, the cookie will continue to persist as long as the user is active but will automatically die five or ten minutes after the user's last action.
- Setting the path to `'/'` will make the cookie visible within an entire domain (Web site).
- Setting the domain to `'.site.com'` will make the cookie visible within an entire domain and every subdomain (`www.site.com`, `admin.site.com`, `pages.site.com`, etc.).
- E-commerce and other privacy-related Web applications should use an SSL (Secure Sockets Layer) connection for all transactions, including the cookie.

Deleting cookies

The final thing to understand about using cookies is how to delete one. While a cookie will automatically expire when the user's browser is closed or when the expiration date/time is met, sometimes you'll want to manually delete the cookie instead. For example, in Web sites that have registered users and login capabilities, you will probably want to delete any cookies when the user logs out.

Although the `setcookie()` function can take up to six arguments, only one is actually required—the cookie 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 *first_name*, you use this line:

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

To delete the *first_name* cookie, you would code:

```
setcookie('first_name');
```

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

```
setcookie('first_name', '', time()-300);
```

To demonstrate all of this, I'll add logout capability to the site, which will appear only to logged-in users. As an added bonus, the header file will be altered so that a *Logout* link appears when the user is logged-in and a *Login* link appears when the user is logged-out.

To delete a cookie:

1. Create a new PHP document in your text editor (**Script 9.4**).

   ```
   <?php # Script 9.4 - logout.php
   ```

Script 9.4 The `logout.php` script deletes the previously established cookies.

```
script
1   <?php # Script 9.4 - logout.php
2   // This page lets the user logout.
3
4   // If no cookie is present, redirect the
    user.
5   if (!isset($_COOKIE['user_id'])) {
6
7      // Start defining the URL.
8      $url = 'http://' . $_SERVER['HTTP_HOST']
       . dirname($_SERVER['PHP_SELF']);
9      // Check for a trailing slash.
10     if ((substr($url, -1) == '/') OR
       (substr($url, -1) == '\\') ) {
11        $url = substr ($url, 0, -1); // Chop
          off the slash.
12     }
13     $url .= '/index.php'; // Add the page.
14     header("Location: $url");
15     exit(); // Quit the script.
16
17  } else { // Delete the cookies.
18     setcookie ('first_name', '',
       time()-300, '/', '', 0);
19     setcookie ('user_id', '',
       time()-300, '/', '', 0);
20  }
21
22  // Set the page title and include the HTML
    header.
23  $page_title = 'Logged Out!';
24  include ('./includes/header.html');
25
26  // Print a customized message.
27  echo "<h1>Logged Out!</h1>
28  <p>You are now logged out, {$_COOKIE
    ['first_name']}!</p>
29  <p><br /><br /></p>";
30
31  include ('./includes/footer.html');
32  ?>
```

2. Check for the existence of a *user_id* cookie; if it is present, delete both cookies.

```
if (!isset($_COOKIE['user_id'])) {
  $url = 'http://' . $_SERVER
  →['HTTP_HOST'] . dirname($_SERVER
  →['PHP_SELF']);
  if ((substr($url, -1) == '/') OR
  →(substr($url, -1) == '\\') ) {
    $url = substr ($url, 0, -1);
  }
  $url .= '/index.php';
  header("Location: $url");
  exit();
} else {
  setcookie ('first_name', '',
  →time()-300, '/', '', 0);
  setcookie ('user_id', '',
  →time()-300, '/', '', 0);
}
```

As with my `loggedin.php` page, if the user is not already logged in, I want this page to redirect the user to the home page. If the user is logged in, these two cookies will effectively delete the existing ones.

3. Make the remainder of the PHP page.

```
$page_title = 'Logged Out!';
include ('./includes/header.html');
echo "<h1>Logged Out!</h1>
<p>You are now logged out, {$_COOKIE
→['first_name']}!</p>
<p><br /><br /></p>";
include ('./includes/footer.html');
?>
```

The page itself is also much like the `loggedin.php` page. Although it may seem odd that you can still refer to the *first_name* cookie (that you just deleted in this script), it makes perfect sense considering the process:

A) This page is requested by the client.

B) The server reads the appropriate cookies from the client's browser.

C) The page is run and does its thing (including sending new cookies).

So, in short, the original *first_name* cookie data is available to this script when it first runs. The set of cookies sent by this page (the delete cookies) aren't available to this page, so the original values are still usable.

4. Save the file as `logout.php`.

To create the logout link:

1. Open `header.html` (refer to Script 7.1) in your text editor.
2. Change the links to (**Script 9.5**)

```
<li class="navtop"><a href=
→ "index.php" title="Go to the Home
→ Page">Home</a></li>
<li><a href="register.php" title=
→ "Register">Register</a></li>
<li><?php
if ( (isset($_COOKIE['user_id'])) &&
→ (!strpos($_SERVER['PHP_SELF'],
→ 'logout.php')) ) {
  echo '<a href="logout.php" title=
→ "Logout">Logout</a>';
} else {
  echo '<a href="login.php" title=
→ "Login">Login</a>';
}
?></li>
```

Instead of having a permanent login link in my template, I'll have it display a *Logout* link if the user is logged in or a *Login* link if the user is not. The preceding conditional will accomplish just that based upon the presence of a cookie.

Because the `logout.php` script would ordinarily display a logout link (because the cookie exists when the page is first being viewed), I have to add a statement to my conditional, checking that the current page is not the `logout.php` script. The `strpos()` function, which checks if one string is found within another string, is an easy way to accomplish this.

Script 9.5 The `header.html` file now displays either a login or a logout link depending upon the user's current status.

script

```
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=iso-8859-1" />
6      <title><?php echo $page_title; ?></title>
7      <style type="text/css" media="all">
       @import "./includes/layout.css";</style>
8   </head>
9   <body>
10  <div id="wrapper"><!-- Goes with the CSS
    layout. -->
11
12     <div id="content"><!-- Goes with the CSS
       layout. -->
13
14        <div id="nav"><!-- Links section -->
15           <h3>Menu</h3>
16           <ul>
17           <li class="navtop"><a href=
             "index.php" title="Go to the Home
             Page">Home</a></li>
18           <li><a href="register.php" title=
             "Register">Register</a></li>
19           <li><?php
20  // Create a login/logout link.
21  if ( (isset($_COOKIE['user_id'])) &&
    (!strpos($_SERVER['PHP_SELF'], 'logout.
    php')) ) {
22     echo '<a href="logout.php" title=
       "Logout">Logout</a>';
23  } else {
24     echo '<a href="login.php" title=
       "Login">Login</a>';
25  }
26  ?></li>
27           </ul>
28        </div>
29        <!-- Script 9.5 - header.html -->
30        <!-- Start of page-specific content. -->
```

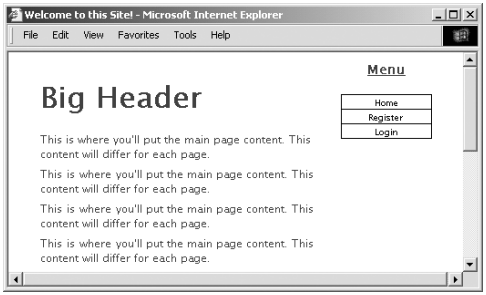

Figure 9.9 The home page with a *Login* link.

Figure 9.10 After the user logs in, the page now has a *Logout* link.

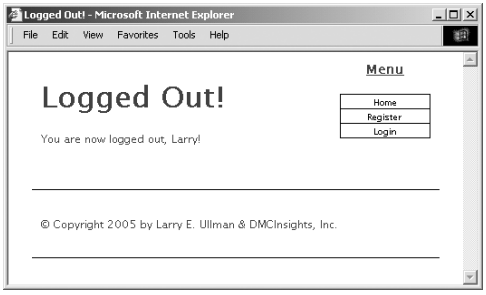

Figure 9.11 The result after logging out.

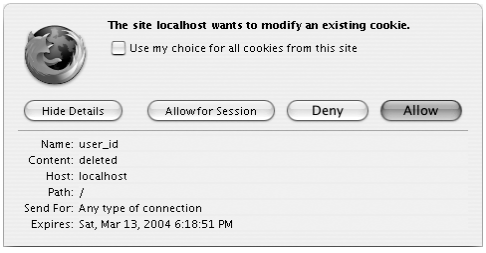

Figure 9.12 This is how the deletion cookie appears in a Firefox prompt.

