Chapter 6

PHP Control and File Handling

Key Skills & Concepts

- `if/else` Statements
- `while` and `do-while` Statements
- `for` and `foreach` Statements
- `switch` Statements
- Basic File Functions
- Additional File Functions
- Session Variables
- Cookies
- Server Variables

PHP is often thought of as simply the mechanism for working with a MySQL database. It is, though, a full-featured programming language with all the features you saw in Chapter 5, as well as a full set of control structures and powerful file-handling tools, browser information features, and techniques for creating and using cookies and session variables. This chapter explores these powerful capabilities of PHP and in the process describes how to use `if-else` and `switch` statements and the conditions that control them, along with `for`, `while`, and `do` loops. The chapter also looks at PHP file management features and functions, the ability to determine the type of browser being used, and how to establish session variables that can protect access to a website. Finally, passing data among web pages using session variables and cookies is discussed.

TIP

It is useful to have the PHP Manual and the PHP Function Reference bookmarked or set up as favorites in your browser, if you don't already, so you can quickly refer to them. The manual is at php.net/manual/en/, and the function reference is at php.net/manual/en/funcref.php.

Control Structures

The scripts so far in this section have been executed from the first statement to the last statement without interruption or change of direction, with the exception of repeated function calls. Often, you will want to ask if the script should go one way or another, or go back and re-execute a particular piece of code. That is the purpose of control structures, which include `if/else` statements; `while`, `do-while`, `for` statements; and `switch` statements.

`if/else` Statements

`if/else` statements are the primary decision-making construct in PHP. They allow you to specify that `if` some expression is `TRUE`, then a group of statements will be executed, `else` a different group of statements will be executed. It takes this form:

```
if (conditional expression) {
   statements executed if TRUE;
   }
else   {
   statements executed if FALSE;
   }
```

The `else` group of statements is optional and is needed only if you want to do something other than continue with the script if the conditional expression is `FALSE`. Also, you can nest `if/else` statements using `elseif`, like this:

```
if (conditional expression) {
   statements executed if TRUE;
   }
elseif (second conditional expression) {
   statements executed if second conditional is TRUE;
   }
else {
   statements executed if second conditional is FALSE;
   }
```

In all cases, the conditional expression must result in a Boolean `TRUE` or `FALSE` (1 or 0). If a variable simply exists, that is, it has been defined as containing something other than `NULL`, `FALSE`, or `0`, then it is `TRUE`.

TIP

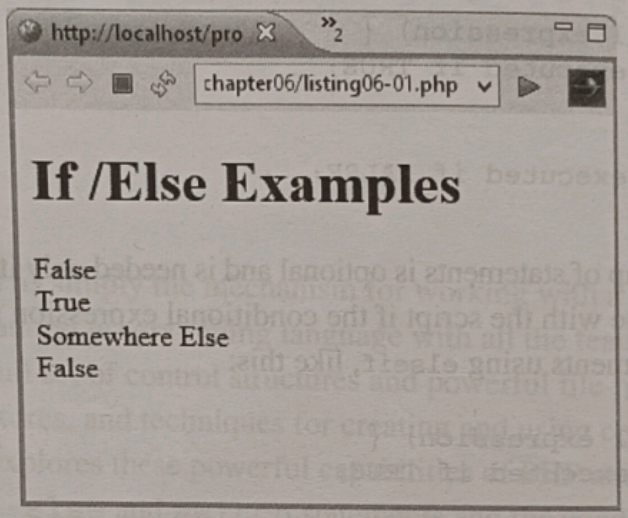
If you define a variable as containing a constant like `NULL`, `TRUE`, or `FALSE`, you must remember *not* to put it in quotation marks.

Many conditional expressions are comparisons that test if two elements are equal, greater than, or less than. Remember that when you test for equality in PHP, you must use a double equal sign (==), not a single one, which means assignment.

TIP

If you want to test for a condition *not* being true, put an exclamation mark (!) in front of the condition. For example, `(! $a < 10)` is true when `$a` is equal to or greater than 10.

Listing 6-1 shows several examples of `if/else` statements, the results of which are shown next. A number of examples will be shown in the following chapters.



Listing 6-1 `if/else` Statements

```
<html>
   <head>
      <title>Listing 6-1</title>
   </head>
   <body>
      <h1>If/Else Examples</h1>
      <?php
         if ($a){  //Tests if $a has been defined
             and is not FALSE or 0
            echo "True", "<br />";
         }
```

```
        else {
            echo "False", "<br />";
        }
        $a = "Something";
        if ($a){
            echo "True", "<br />";
        }
        else {
            echo "False", "<br />";
        }
        $state = "CA";
        if ($state == "WA" ){
            echo "Pacific Northwest", "<br />";
        }
        elseif ($state == "OR") {
            echo "Pacific Northwest", "<br />";
        }
        else {
            echo "Somewhere Else", "<br />";
        }
        echo $b ? "True" : "False";
    ?>
  </body>
</html>
```

Ternary Operator

A shorthand method of doing `if/else` decision making in PHP scripts uses the ternary operator (`? :`), where `?` replaces the `if` test and follows the conditional expression, and the `:` replaces `else`. The following statement produces the same results as the `if` statement that follows it:

```
echo $a ? "True" : "False", "<br />";
if ($a){
    echo "True", "<br />";
    }
else {
    echo "False", "<br />";
    }
```

Joining several `if` statements, as you would with `elseif`, is not recommended with the ternary operator since PHP's behavior is not defined.

