Apress™
Books for Professionals by Professionals

Chapter Seven: “File I/O and the File System”

A Programmer’s Introduction to PHP 4.0

by William Jason Gilmore
ISBN # 1-893115-85-2

CHAPTER 7

File I/O and the File System

This chapter introduces a particularly important aspect of PHP: file I/O (input/output). As you can imagine, data input and output flows are put to considerable use in the developing of Web applications. Not limited to simple reading and writing of files, PHP provides support for viewing and modifying server information, in addition to executing third-party programs. These features are the subject of this chapter.

Verifying a File's Existence and Size

It is useful to be able to determine the existence of a file before attempting to work with it. Two functions are particularly useful for accomplishing this: `file_exists()` and `is_file()`.

file_exists()

The `file_exists()` function will ensure that file exists, returning true if it does, and false otherwise. Its syntax is:

```
bool file_exists (string file)
```

Here's how you can verify the existence of a file:

```
$filename = "userdata.txt";

if (! file_exists ($filename)) :
     print "File $filename does not exist!";
endif;
```

is_file()

The `is_file()` function will return true if file exists and is a readable/writable file. Essentially, `is_file()` is a bullet-proof version of `file_exists()`, verifying not only the file's existence but also whether it can be read from or written to:

```
bool is_file (string file)
```

This example shows how to verify the existence and validity of a file:

```
$file = "somefile.txt";
if (is_file($file)) :
     print "The file $file is valid and exists!";
else :
     print "Either $file does not exist or it is not a valid file!";
endif;
```

Once you have verified that the file of interest does exist and is capable of having various I/O operations performed on it, you can open it.

filesize()

The `filesize()` function will return the size, in bytes, of the file designated by filename, or false should an error occur. Its syntax is:

```
int filesize (string filename)
```

Assume that you want to know the size of a file named pastry.txt. You can use `filesize()` to retrieve this information:

```
$fs = filesize("pastry.txt");
print "Pastry.txt is $fs bytes.";
```

This will return:

```
Pastry.txt is 179 bytes.
```

Before files can be manipulated, they must be opened and assigned a file handle. Once you have finished working with a file, it should be closed. These subjects are the focus of the next section.

Opening and Closing I/O

Before you can perform any I/O operation on a file, you must open it using the `fopen()` function.

fopen()

The `fopen()` function opens a file, assuming that it exists, and returns an integer, better known as a file handle. Its syntax is:

```
int fopen (string file, string mode [, int use_include_path])
```

File may be a file contained in the local file system, an stdio stream, or a remote file obtained via HTTP or FTP.

The input file can be of several forms, denoted by the filename syntax. These forms are listed here:

- If file is perceived to be a local filename, the file will be opened, and a pointer to the file will be returned.
- If file is either php://stdin, php://stdout, or php://stderr, stdio will be opened accordingly.
- If file begins with http://, an HTTP connection will be opened to the file server in question and a file pointer will be returned to the file in question.
- If file begins with ftp://, an FTP connection to the server in question will be opened, and a file pointer will be returned to the specified file. Two particularities are worth noting regarding this option: If the server does not support passive mode FTP, `fopen()` will fail. Furthermore, FTP files can only be opened exclusively either for reading or writing.

NOTE *When an FTP server is in passive mode, it is listening for a connection from a client. In contrast, when an FTP server is in active mode, the server makes the connection to the client. Active mode is generally the default.*

The mode specifies the read/write readiness of the file in question. Table 7-1 lists several possible modes pertaining to how a file can be opened.

Table 7-1. File modes

MODE	MEANING
r	Read only. The file pointer is placed at the beginning of the file.
r+	Reading and writing. The file pointer is placed at the beginning of the file.
w	Write only. The file pointer is placed at the beginning of the file, and the file contents are erased. If the file does not exist, an attempt will be made to create it.
w+	Reading and writing. The file pointer is placed at the beginning of the file, and the file contents are erased. If the file does not exist, an attempt will be made to create it.
a	Write only. The file pointer is placed at the end of the file. If the file does not exist, an attempt will be made to create it.
a+	Reading and writing. The file pointer is placed at the end of the file. If the file does not exist, an attempt will be made to create it.

The third input parameter, `use_include_path`, can be set to 1 if you would like the file path to be compared to the include path contained in the php.ini file (described in Chapter 1). The following listing illustrates the opening of a file with `fopen()`. It is a good idea to use the command `die()` in conjunction with `fopen()`to ensure display of an appropriate message should the function fail:

```
$file = "userdata.txt";                                         // some file
$fh = fopen($file, "a+") or die("File ($file) does not exist!");
```

The next listing will open a connection with the PHP site (http://www.php.net):

```
$site = "http://www.php.net";     // some server that can communicate via HTTP
$sh = fopen($site, "r");          // assigns PHP.net index page to a filehandle.
```

Once you have finished with a file, you should always close it. This is accomplished with the `fclose()` function.

fclose()

The `fclose()` function closes the file designated by filepointer, returning true on success and false otherwise:

```
int fclose (int filepointer)
```

The `fclose()` function will only successfully close those files opened by `fopen()` or `fsockopen()`. Here's how you can close a file:

```
$file = "userdata.txt";
if (file_exists($file)) :
     $fh = fopen($file, "r");
     // execute various file-related functions
     fclose($fh);
else:
     print "File $file does not exist!";
endif;
```

Writing to a File

Once a file has been opened, there are generally two operations that can be performed; writing and reading.

is_writeable()

The `is_writeable()` function will ensure that file exists and is writable. It is capable of checking the writability of both a file and a directory. Its syntax is:

```
bool is_writeable (string file)
```