3. Save the file, upload to the Web server (placed within the *includes* directory), and test the login/logout process in your Web browser (**Figures 9.9**, **9.10**, and **9.11**).

✔ Tips

- To see the result of the `setcookie()` calls in the `logout.php` script, turn on cookie prompting in your browser (**Figure 9.12**).
- Due to a bug in how Internet Explorer on Windows handles cookies, you may need to set the *domain* parameter to `false` (without quotes) in order to get the logout process to work when developing on your own computer (i.e., through *localhost*).
- When deleting a cookie, you should always use the same parameters that were used to set the cookie. If you set the domain and path in the creation cookie, use them again in the deletion cookie.
- To hammer the point home, remember that the deletion of a cookie does not take effect until the page has been reloaded or another page has been accessed (in other words, the cookie will still be available to a page after that page has deleted it). This is why I needed to add the `&& (!strpos($_SERVER['PHP_SELF'], 'logout.php')` clause to the `header.html` conditional (because the cookie itself would still be available on the `logout.php` page).

Using Sessions

Another method of making data available to multiple pages of a Web site is to use sessions. The premise of a session is that data is stored on the server, not in the Web browser, and a session identifier is used to locate a particular user's record (session data). This session identifier is normally stored in the user's Web browser via a cookie, but the sensitive data itself—like the user's ID, name, and so on—always remains on the server.

The question may arise: why use sessions at all when cookies work just fine? First of all, sessions are more secure in that all of the recorded information is stored on the server and not continually sent back and forth between the server and the client. Second, some users reject cookies or turn them off completely. Sessions, while designed to work with a cookie, can function without them, as you'll see in the next section of this chapter.

To demonstrate sessions—and to compare them with cookies—I will rewrite the previous scripts.

Setting session variables

The most important rule with respect to sessions is that each page that will use them must begin by calling the `session_start()` function. This function tells PHP to either begin a new session or access an existing one.

The first time this function is used, `session_start()` will attempt to send a cookie with a name of *PHPSESSID* (the session name) and a value of something like *a61f8670baa8e90a30c878df89a2074b* (32 hexadecimal letters, the session ID). Because of this attempt to send a cookie, `session_start()` must be called before any data is sent to the Web browser, as is the case when using the `setcookie()` and `header()` functions.

Once the session has been started, values can be registered to the session using

```
$_SESSION['key'] = 'value';
```

```
$_SESSION['name'] = 'Jessica';
```

```
$_SESSION['id'] = 48;
```

I'll rewrite the `login.php` script with this in mind.

Allowing for Sessions on Windows

Sessions in PHP requires a temporary directory on the server where PHP can store the session data. For Unix and Mac OS X users, this isn't a problem, as the `/tmp` directory is available explicitly for purposes such as this. For Windows users, you also do not need to do anything special as of version 4.3.6 of PHP. But if you are running Windows and an earlier version of PHP, you must configure the server. Here's how:

1. Create a new folder on your server, such as `C:\temp`.
2. Make sure that *Everyone* (or just the Web server user, if you know that value) can read and write to this folder.
3. Edit your `php.ini` file (see Appendix A, "Installation"), setting the value of *session.save_path* to this folder (`C:\temp`).
4. Restart the Web server.

If you see errors about the *session.save_path* when you first use sessions, pay attention to what the error messages say. Also double-check your path name and that you edited the correct `php.ini` file.

Script 9.6 The `login.php` script now uses sessions instead of cookies.

script

```
1   <?php # Script 9.6 - login.php (3rd version
    after Scripts 9.1 & 9.3)
2   // Send NOTHING to the Web browser prior to
    the session_start() line!
3
4   // Check if the form has been submitted.
5   if (isset($_POST['submitted'])) {
6
7      require_once ('../mysql_connect.php');
       // Connect to the db.
8
9      $errors = array(); // Initialize error
       array.
10
11     // Check for an email address.
12     if (empty($_POST['email'])) {
13        $errors[] = 'You forgot to enter your
          email address.';
14     } else {
15        $e = escape_data($_POST['email']);
16     }
17
18     // Check for a password.
19     if (empty($_POST['password'])) {
20        $errors[] = 'You forgot to enter your
          password.';
21     } else {
22        $p = escape_data($_POST
          ['password']);
23     }
24
25     if (empty($errors)) { // If everything's
       OK.
26
27        /* Retrieve the user_id and
          first_name for
28        that email/password combination. */
29        $query = "SELECT user_id,
          first_name FROM users WHERE
          email='$e' AND password=SHA('$p')";
```

(script continues on next page)

To begin a session:

1. Open `login.php` (refer to Script 9.3) in your text editor.
2. Replace the `setcookie()` lines (36–37) with these lines (**Script 9.6**):

```
session_start();
$_SESSION['user_id'] = $row[0];
$_SESSION['first_name'] = $row[1];
```

The first step is to begin the session. Since there are no `echo()` statements, `include` calls, or HTML prior to this point in the script, it will be safe to use `session_start()` now, although I could have placed it at the top of the script as well. Then, I add two *key-value* pairs to the `$_SESSION` superglobal array to register the user's first name and user ID to the session.

continues on page 346

Script 9.6 *continued*

script

```
30          $result = @mysql_query ($query); // Run the query.
31          $row = mysql_fetch_array ($result, MYSQL_NUM); // Return a record, if applicable.
32
33          if ($row) { // A record was pulled from the database.
34
35              // Set the session data & redirect.
36              session_start();
37              $_SESSION['user_id'] = $row[0];
38              $_SESSION['first_name'] = $row[1];
39
40              // Redirect the user to the loggedin.php page.
41              // Start defining the URL.
42              $url = 'http://' . $_SERVER['HTTP_HOST'] . dirname($_SERVER['PHP_SELF']);
43              // Check for a trailing slash.
44              if ((substr($url, -1) == '/') OR (substr($url, -1) == '\\') ) {
45                  $url = substr ($url, 0, -1); // Chop off the slash.
46              }
47              // Add the page.
48              $url .= '/loggedin.php';
49
50              header("Location: $url");
51              exit(); // Quit the script.
52
53          } else { // No record matched the query.
54              $errors[] = 'The email address and password entered do not match those on file.';
                // Public message.
55              $errors[] = mysql_error() . '<br /><br />Query: ' . $query; // Debugging message.
56          }
57
58      } // End of if (empty($errors)) IF.
59
60      mysql_close(); // Close the database connection.
61
62  } else { // Form has not been submitted.
63
64      $errors = NULL;
65
66  } // End of the main Submit conditional.
67
68  // Begin the page now.
69  $page_title = 'Login';
```

(script continues on next page)

Script 9.6 *continued*

```
script
70  include ('./includes/header.html');
71
72  if (!empty($errors)) { // Print any error messages.
73     echo '<h1 id="mainhead">Error!</h1>
74     <p class="error">The following error(s) occurred:<br />';
75     foreach ($errors as $msg) { // Print each error.
76        echo " - $msg<br />\n";
77     }
78     echo '</p><p>Please try again.</p>';
79  }
80
81  // Create the form.
82  ?>
83  <h2>Login</h2>
84  <form action="login.php" method="post">
85     <p>Email Address: <input type="text" name="email" size="20" maxlength="40" /> </p>
86     <p>Password: <input type="password" name="password" size="20" maxlength="20" /></p>
87     <p><input type="submit" name="submit" value="Login" /></p>
88     <input type="hidden" name="submitted" value="TRUE" />
89  </form>
90  <?php
91  include ('./includes/footer.html');
92  ?>
```

Sessions in Older Versions of PHP

Prior to version 4.1 of PHP (when the `$_SESSION` superglobal became available), session variables were set using the special `session_register()` function. The syntax was

```
session_start();

$name = 'Jessica';

session_register('name');
```

It's very important to notice that the `session_register()` function takes the name of a variable to register without the initial dollar sign (so *name* rather than *$name*).

Once a session variable is registered, you can refer to is using `$HTTP_SESSION_VARS['var']`.

To delete a session variable, you use the `session_unregister()` function.

To repeat, you only need to use these functions if you are using an *old* version of PHP (between 4.0 and 4.1). As always, see the PHP manual for more information on these functions.