TIP

Use parentheses if you want to use multiple ternary operators.

while and do-while Statements

The while and do-while statements are looping constructs that allow you to repeatedly execute a piece of code until a conditional expression is no longer TRUE. The while statement is the foundation of this set of statements and takes one of the following forms:

```
while (conditional expression) {
      statements executed while TRUE;
      }

while (conditional expression) :
      statements executed while TRUE;
      endwhile;
```

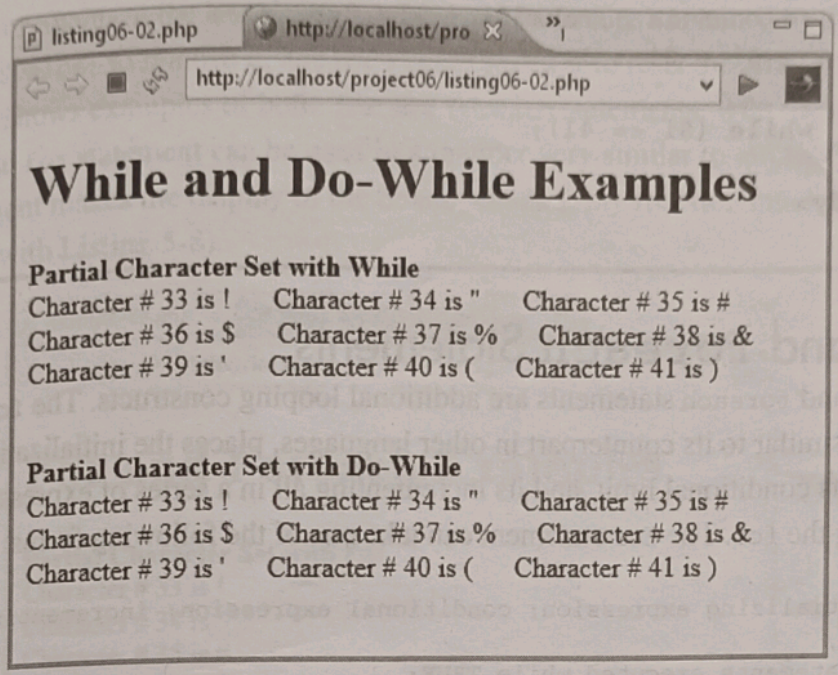
The do-while statement is similar to the while statement, except that the conditional expression is at the end of the statement instead of the beginning. The do-while statement takes this form:

```
do {
   statements executed while TRUE;
   }
while (conditional expression);
```

The most common conditional expression is to compare a counter with some end value. In other words, to initialize a counter and then to loop though some statements, incrementing the counter with each loop, until the counter exceeds the end value. Listing 6-2 shows examples of this for both while and do-while using the array function chr(), which returns a character based on its number in the standard set of characters, with the results shown next. You can see that in this case there is no difference between while and do-while.

NOTE

Code in do-while is guaranteed to execute at least once, since the condition isn't checked until after the first execution. A while statement may not ever execute if the condition immediately evaluates to false.



Listing 6-2 `while` and `do-while` Examples

```
<html>
   <head>
      <title>Listing 6-2</title>
   </head>
   <body>
      <h1>While and Do-While Examples</h1>
      <?php
         echo "<b>Partial Character Set with While</b><br />";
         $i = 33;
         while ($i <= 41) {
            echo "Character # ", $i , " is ", chr($i);
            echo "      Character # ", $i + 1 ,
               " is ", chr($i + 1);
            echo "      Character # ", $i + 2 ,
               " is ", chr($i + 2), "<br />";
            $i = $i + 3;
         }
         echo "<br /> <br /> <b>Partial Character Set with
            Do-While</b><br />";
         $i = 33;
         do {
            echo "Character # ", $i , " is ", chr($i);
            echo "      Character # ", $i + 1 ,
               " is ", chr($i + 1);
            echo "      Character # ", $i + 2 ,
```

```
            " is ", chr($i + 2), "<br />";
        $i = $i + 3;
    }
    while ($i <= 41);
?>
</body>
</html>
```

for and foreach Statements

The for and foreach statements are additional looping constructs. The for statement, which is similar to its counterpart in other languages, places the initialization of the counter, its conditional limit, and its incrementing all in a series of expressions immediately following the for. The for statement can take one of the following forms:

```
for (initializing expression; conditional expression; incrementing expression)
    {
    statements executed while TRUE;
    }
for (initializing expression; conditional expression; incrementing expression):
    statements executed while TRUE;
endfor;
```

In its basic form, the for expression might be `for ($i = 1; $i <= 5; $i++)`, where `$i++` increments `$i` after it is used. If any of these expressions are handled elsewhere in the script, they can be left blank in the for expression, which at its minimum is `for ( ; ; )`.

CAUTION

If the conditional expression never evaluates to true, the loop will never end.

The foreach statement is used to iterate through arrays and objects and cannot be used on any other type of variable. It can display the value of each element in an array, or alternatively, it can display the key and the value of each element in an array. Therefore, it has one of the two following forms:

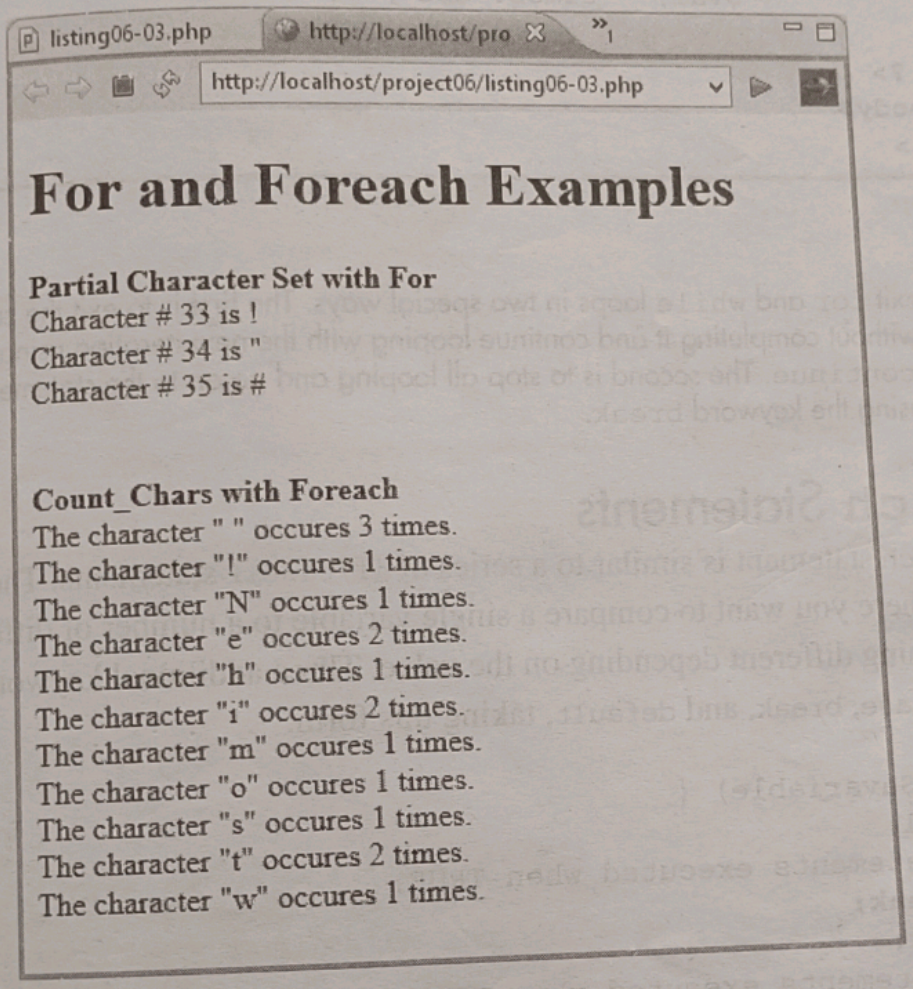
```
foreach ($array as $value) {
    statements executed for each element;
    }

foreach ($array as $key => $value) {
    statements executed for each element;
    }
```

