**Controlling Logins**

Possibly the most important feature of the back-end application is security control. It's imperative that you're able to control who can log in and modify data in your store. It's also a good idea not to provide too much information before a visitor actually logs into the application. For example, you wouldn't want to show menu options that only logged-in users would see.

The Food Store admin application uses *session cookies* to track logged-in managers. A session cookie holds data for a Web session that persists between Web pages. After the application validates the manager from the *admins* database table, it creates a session cookie with the user name for the manager.

We'll use that session cookie to determine what content a visitor will see on the admin Web page. If the session cookie isn't present, we won't produce any data. This is a great technique for controlling information on your Web page.

When our store manager first accesses the admin application, we'll need to present a bare-bones Web page that only has a login form, as shown here.

The Food Store admin login page

Notice that the navigation area (the left side of the page) is almost empty. All of the links to features within the application are hidden when the visitor isn't logged in. Let's build the code for that:

1. Create a file called *adminnav.inc.php* in the admin folder.
2. Enter the following code in the file:

<table width="100%" cellpadding="2">

<tr>

<td><h3>Store Administration</h3></td>

</tr>

<tr>

<td><a href="admin.php"><strong>Home</strong></a></td>

</tr>

<tr>

<td><hr size="1" noshade="noshade" /></td>

</tr>

<?php

if (isset($\_SESSION['store\_admin']))

{

echo "<tr><td>\n";

echo "<form action=\"admin.php\" method=\"get\">\n";

echo "<label><font color=\"#663300\"><br>Browse Products<br></font> </label>\n";

echo "<select name=\"cat\">\n";

$query="SELECT catid,name from categories";

$result=mysql\_query($query);

while($row=mysql\_fetch\_array($result,MYSQL\_ASSOC))

{

$catid = $row['catid'];

$name = $row['name'];

echo "<option value=\"$catid\">$name</option>";

}

echo "</select>\n";

echo "<input name=\"goButton\" type=\"submit\" value=\"browse\" />\n";

echo "<input name=\"content\" type=\"hidden\" value=\"editproducts\" />\n";

echo "</form> </td></tr>\n";

echo "<tr><td><hr size=\"1\" noshade=\"noshade\" /></td></tr>\n";

echo "<tr><td bgcolor=\"#FFFF99\"><a href=\"admin.php?content=newproduct\"><strong>Add a new product</strong></a></td></tr>\n";

echo "<tr><td><hr size=\"1\" noshade=\"noshade\" /></td></tr>\n";

echo "<tr><td><a href=\"admin.php?content=newcat\"><strong>Add a new category</strong></a></td></tr>\n";

echo "<tr><td><hr size=\"1\" noshade=\"noshade\" /></td></tr>\n";

echo "<tr><td><a href=\"admin.php?content=process\"><strong>Process Pending Orders</strong></a></td></tr>\n";

echo "<tr><td><hr size=\"1\" noshade=\"noshade\" /></td></tr>\n";

echo "<tr><td><a href=\"admin.php?content=outofstock\"><strong>List out-of-stock</strong></a></td></tr>\n";

echo "<tr><td><hr size=\"1\" noshade=\"noshade\" /></td></tr>\n";

echo "<tr><td><a href=\"admin.php?content=report\"><strong>Generate report</strong></a></td></tr>\n";

echo "<tr><td><hr size=\"1\" noshade=\"noshade\" /></td></tr>\n";

echo "<tr><td><a href=\"logout.php\"><strong>Log Out</strong></a></td></tr>\n";

echo "<tr><td> </td></tr>\n";

}

?>

</table>

1. Save the file, and exit Notepad.

The adminnav.inc.php code creates a simple one-column HTML table in the navigation section of the Web page. In the top row, it displays a title and then a link to the main administration page.

The meat of the adminnav.inc.php page is in the PHP code. The PHP code uses an if-then statement to check if the *store\_admin* session cookie is set. We'll use this session cookie to track our manager.

If the cookie isn't set, none of the navigation menu items will appear. In that case, we'll just exit the if-then statement and close out the table. If the session cookie is set, we know the manager is logged in, and we show the links for the individual sections of the administration application (which we'll be discussing later on). How cool is that?