3. Save the page as `login.php`, upload to your server, and test in your Web browser (**Figure 9.13**).

   Although `loggedin.php` and the header and script will need to be rewritten, you can still test the login script and see the resulting cookie (**Figure 9.14**). The `loggedin.php` page should redirect you back to the home page, though, as it's still checking for the presence of a `$_COOKIE` variable.

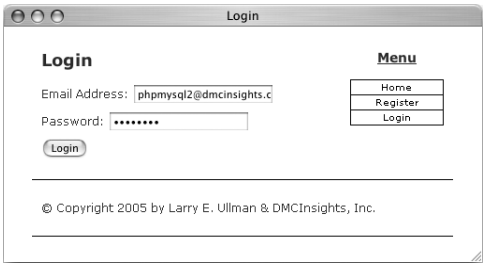

Figure 9.13 The login form remains unchanged to the end user, but the underlying functionality now uses sessions.

Figure 9.14 This cookie, created by PHP's `session_start()` function, stores the session ID.

✔ Tips

- Because sessions will normally send and read cookies, you should always try to begin them as early in the script as possible. Doing so will help you avoid the problem of attempting to send a cookie after the headers (HTML or white space) have already been sent (see Figure 9.1).
- If you want, you can set *session.auto_start* in the `php.ini` file to `1`, making it unnecessary to use `session_start()` on each page. This does put a greater toll on the server and, for that reason, shouldn't be used without some consideration of the circumstances.
- You can store arrays in sessions (making `$_SESSION` a multidimensional array), just as you can strings or numbers.

Script 9.7 I've updated `loggedin.php` so that it refers to `$_SESSION` and not `$_COOKIE`.

```
script
1   <?php # Script 9.7 - loggedin.php (2nd
    version after Script 9.2)
2   # User is redirected here from login.php.
3
4   session_start(); // Start the session.
5
6   // If no session value is present, redirect
    the user.
7   if (!isset($_SESSION['user_id'])) {
8
9       // Start defining the URL.
10      $url = 'http://' . $_SERVER['HTTP_HOST']
        . dirname($_SERVER['PHP_SELF']);
11      // Check for a trailing slash.
12      if ((substr($url, -1) == '/') OR
        (substr($url, -1) == '\\') ) {
13          $url = substr ($url, 0, -1); // Chop
            off the slash.
14      }
15      $url .= '/index.php'; // Add the page.
16      header("Location: $url");
17      exit(); // Quit the script.
18  }
19
20  // Set the page title and include the HTML
    header.
21  $page_title = 'Logged In!';
22  include ('./includes/header.html');
23
24  // Print a customized message.
25  echo "<h1>Logged In!</h1>
26  <p>You are now logged in, {$_SESSION
    ['first_name']}!</p>
27  <p><br /><br /></p>";
28
29  include ('./includes/footer.html');
30  ?>
```

Accessing session variables

Once a session has been started and variables have been registered to it, you can create other scripts that will access those variables. To do so, each script must first enable sessions, again using `session_start()`.

This function will give the current script access to the previously started session (if it can read the *PHPSESSID* value stored in the cookie) or create a new session if it cannot (in which case, it won't be able to access stored values because a new session will have been created).

To then refer to a session variable, use `$_SESSION['var']`, as you would refer to any other array.

To access session variables:

1. Open `loggedin.php` (refer to Script 9.2) in your text editor.
2. Add a call to the `session_start()` function (**Script 9.7**).

   ```
   session_start();
   ```

   Every PHP script that either sets or accesses session variables must use the `session_start()` function. This line must be called before the `header.html` file is included and before anything is sent to the Web browser.

continues on next page

3. Replace the references to `$_COOKIE` with `$_SESSION` (lines 5 and 24 of the original file).

   ```
   if (!isset($_SESSION['user_id'])) {
   ```

   and

   ```
   echo "<h1>Logged In!</h1>
   <p>You are now logged in, {$_SESSION
   →['first_name']}!</p>
   <p><br /><br /></p>";
   ```

   Switching a script from cookies to sessions requires only that you change uses of `$_COOKIE` to `$_SESSION`.

4. Save the file as `loggedin.php`, upload to your Web server, and test in your browser (**Figure 9.15**).

5. Replace the reference to `$_COOKIE` with `$_SESSION` in `header.html` (from Script 9.5 to **Script 9.8**).

   ```
   if ( (isset($_SESSION['user_id']))
   →&& (!strpos($_SERVER['PHP_SELF'],
   →'logout.php')) ) {
   ```

   For the *Login*/*Logout* links to function properly (notice the incorrect link in Figure 9.15), the reference to the cookie variable within the header file must be switched over to sessions. The header file does not need to call the `session_start()` function, as it'll be included by pages that do.

6. Save the header file, upload to the Web server, and test in your browser (**Figure 9.16**).

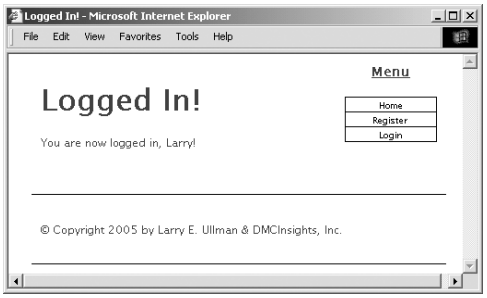

Figure 9.15 After logging in, the user is redirected to `loggedin.php`, which will welcome the user by name using the stored session value.

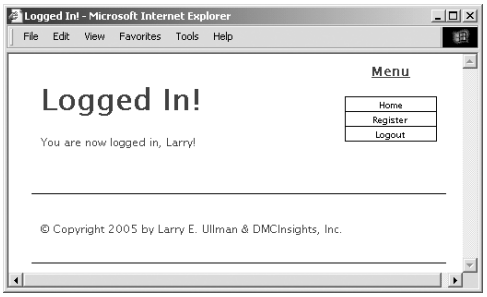

Figure 9.16 With the header file altered for sessions, the proper *Login*/*Logout* links will be displayed (compare with Figure 9.15).

Script 9.8 The `header.html` file now also references `$_SESSION`.

script

```
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=iso-8859-1" />
6      <title><?php echo $page_title; ?></title>
7      <style type="text/css" media="all">
       @import "./includes/layout.css";</style>
8   </head>
9   <body>
10  <div id="wrapper"><!-- Goes with the CSS
    layout. -->
11
12     <div id="content"><!-- Goes with the CSS
       layout. -->
13
14        <div id="nav"><!-- Links section -->
15           <h3>Menu</h3>
16           <ul>
17           <li class="navtop"><a href=
             "index.php" title="Go to the Home
             Page">Home</a></li>
18           <li><a href="register.php" title=
             "Register">Register</a></li>
19           <li><?php
20  // Create a login/logout link.
21  if ( (isset($_SESSION['user_id'])) &&
    (!strpos($_SERVER['PHP_SELF'], 'logout.
    php')) ) {
22     echo '<a href="logout.php" title=
       "Logout">Logout</a>';
23  } else {
24     echo '<a href="login.php" title=
       "Login">Login</a>';
25  }
26  ?></li>
27           </ul>
28        </div>
29        <!-- Script 9.8 - header.html -->
30        <!-- Start of page-specific content. -->
```

✔ Tips

- For the *Login*/*Logout* links to work on the other pages (`register.php`, `index.php`, etc.), you'll need to add the `session_start()` command to each of those.
- If you have an application where the session data does not seem to be accessible from one page to the next, it could be because a new session is being created on each page. To check for this, compare the session ID (the last few characters of the value will suffice) to see if it is the same. You can see the session's ID by viewing the session cookie as it is sent or by using the `session_id()` function:

  ```
  echo session_id();
  ```

- Session variables are available as soon as you've established them. So, unlike when using cookies, you can assign a value to `$_SESSION['var']` and then refer to `$_SESSION['var']` later in that same script.

Deleting session variables

When using sessions—particularly with a login/logout system as I've established here—you need to create a method to delete the session variables. In the current example, this would be necessary when the user logs out.

Whereas a cookie system only requires that another cookie be sent to destroy the existing cookie, sessions are more demanding, since there are both the cookie on the client and the data on the server to consider.