When you first begin to execute a foreach statement, the array pointer is automatically reset to the first element in the array. When you end the execution of a foreach statement,

the array pointer remains at the last element, whose value remains contained by `$value`. You can use `unset($value)` to remove it, and use `reset($array)` to reset the array pointer.

Listing 6-3 shows examples of both `for` and `foreach` statements, with the results shown next. The `for` statement can be used in a manner very similar to `while`. The `foreach` statement makes the display of the `count_chars` array function much easier to read (compare with Listing 5-8).



Listing 6-3 `for` and `foreach` Examples

```
<html>
  <head>
    <title>Listing 6-3</title>
  </head>
  <body>
    <h1>For and Foreach Examples</h1>
    <?php
      echo "<b>Partial Character Set with For</b><br />";
      for ($i = 33; $i <= 35; $i++) {
```

```
            echo "Character # ", $i , " is ", chr($i), "<br />";
        }
        echo "<br /> <br /> <b>Count_Chars with Foreach</b><br />";
        $textString = "Now is the time!";
        $textArray = count_chars($textString, 1);
        foreach ($textArray as $key => $val) {
            echo "The character \"", chr($key) , "\" occurs ",
                $val, " times. <br />";
        }
    ?>
  </body>
</html>
```

TIP

You can exit `for` and `while` loops in two special ways. The first is to exit the current iteration without completing it and continue looping with the next iteration using the keyword `continue`. The second is to stop all looping and go on to the statement after the loop using the keyword `break`.

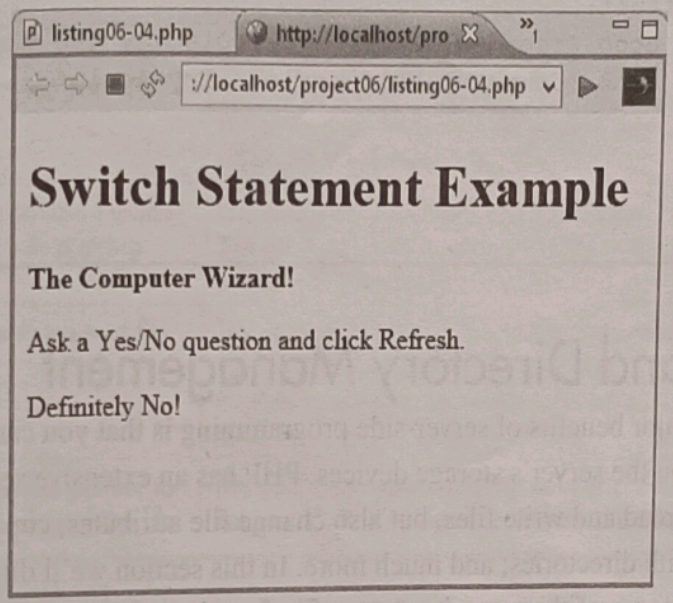
`switch` Statements

The `switch` statement is similar to a series of `if/elseif` statements. The `switch` statement is used where you want to compare a single variable to a number of different values and do something different depending on the value. Three additional keywords are used with `switch`: `case`, `break`, and `default`, taking this form:

```
switch ($avariable) {
   case 1:
      statements executed when TRUE;
      break;
   case 2:
      statements executed when TRUE;
      break;
   case 3:
      statements executed when TRUE;
      break;
   default:
      statements executed when all are FALSE;
}
```

Each `case` expression in the `switch` statement compares the `switch` variable with the `case` value, which can be a string; if it is equal, the statements following the `case` expression are executed, and then the `break` expression sends the script's flow to the first statement after the `switch`'s closing curly brace. If none of the `case` expressions is

successful, the statements following the `default` expression are executed, and the script's flow exits the `switch` statement. Listing 6-4 demonstrates how this works using the random number generator function `rand()`, which provides these results:



Listing 6-4 `switch` Statement Example

```
<html>
   <head>
      <title>Listing 6-4</title>
   </head>
   <body>
      <h1>Switch Statement Example</h1>
      <p><b>The Computer Wizard!</b></p>
      <p>Ask a Yes/No question and click Refresh.</p>
      <?php
         $randNum = rand(1,5);
         switch ($randNum) {
         case 1:
            echo "Definitely Yes!";
            break;
         case 2:
            echo "Probably Yes!";
            break;
         case 3:
            echo "Definitely Maybe!";
            break;
         case 4:
            echo "Probably No!";
```

```
            break;
        case 5:
            echo "Definitely No!";
            break;
        default:
            echo $randNum, "<br />"; //Display the number
            echo "Computer Malfunction, Try Again";
    }
?>
</body>
</html>
```

PHP File and Directory Management

One of the major benefits of server-side programming is that you can read and write information on the server's storage devices. PHP has an extensive set of functions that let you not only read and write files, but also change file attributes; copy, move, and delete files; work with directories; and much more. In this section we'll discuss and briefly demonstrate some of the more important file functions. In the next chapter, I'll show how to create a user authentication system using the file commands.

Basic File Functions

The basic PHP file process has the following elements:

- Establish a file connection, or *file pointer* (also called a "handle"), between PHP and a file using the `fopen()` function.
- Write a data string to an opened file by using the `fwrite()` function.
- Read a certain number of bytes into a string from an opened file by using the `fread()` function.
- Terminate a file pointer with the `fclose()` function.