**Providing a Login Form**

The next step is to provide a simple form that allows the manager to log in. If you take a look back at the admin.php code you created in Chapter 2, you'll notice that the main section of the Web page uses some PHP code to determine the content it presents:

<?php

if (!isset($\_REQUEST['content']))

{

if (!isset($\_SESSION['store\_admin']))

include("adminlogin.html");

else

include("adminmain.inc.php");

}

else

{

$content = $\_REQUEST['content'];

$nextpage = $content . ".inc.php";

include($nextpage);

} ?>

The application uses an HTML *content* variable to determine the information to display in the main section of the Web page. This trick allows us to use the same overall Web page format and easily modify the central content just by calling a different include file. If the content variable isn't set (such as when you first access the Web page), the code performs another check. It checks to see if the visitor has the *store\_admin* session cookie set in their browser session.

If the cookie is set, it means the visitor is already logged into the application, and he or she is sent to the next Web page defined in the content HTML variable. If the cookie isn't set, the visitor isn't logged in. So the main section includes the file *adminlogin.html*.

The adminlogin.html file is a simple HTML file that produces our login form. Let's create that file.

1. Create a file called *adminlogin.html* in the admin folder (note that it is a .html file and not a .php file.
2. Enter the following code:

<h2><br>Store Web Site Administration<br></h2>

<p>Please log in below</p>

<form action="validate.php" method="post">

<b>User Name:</b><br>

<input type="text" size="20" name="userid"><br>

<br>

<b>Password:</b><br><input type="password" size="20" name="password">

<br><br>

<input type="submit" value="Login">

</form>

This is a simple HTML form asking for the user name and password for our administrator. After the manager clicks the Login button, the page passes the form information to the validate.php file using the HTML POST method. All the action happens here.

The validate.php file isn't really part of our admin Web page. You'll notice that the login form doesn't return us to the admin.php program to display content in the main Web page. Instead, it passes control directly to the validate.php page. The code in the validate.php file checks the login information provided by the manager. Then it forwards the session to the admin.php page, but it also specifies the name of the next Web page to display.

Let's create the validate.php file:

* 1. Create a file called *validate.php* in the admin folder.
  2. Enter the following code:

<?php

session\_start();

include ("/mylibrary/login.php");

login();

$userid = $\_POST['userid'];

$password = $\_POST['password'];

$query = "SELECT userid, name from admins where userid = '$userid' and password = PASSWORD('$password')";

$result = mysql\_query($query);

if (mysql\_num\_rows($result) == 0)

{

echo "<h2>Sorry, your account was not validated.</h2><br>\n";

echo "<a href=\"admin.php\">Try again</a><br>\n";

} else

{

$\_SESSION['store\_admin'] = $userid;

header("Location: admin.php");

}

?>

* 1. Save the file and exit.

The validate.php file retrieves the data from the HTML form using the standard PHP $\_POST[] array variable and stores the values in PHP variables. Next, it uses those values to create an SQL query to compare the user name and password the manager supplied to the values in the admins database table. If the values don't match an existing record, the code displays a simple error message, along with a link back to the login page.

The cool part happens when the user data is validated.

$\_SESSION['store\_admin'] = $userid;

header("Location: admin.php");

First, the code creates a *store\_admin* session cookie (remember, that's the session cookie name the code in the adminnav.inc.php file is looking for) and stores the user name there. The next line does some amazing trickery.

The PHP *header()* function allows us to send HTTP header information to the client browser. HTTP headers control the operation of how the client's browser interprets data in the session. The trick is this function must appear before any HTML code is sent to the client's browser. You can't have any echo statements before the header() function, or it'll fail.

HTTP headers provide for lots of control over the Web session. The HTTP *Location* header redirects the client browser to another page. In this particular case, we want to return the logged-in manager back to the admin.php page so he or she can now see the full administration Web page.

Now that your store manager is logged into the application, it's time to provide some useful information. We'll take a look at this in Chapter 4.