To delete an individual session variable, you can use the `unset()` function (which works with any variable in PHP):

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

To delete every session variable, reset the entire `$_SESSION` array:

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

Finally, to remove all of the session data from the server, use `session_destroy()`:

```
session_destroy();
```

Note that prior to using any of these methods, the page must begin with `session_start()` so that the existing session is accessed.

To delete a session:

1. Create a new PHP script in your text editor (**Script 9.9**).

   `<?php # Script 9.9 - logout.php`

   The logout script will log out the user and delete all the session information.

2. Invoke the session.

   `session_start();`

   Anytime you are using sessions, you must use the `session_start()` function, preferably at the very beginning of a page. This is true even if you are deleting a session.

Script 9.9 Destroying a session requires special syntax.

script

```
1   <?php # Script 9.9 - logout.php (2nd
    version after Script 9.4)
2   // This page lets the user logout.
3
4   session_start(); // Access the existing
    session.
5
6   // If no session variable exists, redirect
    the user.
7   if (!($_SESSION['user_id'])) {
8
9      // Start defining the URL.
10     $url = 'http://' . $_SERVER['HTTP_HOST']
       . dirname($_SERVER['PHP_SELF']);
11     // Check for a trailing slash.
12     if ((substr($url, -1) == '/') OR (substr
       ($url, -1) == '\\') ) {
13        $url = substr ($url, 0, -1); // Chop
          off the slash.
14     }
15     $url .= '/index.php'; // Add the page.
16     header("Location: $url");
17     exit(); // Quit the script.
18
19  } else { // Cancel the session.
20     $_SESSION = array(); // Destroy the
       variables.
21     session_destroy(); // Destroy the
       session itself.
22     setcookie ('PHPSESSID', '', time()-300,
       '/', '', 0); // Destroy the cookie.
23  }
24
25  // Set the page title and include the HTML
    header.
26  $page_title = 'Logged Out!';
27  include ('./includes/header.html');
28
29  // Print a customized message.
30  echo "<h1>Logged Out!</h1>
31  <p>You are now logged out!</p>
32  <p><br /><br /></p>";
33
34  include ('./includes/footer.html');
35  ?>
```

3. Check for the presence of the `$_SESSION['user_id']` variable.

```
if (!($_SESSION['user_id'])) {
  $url = 'http://' . $_SERVER
  →['HTTP_HOST'] . dirname($_SERVER
  →['PHP_SELF']);
  if ((substr($url, -1) == '/') OR
  →(substr($url, -1) == '\\') ) {
    $url = substr ($url, 0, -1);
  }
  $url .= '/index.php';
  header("Location: $url");
  exit();
```

As with the `logout.php` script in the cookie examples, if the user is not currently logged in, he or she will be redirected.

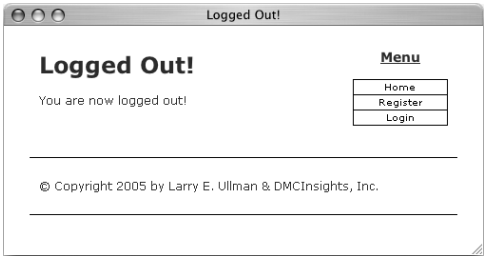

Figure 9.17 The logout page.

4. Destroy all of the session material.

```
} else {
  $_SESSION = array();
  session_destroy();
  setcookie ('PHPSESSID', '',
  →time()-300, '/', '', 0);
}
```

The second line here will reset the entire `$_SESSION` variable as a new array, erasing its existing values. The third line removes the data from the server, and the fourth sends a cookie to replace the existing session cookie in the browser.

5. Create the HTML and print a message.

```
$page_title = 'Logged Out!';
include ('./includes/header.html');
echo "<h1>Logged Out!</h1>
<p>You are now logged out!</p>
<p><br /><br /></p>";
include ('./includes/footer.html');
?>
```

Unlike when using the cookie `logout.php` script, you cannot refer to the user by their first name anymore, as all of that data has been deleted.

6. Save the file as `logout.php`, upload to your Web server, and test in your browser (**Figure 9.17**).

✔ Tips

- If you are using an older version of PHP (prior to version 4.1) and the `$_SESSION` array is not available, use `session_unset()` in lieu of `$_SESSION = array()`.
- Never set `$_SESSION` equal to `NULL`, because that could cause problems on some servers.
- To delete just one session variable, use `unset($_SESSION['var'])`.

Changing the session behavior

As part of PHP's support for sessions, there are about 20 different configuration options you can set for how PHP handles sessions. **Table 9.1** lists the most important of these.

Each of these settings, except for *session.use_trans_sid*, can be set within your PHP script using the `ini_set()` function (covered in the preceding chapter):

```
ini_set (parameter, new_setting);
```

For example, to change where PHP stores the session data, use

```
ini_set ('session.save_path',
→ '/path/to/folder');
```

To set the name of the session (perhaps to make a more user-friendly one), you can use either `ini_set()` or the simpler `session_name()` function.

```
session_name('YourSession');
```

The benefits of creating your own session name are twofold: it's marginally more secure and it may be better received by the end user (since the session name is the cookie name the end user will see). That being said, for `session_name()` to work, it must be called before every use of `session_start()` in your entire Web application. I'll rewrite the example with this in mind.

Session Configuration Settings

Setting	Example	Meaning
session.auto_start	0	If sessions should be automatically used (0 means no).
session.cookie_domain	www.dmcinsights.com	The URL wherein the session cookie should be accessible.
session.cookie_lifetime	0	How long, in seconds, the session cookie should exist (0 means for the life of the browser).
session.cookie_path	/	The domain path wherein the cookie should be accessible.
session.cookie_secure	0	Whether or not the cookie must be sent over a secure connection (0 means no).
session.gc_probability	1	The odds of performing garbage collection from 1 to 100.
session.gc_maxlifetime	1440	The time period in seconds a session should last.
session.name	PHPSESSID	The name given to all sessions.
session.save_handler	files	How the session data will be stored.
session.save_path	/tmp	Where session data will be stored.
session.serialize_handler	php	What method should be used to serialize the session variables.
session.use_cookies	1	Whether or not the session ID should be stored in a cookie (0 means no).
session.use_only_cookies	0	Whether or not the session ID must be stored in a cookie (0 means no).
session.use_trans_sid	0	Whether or not PHP should add the session ID to every link in an application (0 means no).

Table 9.1 PHP's session configuration options, with the default setting listed as most of the examples.

Script 9.10 The `login.php` script now uses an original session name.

```
script
1   <?php # Script 9.10 - login.php (4th
    version after Scripts 9.1, 9.3 & 9.6)
2   // Send NOTHING to the Web browser prior to
    the session_start() line!
3
4   // Check if the form has been submitted.
5   if (isset($_POST['submitted'])) {
6
7      require_once ('../mysql_connect.php');
       // Connect to the db.
8
9      $errors = array(); // Initialize error
       array.
10
11     // Check for an email address.
12     if (empty($_POST['email'])) {
13        $errors[] = 'You forgot to enter your
          email address.';
14     } else {
15        $e = escape_data($_POST['email']);
16     }
17
18     // Check for a password.
19     if (empty($_POST['password'])) {
20        $errors[] = 'You forgot to enter your
          password.';
21     } else {
22        $p = escape_data($_POST
          ['password']);
23     }
24
25     if (empty($errors)) { // If everything's
       OK.
26
27        /* Retrieve the user_id and
          first_name for
28        that email/password combination. */
29        $query = "SELECT user_id, first_name
          FROM users WHERE email='$e' AND
          password=SHA('$p')";
```

(script continues on next page)

To use your own session names:

1. Open `login.php` (refer to Script 9.6) in your text editor.
2. Before the `session_start()` call (line 36), add the following (**Script 9.10**):

   ```
   session_name ('YourVisitID');
   ```

   Instead of having the session be named *PHPSESSID*, which may be imposing as a cookie name, I'll use the friendlier *YourVisitID*.