Table 6-1 provides more information about these basic file functions, and Listing 6-5 provides a brief demonstration of how these functions are used.

Function	Description	Explanation
`fclose()`	Terminates a file pointer	The only argument is the file pointer created with `fopen()`. Returns `TRUE` for a successful close and `FALSE` otherwise.
`fopen()`	Connects to a file and creates a file pointer	The required arguments are a string with the path and filename, and a string with the mode (see Table 6-2 for the list of modes). Returns a file pointer that can be used with other file functions or `FALSE`.
`fread()`	Reads a certain number of bytes into a string	The required arguments are the file pointer created with `fopen()` and the number of bytes to be read. Returns a string with data read or `FALSE`.
`fwrite()`	Writes a string to a file	The required arguments are the file pointer created with `fopen()` and the string to be written. Optionally, the number of bytes to be written can be added. Returns the number of bytes written or `FALSE`.
`rewind()`	Resets the file pointer to the beginning of the file	The only argument is the file pointer created with `fopen()`. Returns `TRUE` if successful, and `FALSE` otherwise.

Table 6-1 Basic File Functions

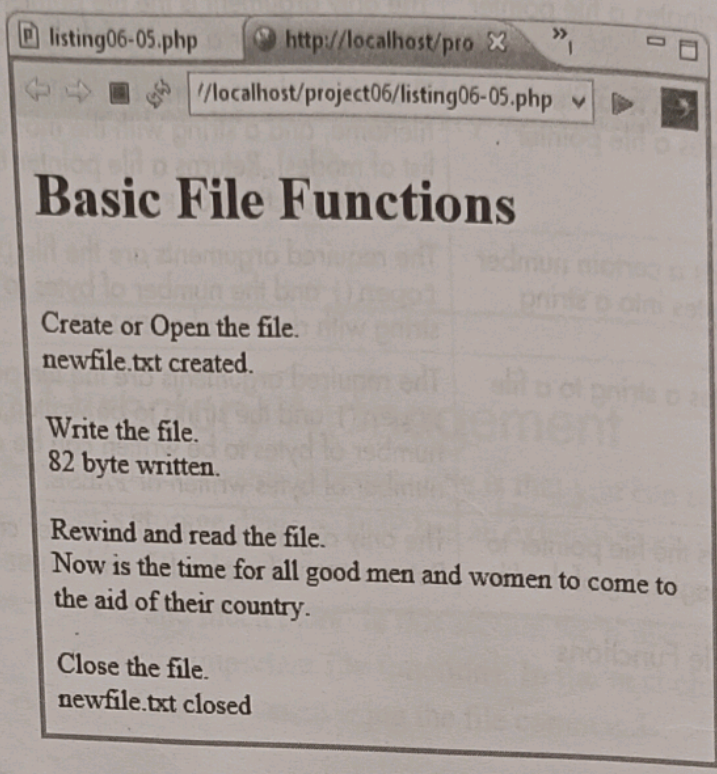
TIP

The standard name often used for the file pointer variable is `$fp`.

Mode	Explanation
`a`	Write only, starting at the end of the file or creating a new file, appending new information to what previously existed in the file.
`a+`	Write and read, starting at the end of the file or creating a new file, appending new information to what previously existed in the file.
`r`	Read only, from the beginning of the file.
`r+`	Read and write, from the beginning of the file.
`w`	Write only, after deleting any file contents or creating a new file.
`w+`	Write and read, after deleting any file contents or creating a new file.
`x`	Create a new file and write only, from the beginning of the file. Returns `FALSE` if the file exists.
`x+`	Create a new file and write and read, from the beginning of the file. Returns `FALSE` if the file exists.

Table 6-2 `fopen()` Modes

Figure 6-1 proves that the script in Listing 6-5 does work, providing this response online:



NOTE

You see the multiple lines in the file shown in Figure 6-1 only if you run the script multiple times. The `a+` parameter of `fopen` allows this to happen.

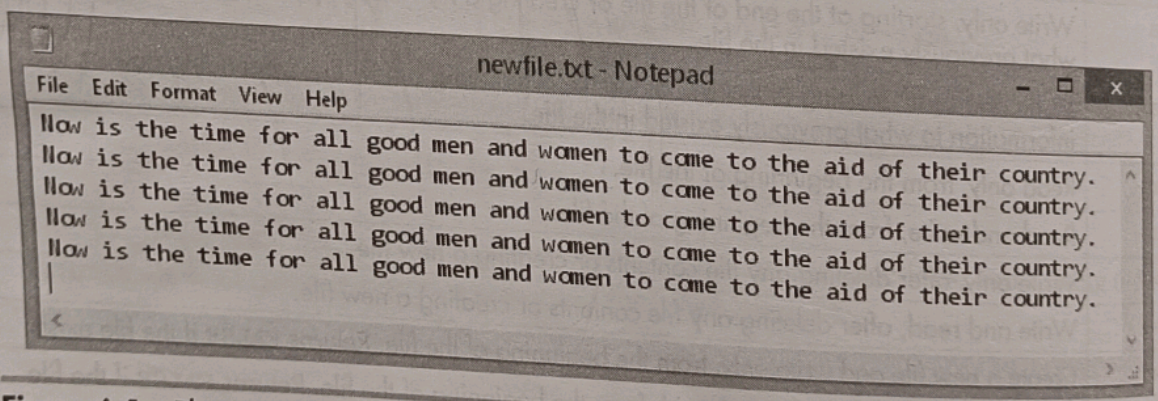


Figure 6-1 Placing `\r\n` (carriage return, linefeed) at the end of each line causes the file to display as separate lines instead of one continuous string of text. It also can be used to read a line at a time.