continues on page 356

Script 9.10 *continued*

script

```
30          $result = @mysql_query ($query); // Run the query.
31          $row = mysql_fetch_array ($result, MYSQL_NUM); // Return a record, if applicable.
32
33          if ($row) { // A record was pulled from the database.
34
35             // Set the session data & redirect.
36             session_name ('YourVisitID');
37             session_start();
38             $_SESSION['user_id'] = $row[0];
39             $_SESSION['first_name'] = $row[1];
40
41             // Redirect the user to the loggedin.php page.
42             // Start defining the URL.
43             $url = 'http://' . $_SERVER['HTTP_HOST'] . dirname($_SERVER['PHP_SELF']);
44             // Check for a trailing slash.
45             if ((substr($url, -1) == '/') OR (substr($url, -1) == '\\') ) {
46                $url = substr ($url, 0, -1); // Chop off the slash.
47             }
48             // Add the page.
49             $url .= '/loggedin.php';
50
51             header("Location: $url");
52             exit(); // Quit the script.
53
54          } else { // No record matched the query.
55             $errors[] = 'The email address and password entered do not match those on file.';
               // Public message.
56             $errors[] = mysql_error() . '<br /><br />Query: ' . $query; // Debugging message.
57          }
58
59       } // End of if (empty($errors)) IF.
60
61       mysql_close(); // Close the database connection.
62
63    } else { // Form has not been submitted.
64
65       $errors = NULL;
66
67    } // End of the main Submit conditional.
68
```

(script continues on next page)

Script 9.10 *continued*

script

```
69  // Begin the page now.
70  $page_title = 'Login';
71  include ('./includes/header.html');
72
73  if (!empty($errors)) { // Print any error messages.
74      echo '<h1 id="mainhead">Error!</h1>
75      <p class="error">The following error(s) occurred:<br />';
76      foreach ($errors as $msg) { // Print each error.
77          echo " - $msg<br />\n";
78      }
79      echo '</p><p>Please try again.</p>';
80  }
81
82  // Create the form.
83  ?>
84  <h2>Login</h2>
85  <form action="login.php" method="post">
86      <p>Email Address: <input type="text" name="email" size="20" maxlength="40" /> </p>
87      <p>Password: <input type="password" name="password" size="20" maxlength="20" /></p>
88      <p><input type="submit" name="submit" value="Login" /></p>
89      <input type="hidden" name="submitted" value="TRUE" />
90  </form>
91  <?php
92  include ('./includes/footer.html');
93  ?>
```

3. Repeat the process for `loggedin.php` (compare Script 9.7 with **Script 9.11**).

   Because every page must use the same session name, this line of code has to be added to the `loggedin.php` and `logout.php` scripts for them to work properly.

Script 9.11 The same session name (*YourVisitID*) must be used across every script.

script

```
1   <?php # Script 9.11 - loggedin.php (3rd
    version after Scripts 9.2 & 9.7)
2   # User is redirected here from login.php.
3
4   session_name ('YourVisitID');
5   session_start(); // Start the session.
6
7   // If no session value is present, redirect
    the user.
8   if (!isset($_SESSION['user_id'])) {
9
10      // Start defining the URL.
11      $url = 'http://' . $_SERVER['HTTP_HOST']
        . dirname($_SERVER['PHP_SELF']);
12      // Check for a trailing slash.
13      if ((substr($url, -1) == '/') OR (substr
        ($url, -1) == '\\') ) {
14          $url = substr ($url, 0, -1); // Chop
            off the slash.
15      }
16      $url .= '/index.php'; // Add the page.
17      header("Location: $url");
18      exit(); // Quit the script.
19  }
20
21  // Set the page title and include the HTML
    header.
22  $page_title = 'Logged In!';
23  include ('./includes/header.html');
24
25  // Print a customized message.
26  echo "<h1>Logged In!</h1>
27  <p>You are now logged in, {$_SESSION
    ['first_name']}!</p>
28  <p><br /><br /></p>";
29
30  include ('./includes/footer.html');
31  ?>
```

Script 9.12 The `logout.php` page uses the `session_name()` function to also determine the name of the cookie to be sent.

script

```
1   <?php # Script 9.12 - logout.php (3rd
    version after Scripts 9.4 & 9.9)
2   // This page lets the user logout.
3
4   session_name ('YourVisitID');
5   session_start(); // Access the existing
    session.
6
7   // If no session variable exists, redirect
    the user.
8   if (!($_SESSION['user_id'])) {
9
10      // Start defining the URL.
11      $url = 'http://' . $_SERVER['HTTP_HOST']
        . dirname($_SERVER['PHP_SELF']);
12      // Check for a trailing slash.
13      if ((substr($url, -1) == '/') OR (substr
        ($url, -1) == '\\') ) {
14          $url = substr ($url, 0, -1); // Chop
            off the slash.
15      }
16      $url .= '/index.php'; // Add the page.
17      header("Location: $url");
18      exit(); // Quit the script.
19
20  } else { // Cancel the session.
21      $_SESSION = array(); // Destroy the
        variables.
22      session_destroy(); // Destroy the
        session itself.
23      setcookie (session_name(), '',
        time()-300, '/', '', 0); // Destroy
        the cookie.
24  }
25
26  // Set the page title and include the HTML
    header.
27  $page_title = 'Logged Out!';
28  include ('./includes/header.html');
29
30  // Print a customized message.
```

(script continues on next page)

4. Add the following line to `logout.php` (compare Script 9.9 with **Script 9.12**):

   ```
   session_name ('YourVisitID');
   ```

5. Change the `setcookie()` line of `logout.php` so that it uses the `session_name()` function:

   ```
   setcookie (session_name(), '',
   → time()-300, '/', '', 0);
   ```

   The `session_name()` function will set the session name or return the current session name (if no argument is given). Since I want to send a cookie using the same cookie name as was used to create the cookie, the `session_name()` function will set that value appropriately.

continues on next page

6. Save all the files, upload to your Web server, and test in your Web browser.
7. If desired, view the cookie that was set during the login process (**Figure 9.18**).

Script 9.12 *continued*

script

```
31  echo "<h1>Logged Out!</h1>
32  <p>You are now logged out!</p>
33  <p><br /><br /></p>";
34
35  include ('./includes/footer.html');
36  ?>
```

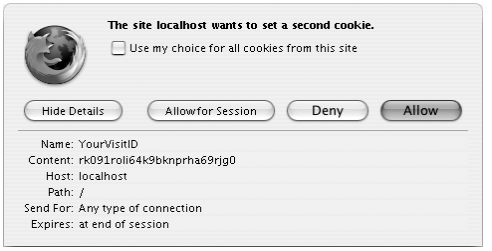

Figure 9.18 The cookie's name will correspond to the session name.

Sessions and Cookies

In the previous examples I've accomplished the same tasks (logging in and logging out) using cookies and sessions. Obviously, both are easy to use in PHP, but the true question is when to use one or the other.

Sessions have the following advantages over cookies:

- They are generally more secure (because the data is being retained on the server).
- They allow for more data to be stored.
- They can be used without cookies.

Whereas cookies have the following advantages over sessions:

- They are easier to program.
- They require less of the server.

In general, to store and retrieve just a couple of small pieces of information, use cookies. For most of your Web applications, though, you'll use sessions. But since sessions do rely upon cookies by default, I'll discuss how to better manage this relationship.

Changing the session cookie settings

As it stands, the cookie sent by the `session_start()` function uses certain default parameters: an expiration of `0` (meaning the cookie will last as long as the browser remains open), a path of `'/'` (the cookie is available in the current folder and all of its subfolders), and no domain name. To change any of these settings, you can use the `session_set_cookie_params()` function:

```
session_set_cookie_params(expiration,
→ 'path', 'domain', secure);
```

The expiration setting is the only required value and is set in seconds with `0` as the default. This is not the number of seconds from the epoch (as is the case with the `setcookie()` function), and therefore you would use just *300* (for five minutes) rather than *time() + 300* (for five minutes from now).

To change the session cookie settings:

1. Open `login.php` (refer to Script 9.10) in your text editor.
2. Prior to the `session_start()` call (line 37), add the following (**Script 9.13**):

   ```
   session_set_cookie_params (900,
   → '/ch09/', 'www.domain.com');
   ```

   The `session_set_cookie_params()` function must be used before `session_start()` to be effective. Change the path and domain setting to those values that make sense for your application, or omit the values to use the defaults. In this example, I'll give the cookie an expiration time that's 15 minutes from now.
3. Save the file as `login.php`, upload to your Web server, and test in your browser.