Listing 6-5 Basic File Functions

```
<html>
   <head>
      <title>Listing 6-5</title>
   </head>
   <body>
      <h1>Basic File Functions</h1>
      <?php
         echo "<br />", "Create or Open the file.", "<br />";
            $fp = fopen("newfile.txt", "a+");
            if ($fp){
               echo "newfile.txt created.", "<br />";
            }
            else {
               echo "newfile.txt cannot be opened.", "<br />";
            }
         echo "<br />", "Write the file.", "<br />";
            $bytes = fwrite ($fp, "Now is the time for all good men
            and women to come to the aid of their country. \r\n");
            if ($bytes){
               echo $bytes, " byte written.", "<br />";
            }
            else {
               echo "File not written.", "<br />";
            }
         echo "<br />", "Rewind and read the file.", "<br />";
            rewind($fp);
            $data = fread ($fp, $bytes);
            if ($data){
               echo $data, "<br />";
            }
            else {
               echo "File not read.", "<br />";
            }
         echo "<br />", "Close the file.", "<br />";
            if (fclose ($fp)) {
               echo "newfile.txt closed", "<br />";
            }
            else {
               echo "newfile.txt not closed.", "<br />";
            }
      ?>
   </body>
</html>
```

TIP

As your scripts get longer, it becomes very tempting to close up your code, such as putting `if` statements on one line and not indenting. The problem with that is the code becomes harder to visually check and therefore is prone to problems.

Additional File Functions

Although `fread()` and `fwrite()` are very functional, as you've seen, PHP provides a number of other ways to read and write data on a server's storage devices. Both `fread()` and `fwrite()` are aimed at reading or writing parts of a file to or from strings. PHP can also read to an array, read a character at a time, read a line at a time, and move around within a file using the functions described in Table 6-3, which are demonstrated in Listing 6-6, the results of which are shown in Figure 6-2.

Function	Description	Explanation
`feof()`	Determines if the pointer is at end of file (EOF)	The only argument is the file pointer created with `fopen()`. Returns `TRUE` if the EOF is reached and `FALSE` otherwise.
`fgetc()`	Reads a single character from an open file	The only argument is the file pointer created with `fopen()`. Returns a string with the character read or `FALSE` if the EOF is reached.
`fgets()`	Reads a line designated by `\n` from an open file	The required argument is the file pointer created with `fopen()` and, optionally, the number of bytes to be read. Returns a string with data read or `FALSE`.
`file()`	Reads an entire file into an array, where each element is a line designated by `\n`	The required argument is a string with the path and filename. The default is to read as binary data, but you can change that with the `FILE_TEXT` constant as the second argument.
`filesize()`	Determines the size of a file	The only argument is a string with the path and filename. Returns the file size in bytes or `FALSE`.
`fseek()`	Moves the file pointer of an open file	The required arguments are the file pointer created with `fopen()` and the number of bytes to move the pointer. The default is to move from the beginning of the file, but adding the `SEEK_CUR` constant as the third argument adds the number of bytes to the current position, while using the `SEEK_END` constant as the third argument *adds* the bytes to the EOF. Returns a 0 for success; otherwise, a –1.
`ftell()`	Provides the current position of the file pointer of an open file	The only argument is the file pointer created with `fopen()`. Returns the number of bytes from the beginning of the file as an integer, or `FALSE`.

Table 6-3 Additional File Functions

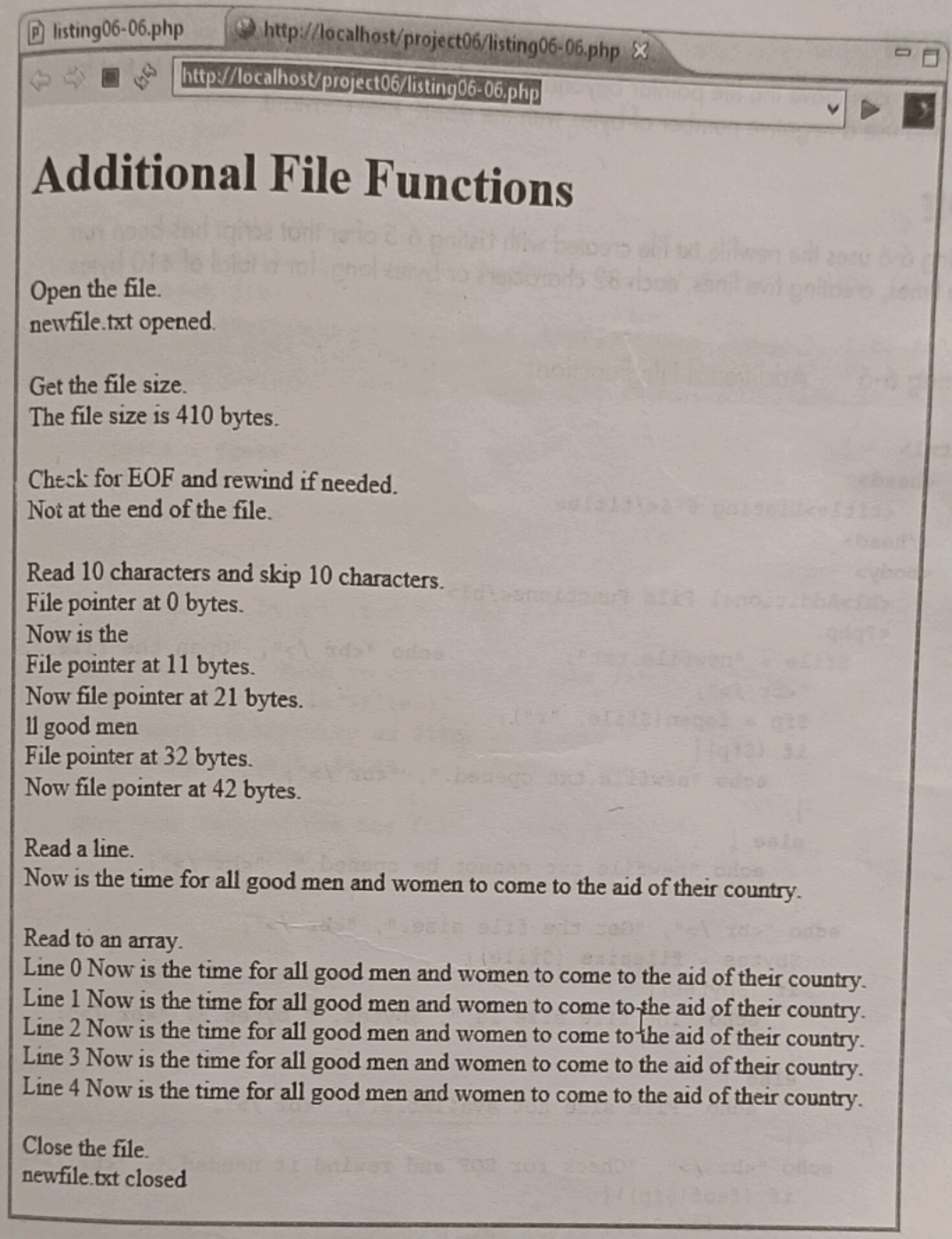


Figure 6-2 PHP gives you a number of ways to read and roam about a file.

TIP

You can read an entire file using the `filesize()` function, as you can see in the following code snippet:

```
$filename = "newfile.txt";
$fp = fopen($filename, "r");
$wholefile = fread($fp, filesize($filename));
```

CAUTION

`fseek()` can move the file pointer beyond the end of a file, so you may want to test for that and use a negative number of bytes with the `SEEK_END` constant.

NOTE

Listing 6-6 uses the newfile.txt file created with Listing 6-5 after that script has been run five times, creating five lines, each 82 characters or bytes long, for a total of 410 bytes.

Listing 6-6 Additional File Functions

```
<html>
   <head>
      <title>Listing 6-6</title>
   </head>
   <body>
      <h1>Additional File Functions</h1>
      <?php
         $file = "newfile.txt";          echo "<br />", "Open the file.",
            "<br />";
            $fp = fopen($file, "r");
            if ($fp){
               echo "newfile.txt opened.", "<br />";
            }
            else {
               echo "newfile.txt cannot be opened.", "<br />";
            }
         echo "<br />", "Get the file size.", "<br />";
            $bytes = filesize ($file);
            if ($bytes){
               echo "The file size is ", $bytes, " bytes.", "<br />";
            }
            else {
               echo "File size not available.", "<br />";
            }
         echo "<br />", "Check for EOF and rewind if needed.", "<br />";
            if (feof($fp)){
               rewind($fp);
               echo "Returned to the beginning of the file.", "<br />";
            }
            else {
               echo "Not at the end of the file.", "<br />";
            }
         echo "<br />", "Read 10 characters and skip 10 characters.",
            "<br />";
            echo "File pointer at  ", ftell($fp), " bytes.", "<br />";
```

```
        while (ftell($fp) < 42) {
            while ($i < 11) {
                $char = fgetc ($fp);
                echo $char ;
                $i++ ;
            }
            $i = 0;
            echo "<br />";
            echo  "File pointer at  ", ftell($fp), " bytes.", "<br />";
            fseek($fp, 10, SEEK_CUR);
            echo "Now file pointer at ", ftell($fp), " bytes.", "<br />";
        }
    echo "<br />", "Read a line.", "<br />";
        rewind($fp);
        $data = fgets ($fp);
        if ($data){
            echo $data, "<br />";
        }
        else {
            echo "File not read.", "<br />";
        }
    echo "<br />", "Read to an array.", "<br />";
        $textArray = file($file);
        foreach ($textArray as $line => $text) {
            echo "Line ", $line, " ", $text, "<br />";
        }
    echo "<br />", "Close the file.", "<br />";
        if (fclose ($fp)) {
            echo "newfile.txt closed", "<br />";
        }
        else {
            echo "newfile.txt not closed.", "<br />";
        }
    ?>
  </body>
</html>
```

TIP

As Listing 6-6 shows, while you are debugging a script, it is very helpful to put in a number of echoes to display what is happening and to comment out specific sections until you isolate a problem. In the final version of the script, the echoes can become comments.

PHP file functions, which number many more than mentioned in this and the previous section, provide a comprehensive ability to work with files. To see the full set of file functions, see php.net/manual/en/ref.filesystem.php.

Cookies and Session and Server Variables

Web pages use variables to temporarily store information while that page is being used. It is likely, though, that you will want to collect and store information from the user, from the script being run, and from the server and to use that information on other pages during the current session, or across multiple sessions. In this section, we'll discuss creating and using session variables, cookies, and server variables to perform that function of collecting and storing. Start by comparing the three objects in Table 6-4.

Session variables, cookies, and server variables are all PHP predefined, or built-in, variables called *superglobals* because they are available throughout a website on as many pages as are contained in the domain. The superglobals used here are `$_SESSION`, `$_COOKIE`, and `$_SERVER`. Their use will be explained in the following sections. All the superglobals discussed in this book have the `$_` leading characters and here, as well as in many other reference sources, are displayed in all capital letters, although that is not a requirement. You'll see other superglobals later in this book.

Superglobals are *associative arrays,* arrays whose key, or array index, is a string instead of an integer. Generally, the *key* is an element name, while the *value* is the element value. For example: `$_SESSION["name"] = $name;` where `["name"]` is the *key* for the array element and the contents of `$name` is its *value.* Some superglobal arrays, such as `$_SESSION` and `$_COOKIE`, allow you to define the keys, while others, such as `$_SERVER`, have predefined keys. You will see examples of these in the sections that follow.