   After 15 minutes, the cookie will expire and the PHP scripts should no longer be able to access the session values (*first_name* and *user_id*).

continues on page 362

Script 9.13 This version of the `login.php` script sets explicit cookie parameters.

script

```
1   <?php # Script 9.13 - login.php (5th
    version after Scripts 9.1, 9.3, 9.6 & 9.10)
2   // Send NOTHING to the Web browser prior to
    the session_start() line!
3
4   // Check if the form has been submitted.
5   if (isset($_POST['submitted'])) {
6
7      require_once ('../mysql_connect.php');
       // Connect to the db.
8
9      $errors = array(); // Initialize error
       array.
10
11     // Check for an email address.
12     if (empty($_POST['email'])) {
13        $errors[] = 'You forgot to enter your
          email address.';
14     } else {
15        $e = escape_data($_POST['email']);
16     }
17
18     // Check for a password.
19     if (empty($_POST['password'])) {
20        $errors[] = 'You forgot to enter your
          password.';
21     } else {
22        $p = escape_data($_POST
          ['password']);
23     }
24
25     if (empty($errors)) { // If everything's
OK.
26
27        /* Retrieve the user_id and
          first_name for
28        that email/password combination. */
29        $query = "SELECT user_id,
          first_name FROM users WHERE email=
          '$e' AND password=SHA('$p')";
30        $result = @mysql_query ($query); //
          Run the query.
```

(script continues on next page)

Script 9.13 *continued*

```
script
31          $row = mysql_fetch_array ($result, MYSQL_NUM); // Return a record, if applicable.
32
33          if ($row) { // A record was pulled from the database.
34
35             // Set the session data & redirect.
36             session_name ('YourVisitID');
37             session_set_cookie_params (900, '/ch09/', 'www.domain.com');
38             session_start();
39             $_SESSION['user_id'] = $row[0];
40             $_SESSION['first_name'] = $row[1];
41
42             // Redirect the user to the loggedin.php page.
43             // Start defining the URL.
44             $url = 'http://' . $_SERVER['HTTP_HOST'] . dirname($_SERVER['PHP_SELF']);
45             // Check for a trailing slash.
46             if ((substr($url, -1) == '/') OR (substr($url, -1) == '\\') ) {
47                $url = substr ($url, 0, -1); // Chop off the slash.
48             }
49             // Add the page.
50             $url .= '/loggedin.php';
51
52             header("Location: $url");
53             exit(); // Quit the script.
54
55          } else { // No record matched the query.
56             $errors[] = 'The email address and password entered do not match those on file.';
               // Public message.
57             $errors[] = mysql_error() . '<br /><br />Query: ' . $query; // Debugging message.
58          }
59
60       } // End of if (empty($errors)) IF.
61
62       mysql_close(); // Close the database connection.
63
64    } else { // Form has not been submitted.
65
66       $errors = NULL;
67
68    } // End of the main Submit conditional.
69
70    // Begin the page now.
71    $page_title = 'Login';
```

(script continues on next page)

4. View the cookie being sent (**Figure 9.19**).
5. Alter `loggedin.php` and `logout.php` so that the `setcookie()` function uses the same parameters as `login.php` (except for the expiration time on the logout page, of course).

✔ Tips

- The `session_get_cookie_params()` function returns an array of the current session cookie settings.
- The session cookie parameters can also be altered using the `ini_set()` function.
- The expiration time of the cookie refers only to the longevity of the cookie in the Web browser, not to how long the session data will be stored on the server.

Script 9.13 *continue*

```
script
72  include ('./includes/header.html');
73
74  if (!empty($errors)) { // Print any error
    messages.
75      echo '<h1 id="mainhead">Error!</h1>
76      <p class="error">The following error(s)
        occurred:<br />';
77      foreach ($errors as $msg) { // Print
        each error.
78          echo " - $msg<br />\n";
79      }
80      echo '</p><p>Please try again.</p>';
81  }
82
83  // Create the form.
84  ?>
85  <h2>Login</h2>
86  <form action="login.php" method="post">
87      <p>Email Address: <input type="text"
        name="email" size="20" maxlength="40"
        /> </p>
88      <p>Password: <input type="password"
        name="password" size="20" maxlength="20"
        /></p>
89      <p><input type="submit" name="submit"
        value="Login" /></p>
90      <input type="hidden" name="submitted"
        value="TRUE" />
91  </form>
92  <?php
93  include ('./includes/footer.html');
94  ?>
```

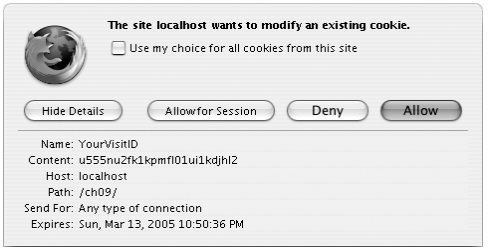

Figure 9.19 The session cookie now has an expiration time set.

Using sessions without cookies

One of the problems with sessions is that, by default, they rely on the use of a cookie to work properly. When a session is started, it sends a cookie that resides in the user's Web browser. Every subsequent page that calls `session_start()` makes use of the cookie, which contains the session name and ID, to know to use an existing session and to not create a new one. The problem is that users may have cookies turned off in their Web browser or may not accept the cookie because they do not understand its purpose. If this is the case, PHP will create a new session for each page and none of the registered variables will be accessible.

You can use sessions without cookies by passing along the session name and ID from page to page. This is simple enough to do, but if you forget to pass the session in only one instance, the entire process is shot.

To pass the session name from page to page, you can use the `SID` constant, which stands for *session ID* and has a value like *session_name=session_ID*. If this value is appended to every URL within the site, the sessions will still work even if the user did not accept the cookie. Note, though, that PHP only assigns a value to `SID` if no session cookie is present.

Garbage Collection

Garbage collection with respect to sessions is the process of deleting the session files (where the actual data is stored). Creating a logout system that destroys a session is ideal, but there's no guarantee all users will formally log out as they should. For this reason, PHP includes a cleanup process.

Whenever the `session_start()` function is called, PHP's garbage collection kicks in, checking the last modification date of each session (a session is modified whenever variables are set or retrieved). The server overhead of all this can become costly for busy sites, so you can tweak PHP's behavior in this regard.

Two settings dictate garbage collection: `session_gc_maxlifetime` and `session.gc_probability`. The first states after how many seconds of inactivity a session is considered idle and will therefore be deleted. The second setting determines the probability that garbage collection is performed, on a scale of 1 to 100. So, with the default settings, each call to `session_start()` has a 1 percent chance of invoking garbage collection. If PHP does start the cleanup, any sessions that have not been used in more than 1,440 seconds will be deleted.

With this in mind, you can alter PHP's garbage collection habits to better suit your application. Twenty-four minutes is a reasonable amount of idle time, but you'll want to increase the probability to somewhere closer to 30 percent so that there is a good balance between performance and clutter.

To use sessions without cookies:

1. Open `login.php` (refer to Script 9.13) in your text editor.
2. Replace the `session_set_cookie_params()` line with this one (**Script 9.14**):
   `ini_set('session.use_cookies', 0);`
   This code will tell PHP to specifically not use any cookies.
3. Alter the final `$url` creation line to be
   `$url .= '/loggedin.php?' . SID;`
   The addition of `?` and `SID` to the redirect will add *?session_name=session_ID* to the URL, effectively passing the session ID to the `loggedin.php` script.

continues on page 366

Script 9.14 This version of the `login.php` script does not use cookies at all, instead maintaining the state by passing the session ID in the URL.

script

```
1   <?php # Script 9.14 - login.php (6th version
    after Scripts 9.1, 9.3, 9.6, 9.10 & 9.13)
2   // Send NOTHING to the Web browser prior to
    the session_start() line!
3
4   // Check if the form has been submitted.
5   if (isset($_POST['submitted'])) {
6
7      require_once ('../mysql_connect.php');
       // Connect to the db.
8
9      $errors = array(); // Initialize error
       array.
10
11     // Check for an email address.
12     if (empty($_POST['email'])) {
13        $errors[] = 'You forgot to enter your
          email address.';
14     } else {
15        $e = escape_data($_POST['email']);
16     }
17
18     // Check for a password.
19     if (empty($_POST['password'])) {
20        $errors[] = 'You forgot to enter your
          password.';
21     } else {
22        $p = escape_data($_POST['password']);
23     }
24
25     if (empty($errors)) { // If everything's
       OK.
26
27        /* Retrieve the user_id and
          first_name for
28        that email/password combination. */
29        $query = "SELECT user_id, first_name
          FROM users WHERE email='$e' AND
          password=SHA('$p')";
30        $result = @mysql_query ($query); //
          Run the query.
```

(script continues on next page)

Script 9.14 *continued*

```
script
31          $row = mysql_fetch_array ($result, MYSQL_NUM); // Return a record, if applicable.
32
33          if ($row) { // A record was pulled from the database.
34
35             // Set the session data & redirect.
36             session_name ('YourVisitID');
37             ini_set('session.use_cookies', 0); // Don't use cookies.
38             session_start();
39             $_SESSION['user_id'] = $row[0];
40             $_SESSION['first_name'] = $row[1];
41
42             // Redirect the user to the loggedin.php page.
43             // Start defining the URL.
44             $url = 'http://' . $_SERVER['HTTP_HOST'] . dirname($_SERVER['PHP_SELF']);
45             // Check for a trailing slash.
46             if ((substr($url, -1) == '/') OR (substr($url, -1) == '\\') ) {
47                $url = substr ($url, 0, -1); // Chop off the slash.
48             }
49             // Add the page.
50             $url .= '/loggedin.php?' . SID; // Add the session name & ID.
51
52             header("Location: $url");
53             exit(); // Quit the script.
54
55          } else { // No record matched the query.
56             $errors[] = 'The email address and password entered do not match those on file.';
               // Public message.
57             $errors[] = mysql_error() . '<br /><br />Query: ' . $query; // Debugging message.
58          }
59
60       } // End of if (empty($errors)) IF.
61
62       mysql_close(); // Close the database connection.
63
64    } else { // Form has not been submitted.
65
66       $errors = NULL;
67
68    } // End of the main Submit conditional.
69
70    // Begin the page now.
71    $page_title = 'Login';
```

(script continues on next page)

4. Save the file, upload to your Web server, and test in your Web browser (**Figure 9.20**).
5. Copy the URL from the browser and paste it into another browser (**Figure 9.21**).

   Called *session hijacking*, this is one of the reasons to rely upon cookies whenever possible. The next section of this chapter will introduce preventive measures.

To spare you from having to review minor modifications to the same scripts yet again (and to save precious book space), I am not including new versions of `header.html`, `loggedin.php`, and `logout.php`. But since you should know how to edit these files in order to use sessions without cookies, I'll quickly outline the necessary changes.

Script 9.14 *continue*

script

```
72  include ('./includes/header.html');
73
74  if (!empty($errors)) { // Print any error
    messages.
75      echo '<h1 id="mainhead">Error!</h1>
76      <p class="error">The following error(s)
        occurred:<br />';
77      foreach ($errors as $msg) { // Print
        each error.
78          echo " - $msg<br />\n";
79      }
80      echo '</p><p>Please try again.</p>';
81  }
82
83  // Create the form.
84  ?>
85  <h2>Login</h2>
86  <form action="login.php" method="post">
87      <p>Email Address: <input type="text"
        name="email" size="20" maxlength="40"
        /> </p>
88      <p>Password: <input type="password"
        name="password" size="20" maxlength="20"
        /></p>
89      <p><input type="submit" name="submit"
        value="Login" /></p>
90      <input type="hidden" name="submitted"
        value="TRUE" />
91  </form>
92  <?php
93  include ('./includes/footer.html');
94  ?>
```

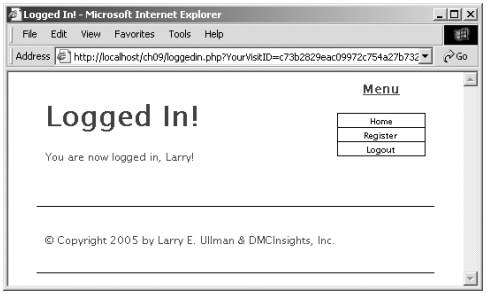

Figure 9.20 When the browser is redirected to the `loggedin.php` page, the session name and ID will be appended to the URL.

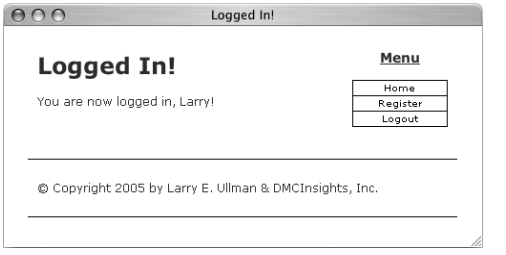

Figure 9.21 By using an existing session ID in a new browser, I can hijack another user's session and have access to all of that user's registered session data.

To edit the other files:

1. Edit `header.html` so that every link includes *?session_name=session_ID* (script not shown).

   The other problem with using sessions without cookies (besides the security issue) is that you must account for every link in your entire application. To modify the header file, you'll need to define the links like so:

   ```
   <a href="index.php?<?php echo
   → SID; ?>" title="Go to the Home
   → Page">Home</a>
   ```

2. Edit `loggedin.php` so that it also includes the `ini_set()` line.

   This script would then begin like so:

   ```
   session_name ('YourVisitID');
   ini_set('session.use_cookies', 0);
   session_start();
   ```

   You would also need to do this step for any other page that uses sessions.

3. Make the same changes to `logout.php`.

4. Remove the `setcookie()` line from `logout.php`.

   Since a session cookie is no longer being used, there is no reason to set another cookie deleting it.

✔ Tips

- If you have access to your `php.ini` file, you can set *session.use_trans_sid* to *1* or *On*. Doing so will have PHP automatically append `SID` to every URL as you have done manually here. It will slow down execution of the scripts, though, because PHP will need to check every page for URLs.
- The `session_id()` function returns the current session value (or allows you to specify which session to use).
- You can also pass `SID` from one page to another by storing it as a hidden input type in a form.
- Depending on the Web browser being used by the client, a session may be either browser-specific or window-specific. If the latter is the case, a pop-up window in your site will not be part of the same session unless it has received the session ID.
- Remember that using this method of storing and passing the session ID is less secure than using cookies for that purpose. If security isn't really a concern for your Web site (for example, if you're not dealing with personal information or e-commerce), then this is less of an issue.

Improving Session Security

Because important information is normally stored in a session (as opposed to a cookie), security becomes more of an issue. Remember that with sessions there are two considerations: the session ID, which is a reference point to the session data, and the session data itself, stored on the server. A malicious person is far more likely to hack into a session through the session ID than the data on the server, so I'll focus on that side of things here.

Storing the session ID in a cookie is considered the more secure method of using sessions, as opposed to passing the session ID along in URLs or storing it in hidden form inputs. Those alternatives are less secure because the session could easily be *hijacked* by another user, as you already witnessed. If I can learn another user's session ID, I can easily trick a server into thinking that it is *my* session ID. At that point I have effectively taken over the original user's entire session and may have access to their data. So storing the session ID in a cookie makes it somewhat harder to steal.

One method of preventing hijacking is to store some sort of user identifier in the session, and then to repeatedly double-check this value. The *HTTP_USER_AGENT*—a combination of the browser and operating system being used—is a likely candidate for this purpose. This adds a layer of security in that I could only hijack another user's session if I am running the exact same browser and operating system. For example, a login page would have

```
$_SESSION['agent'] = $_SERVER
→['HTTP_USER_AGENT'];
```

Then subsequent pages would check the stored *HTTP_USER_AGENT* against the user's *HTTP_USER_AGENT* (which should be the same).

```
if ($_SERVER['HTTP_USER_AGENT'] !=
→$_SESSION['agent']) {

   /* The session has probably

   been hijacked! */

}
```

As a demonstration of this, I'll modify the examples one last time. While I'm focusing on security, I'll encrypt the `$_SERVER['HTTP_USER_AGENT']` information using the `md5()` function to make it harder to fake.

Preventing Session Fixation

Another specific kind of session attack is known as *session fixation*. This is where one user specifies the session ID that another user should use. This session ID could be randomly generated or legitimately created. In either case, the real user will go into the site using the fixed session ID and do whatever. Then the malicious user can access that session because they know what the session ID is. You can help protect against these types of attack by changing the session ID. The `session_regenerate_id()` does just that, providing a new session ID to refer to the current session data. You can use this function should anything of consequence change during a user's session.