TIP

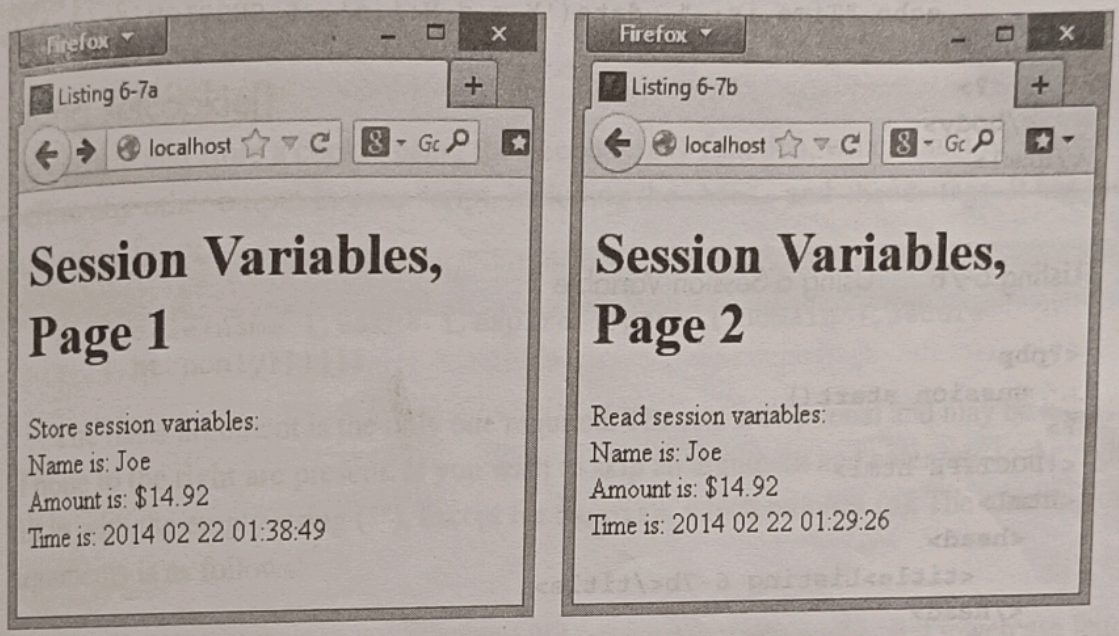
When you are entering the string for an associative array's key, you can do it without the quotes. However, this is a bad practice because PHP considers the string with quotes to be a different entity than the string without quotes, so it is very easy to confuse the two, creating an error in the script that is hard to find.

Object	Created With	Accessed With	Stored On	Duration
Session variable	`$_SESSION`	`$_SESSION`	Server	Session or less
Cookie	`setcookie()`	`$_COOKIE`	Client	Until expires
Server variable	Automatically	`$_SERVER`	Server	Unlimited

Table 6-4 Comparison of External Variables

Session Variables

Session variables allow you to collect and use information across multiple pages in a website during a single *session*, or encounter between a user and a website. To define a session, you must start it with the `session_start()` function, which must precede all other code sent to the client, including `<html>` and `<head>`. You can then store information in the `$_SESSION` array. If you go to another page and again start it with the `session_start()` function, you can retrieve the information in the `$_SESSION` array. Listing 6-7a and Listing 6-7b demonstrate this with the results shown here:



TIP

Testing session variables as described in "Session Variables" should be done with an independent browser and not the browser within Zend Studio. Also, using *http://localhost/* with some Apache servers may not work, although it does with the latest Zend Server. If you have a problem, use `http://127.0.0.1/`, and it should be fine.

Listing 6-7a Creating a Session Variable

```
<?php
   session_start()
?>
<!DOCTYPE html>
<html>
   <head>
      <title>Listing 6-7a</title>
   </head>
```

```
    <body>
      <h1>Session Variables, Page 1</h1>
      <?php
        $name = "Joe";
        $amount = "$14.92";
        echo "Store session variables:", "<br />";
          $_SESSION["name"] = $name;
          $_SESSION["amount"] = $amount;
          $_SESSION["time"]   = time();
        echo "Name is: ", $_SESSION["name"], "<br />";
        echo "Amount is: ", $_SESSION["amount"], "<br />";
        echo "Time is: ", date('Y m d H:i:s', $_SESSION["time"]),
"<br />";
      ?>
    </body>
</html>
```

Listing 6-7b Using a Session Variable

```
<?php
   session_start()
?>
<!DOCTYPE html>
<html>
   <head>
      <title>Listing 6-7b</title>
   </head>
   <body>
      <h1>Session Variables, Page 2</h1>
      <?php
         echo "Read session variables:", "<br />";
         echo "Name is: ", $_SESSION["name"], "<br />";
         echo "Amount is: ", $_SESSION["amount"], "<br />";
         echo "Time is: ", date('Y m d H:i:s', $_SESSION["time"]),
"<br />";
      ?>
   </body>
</html>
```

NOTE

Session variables have a limited life, which is set by the `session.cache_expire` *value in the pnp.ini file on the server. Depending on the use, this can be set from a few minutes to a number of hours in increments of a minute. The default is 180 minutes, or three hours.*

Cookies

Cookies store information on client computers. Once a cookie has been stored, or *set*, the same site (really *domain*) that created it will automatically receive that information the next time it is connected to the client. This allows the site to recognize a returning client. Cookies are created with the `setcookie()` function and accessed with the `$_COOKIE` superglobal array.

NOTE

Some people do not allow cookies to be stored on their computer due to security concerns.

Using setcookie()

You create and store a cookie using the `setcookie()` function, which must be placed before any other output in your script, including the `<html>` and `<head>` tags. It has the following form:

```
setcookie(name [,value [,expire [,path [,domain [,secure
   [,httponly]]]]]]
```

The `name` argument is the only one required. The rest are optional and may be left blank if none to the right are present. If you wish to skip an argument and enter one to the right, fill in with the empty string (""), except for `$expire`, which requires (`0`). The meaning of the arguments is as follows:

- **`name`** is the name of the cookie and can be any label using upper- and lowercase letters, numbers, and the underscore.
- **`value`** is the information you want stored with the cookie, as a string.
- **`expire`** is the time in seconds since 1/1/1970 that you want the cookie to expire. You can create this with the `time()` function, which will give you the number of seconds since 1/1/1970 to the current moment, and then add the number of seconds you want the cookie active. For example, `time()+(60*60*24*10)` would let the cookie be active for ten days from the current time. The default is 0, which means that the cookie will expire at the end of the session.
- **`path`** is the path on the server where the cookie will be available. If the path is set to "/", the entire domain will have access to the cookie.
- **`domain`** is the domain where the cookie will be available. If you want the cookie available to all subdomains, precede the domain with a period. For example, `.somedomain.com`.

- `secure`, if set to `TRUE`, says that the cookie should be sent only over a secure connection such as HTTPS. The default is `FALSE`.
- `httponly`, if set to `TRUE`, says the cookie should only be accessible with the HTTP protocol and not on the client via JavaScript. The default is `FALSE`.

The `value` argument can be anything that you can put in a string, such as a name or an amount. If you want to store several separate pieces of information in a cookie, you need to use several cookies to do that. From the standpoint of PHP and the server, though, you can think of a set of cookies sent from one site to one client as an array. On the client, they are stored as separate pieces of information, but in PHP, you can address them as elements of a single array. Listing 6-8, later in this chapter, will provide an example of this.

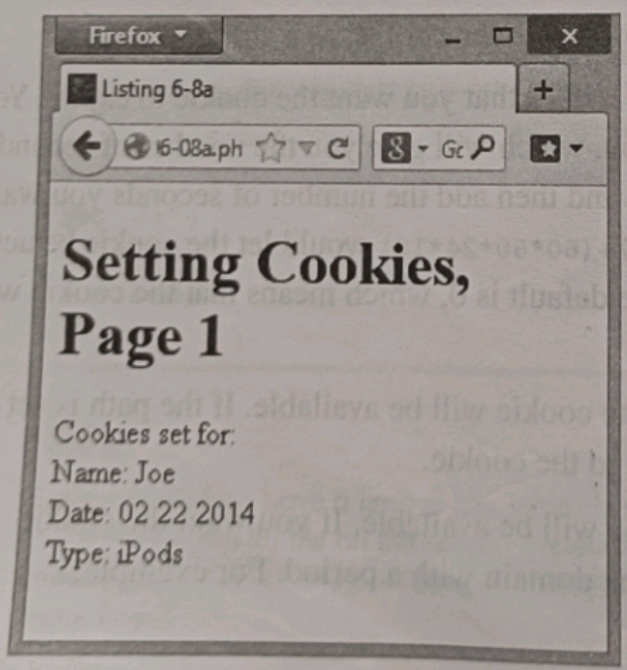
CAUTION

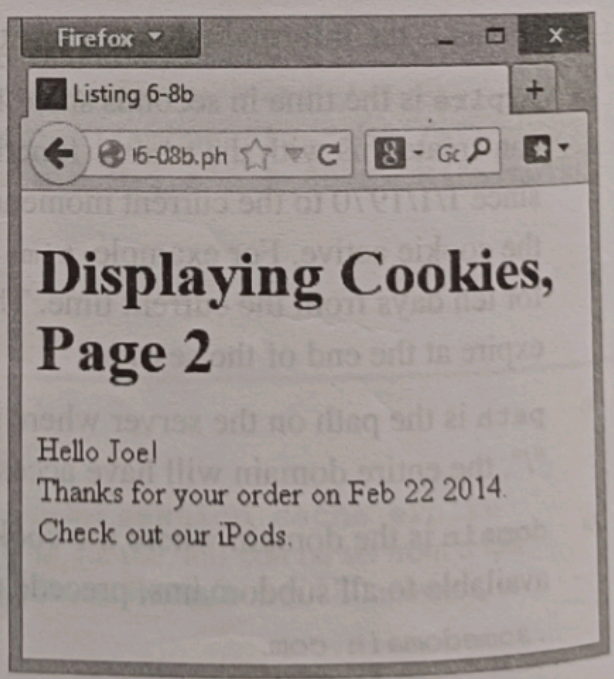
Cookies are viewable by spyware applications, so do not use cookies to store sensitive information like usernames, passwords, and account numbers.

Using $_COOKIE

After a cookie has been set, the next time the domain that set it reconnects with the client, the client will automatically return the cookie information to the server, and it will be stored in the `$_COOKIE` superglobal array. You must close the connection with the client in which you set the cookie and reopen the connection before `$_COOKIE` will contain the information.

Listing 6-8a demonstrates a website named "MatTech" storing a customer's first name, the date of her or his last order, and the type of merchandise purchased using cookies that don't expire for 90 days and that are available throughout the domain that set the cookies. Listing 6-8b demonstrates receiving and displaying the information, like this:





Listing 6-8a Setting a Cookie

```
<?php
   //Set cookies for name, date, and type of purchase
   $name = "Joe" ;
   $date = time() ;
   $type = "iPods" ;
   $expire = time()+(60*60*24*90);
   setcookie("MatTech[name]", $name, $expire, "/");
   setcookie("MatTech[date]", $date, $expire, "/");
   setcookie("MatTech[type]", $type, $expire, "/");
?>
<!DOCTYPE html>
<html>
   <head>
      <title>Listing 6-8a</title>
   </head>
   <body>
      <h1>Setting Cookies, Page 1</h1>
      <?php
         echo "Cookies set for:", "<br />";
         echo "Name: ", $name, "<br />";
         echo "Date: ", date('m d Y', $date), "<br />";
         echo "Type: ", $type, "<br />";
      ?>
   </body>
</html>
```

Listing 6-8b Displaying a Cookie

```
<html>
   <head>
      <title>Listing 6-8b</title>
   </head>
   <body>
      <h1>Displaying Cookies, Page 2</h1>
      <?php
         if (isset ($_COOKIE["MatTech"])) {  //Check if there
            echo "Hello ", $_COOKIE["MatTech"][name], "! <br />";
            echo "Thanks for your order on ", date('M d Y',
               $_COOKIE["MatTech"][date]), ". <br />";
            echo "Check out our ", $_COOKIE["MatTech"][type], ". <br />";
         }
      ?>
   </body>
</html>
```

TIP

To delete, or unset, a cookie, set a new one with the same name, a value of "", and a date prior to the current date, such as `time()-3600`. If you set an array of cookies, you must delete them all. For the cookies set in Listings 6-8a and b, Listing 6-8c shows what is needed to delete them:

Listing 6-8c Deleting a Cookie

```
<?php
   //Delete cookies for name, date, and type of purchase
   $expire = time()-(60*60);
   setcookie("MatTech[name]", "", $expire, "/");
   setcookie("MatTech[date]", "", $expire, "/");
   setcookie("MatTech[type]", "", $expire, "/");
?>
```

Server Variables

Server variables provide information about the server, the software that is running on it, the current script that produced the request, and the client and its software. The server in communication with the client produces this information and stores it in the superglobal array `$_SERVER`. Each web server produces a slightly different set of elements in the array, which are defined by their associative keys. Listing 6-9 is a short script for displaying the `$_SERVER` array elements on your server. Figure 6-3 shows the server variables generated with the Apache server in Zend Server on my computer.

Listing 6-9 Server Variables Generator

```
<html>
   <head>
      <title>Listing 6-9</title>
   </head>
   <body>
      <h1>Exploring Server Variables</h1>
      <?php
       foreach ($_SERVER as $key => $value) {
          echo "<b> $key   :   </b> $value ", "<br />";
       }
      ?>
   </body>
</html>
```