Script 9.15 This final version of the `login.php` script also stores an encrypted form of the user's *HTTP_USER_AGENT* (the browser and operating system of the client) in a session.

script

```
1   <?php # Script 9.15 - login.php (7th
    version after Scripts 9.1, 9.3, 9.6, 9.10.
    9.13 & 9.14)
2   // Send NOTHING to the Web browser prior to
    the session_start() line!
3
4   // Check if the form has been submitted.
5   if (isset($_POST['submitted'])) {
6
7      require_once ('../mysql_connect.php');
       // Connect to the db.
8
9      $errors = array(); // Initialize error
       array.
10
11     // Check for an email address.
12     if (empty($_POST['email'])) {
13        $errors[] = 'You forgot to enter your
          email address.';
14     } else {
15        $e = escape_data($_POST['email']);
16     }
17
18     // Check for a password.
19     if (empty($_POST['password'])) {
20        $errors[] = 'You forgot to enter your
          password.';
21     } else {
22        $p = escape_data($_POST
          ['password']);
23     }
24
25     if (empty($errors)) { // If everything's
       OK.
26
27        /* Retrieve the user_id and
          first_name for
28        that email/password combination. */
```

(script continues on next page)

To use sessions more securely:

1. Open `login.php` (refer to Script 9.14) in your text editor.
2. Delete the `ini_set()` line and remove the reference to `SID` (**Script 9.15**).

   For security purposes, I'll revert to using cookies to store the session ID. I also no longer need to append `SID` to the `header()` redirection URL.
3. After assigning the other session variables, store the *HTTP_USER_AGENT*.

   ```
   $_SESSION['agent'] = md5($_SERVER
   → ['HTTP_USER_AGENT']);
   ```

   The *HTTP_USER_AGENT* is part of the `$_SERVER` array (you may recall using it way back in Chapter 1, "Introduction to PHP"). It will have a value like *Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; .NET CLR 1.1.4322)*. This variable is run through the `md5()` function, which will turn it into a 32-character hexadecimal *hash* (although it's just easier to say that the data is encrypted).
4. Save the file and upload to your Web server.

continues on page 372

Script 9.15 *continued*

script

```
29          $query = "SELECT user_id, first_name FROM users WHERE email='$e' AND password=SHA('$p')";
30          $result = @mysql_query ($query); // Run the query.
31          $row = mysql_fetch_array ($result, MYSQL_NUM); // Return a record, if applicable.
32
33          if ($row) { // A record was pulled from the database.
34
35             // Set the session data & redirect.
36             session_name ('YourVisitID');
37             session_start();
38             $_SESSION['user_id'] = $row[0];
39             $_SESSION['first_name'] = $row[1];
40             $_SESSION['agent'] = md5($_SERVER['HTTP_USER_AGENT']);
41
42             // Redirect the user to the loggedin.php page.
43             // Start defining the URL.
44             $url = 'http://' . $_SERVER['HTTP_HOST'] . dirname($_SERVER['PHP_SELF']);
45             // Check for a trailing slash.
46             if ((substr($url, -1) == '/') OR (substr($url, -1) == '\\') ) {
47                $url = substr ($url, 0, -1); // Chop off the slash.
48             }
49             // Add the page.
50             $url .= '/loggedin.php';
51
52             header("Location: $url");
53             exit(); // Quit the script.
54
55          } else { // No record matched the query.
56             $errors[] = 'The email address and password entered do not match those on file.';
               // Public message.
57             $errors[] = mysql_error() . '<br /><br />Query: ' . $query; // Debugging message.
58          }
59
60       } // End of if (empty($errors)) IF.
61
62       mysql_close(); // Close the database connection.
63
64    } else { // Form has not been submitted.
65
66       $errors = NULL;
67
```

(script continues on next page)

Script 9.15 *continued*

script

```
} // End of the main Submit conditional.

// Begin the page now.
$page_title = 'Login';
include ('./includes/header.html');

if (!empty($errors)) { // Print any error messages.
    echo '<h1 id="mainhead">Error!</h1>
    <p class="error">The following error(s) occurred:<br />';
    foreach ($errors as $msg) { // Print each error.
        echo " - $msg<br />\n";
    }
    echo '</p><p>Please try again.</p>';
}

// Create the form.
?>
<h2>Login</h2>
<form action="login.php" method="post">
    <p>Email Address: <input type="text" name="email" size="20" maxlength="40" /> </p>
    <p>Password: <input type="password" name="password" size="20" maxlength="20" /></p>
    <p><input type="submit" name="submit" value="Login" /></p>
    <input type="hidden" name="submitted" value="TRUE" />
</form>
<?php
include ('./includes/footer.html');
?>
```

5. Open `loggedin.php` (Script 9.11) in your text editor.

6. Change the `!isset($_SESSION['user_id'])` conditional to (**Script 9.16**)

```
if (!isset($_SESSION['agent'])
→ OR ($_SESSION ['agent'] != md5
→ ($_SERVER['HTTP_USER_AGENT'])) ) {
```

This conditional checks for two things. First, it sees if the `$_SESSION['agent']` variable is not set (this part is just as it was before, although *agent* is being used instead of *user_id*). The second part of the conditional checks if the `md5()` version of `$_SERVER['HTTP_USER_AGENT']` does not equal the value stored in `$_SESSION['agent']`. If either of these conditions are true, the user will be redirected.

Script 9.16 This `loggedin.php` script now confirms that the user accessing this page has the same *HTTP_USER_AGENT* as they did when they logged in.

script

```
1   <?php # Script 9.16 - loggedin.php
    (4th version after Scripts 9.2, 9.7 & 9.11)
2   # User is redirected here from login.php.
3
4   session_name ('YourVisitID');
5   session_start(); // Start the session.
6
7   // If no session value is present, redirect
    the user.
8   if (!isset($_SESSION['agent']) OR
    ($_SESSION ['agent'] != md5($_SERVER
    ['HTTP_USER_AGENT'])) ) {
9
10      // Start defining the URL.
11      $url = 'http://' . $_SERVER['HTTP_HOST']
        . dirname($_SERVER['PHP_SELF']);
12      // Check for a trailing slash.
13      if ((substr($url, -1) == '/') OR (substr
        ($url, -1) == '\\') ) {
14          $url = substr ($url, 0, -1); // Chop
            off the slash.
15      }
16      $url .= '/index.php'; // Add the page.
17      header("Location: $url");
18      exit(); // Quit the script.
19  }
20
21  // Set the page title and include the HTML
    header.
22  $page_title = 'Logged In!';
23  include ('./includes/header.html');
24
25  // Print a customized message.
26  echo "<h1>Logged In!</h1>
27  <p>You are now logged in, {$_SESSION
    ['first_name']}!</p>
28  <p><br /><br /></p>";
29
30  include ('./includes/footer.html');
31  ?>
```

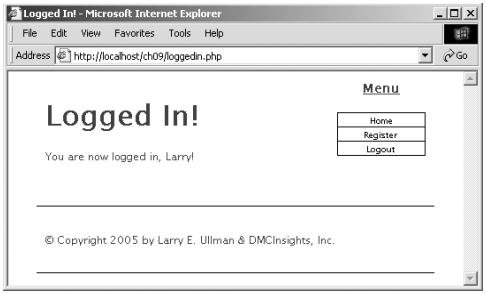

Figure 9.22 You cannot tell any difference by running the application, but this final version of the login system is more secure. Specifically, it helps to prevent session hijacking.

7. Save this file, upload to your Web server, and test in your Web browser by logging in (**Figure 9.22**).

✔ Tips

- For critical uses of sessions, require the use of cookies and transmit them over a secure connection, if at all possible. You can even set PHP to only use cookies by setting *session.use_only_cookies* to *1* (as of PHP 4.3).
- If you are using a server shared with other domains, changing the *session.save_path* from its default setting—which is accessible by all users—to something more local will be more secure.
- On the server side of security, the session data itself can be stored in a database rather than a text file. This is a more secure, but more programming-intensive, option.
- The user's IP address (the network address from which the user is connecting) is *not* a good unique identifier, for two reasons. First, a user's IP address can, and normally does, change frequently (ISP's dynamically assign them for short periods of time). Second, many users accessing a site from the same network (like a home network or an office) could all have the same IP address.