It is important to note that PHP will likely be running under the user ID that the Web server is using (typically "nobody"). An example of `is_writeable()` is included in the next section, "`fwrite().`"

fwrite()

The `fwrite()` function will simply write the contents of string to the file specified by filepointer. Its syntax is:

```
int fwrite (int filepointer, string string [, int length])
```

If the optional input parameter length is provided, writing will stop either after length characters have been written or after the end of string has been reached. The following example shows how to check the writability of a file:

```
<?
// user site traffic Information
$data = "08:13:00|12:37:12|208.247.106.187|Win98";
$filename = "somefile.txt";
// If file exists and Is writable
if ( is_writeable($filename) ) :
     // open file and place file pointer at end of file
     $fh = fopen($filename, "a+");
     // write $data to file
     $success = fwrite($fh, $data);
     // close the file
     fclose($fh);
else :
     print "Could not open $filename for writing";
endif;
?>
```

NOTE `Fputs()` *is an alias to* `fwrite()` *and can be used by substituting the function name* `fwrite` *with* `fputs`.

fputs()

The `fputs()` function is an alias to `fwrite()` and is implemented in exactly the same way. Its syntax is:

```
int fputs (int filepointer, string string [, int length])
```

As you have seen, I prefer `fputs()` to `fwrite()`. Keep in mind that this is just a stylistic preference and has nothing to do with any differences between the two functions.

Reading from a File

The ability to read from a file is of obvious importance. The following are a set of functions geared toward making file reading an efficient process. You will see that the syntax of many of the functions are almost replicas of those used for writing.

is_readable()

The `is_readable()` function will ensure that file exists and is readable. It is capable of checking the readability of both a file and a directory. Its syntax is:

```
bool is_readable (string filename)
```

It is important to note that PHP will likely be running under the user ID that the Web server is using (probably "nobody"), and therefore the file will have to be world readable for `is_readable()` to return a true value. Here's how you would ensure that a file exists and is readable:

```
if ( is_readable($filename) ) :
     // open file and place file pointer at end of file
     $fh = fopen($filename, "r");
else :
     print "$filename Is not readable!";
endif;
```

fread()

The `fread()` function reads up to length bytes from the file designated by filepointer, returning the file's contents. Its syntax is:

```
string fread (int filepointer, int length)
```

The file pointer must point to an opened file that is readable (see function `is_readable()`). Reading will stop either when length bytes have been read or when the end of the file has been reached. Consider the sample textfile pastry.txt, shown in Listing 7-1. It could be read in and displayed to the browser using this code:

```
$fh = fopen('pastry.txt', "r") or die("Can't open file!");
$file = fread($fh, filesize($fh));
print $file;
fclose($fh);
```

By using `filesize()` to retrieve the byte size of pastry.txt, you ensure that `fread()` will read in the entire contents of the file.

Listing 7-1: A sample file, pastry.txt

```
Recipe: Pastry Dough
1 1/4 cups all-purpose flour
3/4 stick (6 tablespoons) unsalted butter, chopped
2 tablespoons vegetable shortening
1/4 teaspoon salt
3 tablespoons water
```

fgetc()

The `fgetc()` function returns a string containing one character from the file pointed to by filepointer or returns false on reaching the end of file. Its syntax is:

```
string fgetc (int filepointer)
```

The file pointer must point to an opened file that is readable. (To ensure that a file is readable, see "`is_readable()`," earlier in this chapter.) Here is an example of outputting a file, character by character:

```
$fh = fopen("pastry.txt", "r");
while (! feof($fh)) :
     $char = fgetc($fh);
     print $char;
endwhile;

fclose($fh);
```

fgets()

The `fgets()` function returns a string read from a file pointed to by the file pointer. The file pointer must point to an opened file that is readable (see "`is_readable(),`" earlier in this chapter). Its syntax is:

```
string fgets (int filepointer, int length)
```

Reading will stop when one of the following conditions has been met:

- Length: 1 byte is read.
- A newline is read (returned with the string).
- An end of file (EOF) is read.

If you are interested in reading in a file line by line, you should just set the length parameter to a value higher than the number of bytes on a line. Here's an example of outputting a file, line by line:

```
$fh = fopen("pastry.txt", "r");
while (! feof($fh)) :
     $line = fgets($fh, 4096);
     print $line."<br>";
endwhile;
fclose($fh);
```

fgetss()

The `fgetss()` function operates exactly like `fgets()`, except that it will attempt to strip all HTML and PHP tags from the file designated by filepointer as its text is read:

```
string fgetss (int filepointer, int length [, string allowable_tags])
```

Before proceeding with an example, take a moment to read through Listing 7-2, as it is the file used in Listings 7-3 and 7-4.

Listing 7-2: The science.html sample program

```
<html>
<head>
<title>Breaking News - Science</title>
<body>
<h1>Alien lifeform discovered</h1><br>
<b>August 20, 2000</b><br>
Early this morning, a strange new form of fungus was found growing in the closet
of an old apartment refrigerator. It is not known if powerful radiation emanating
from the tenant's computer monitor aided in this evolution.
</body>
</html>
```

Listing 7-3: Stripping all tags from an HTML file before browser display

```
<?
$fh = fopen("science.html", "r");
while (!feof($fh)) :
   print fgetss($fh, 2048);
endwhile;
fclose($fh);
?>
```

As you can see from the resulting output, all HTML tags are stripped from science.html, eliminating all formatting:

```
Breaking News - Science Alien lifeform discovered August 20, 2000. Early this
morning, a strange new form of fungus was found growing in the closet of an old
apartment refrigerator. It is not known if powerful radiation emanating from the
tenant's computer monitor aided in this evolution.
```

Of course, you might be interested in stripping all but a select few tags from the file, for example line breaks (<br>). This is illustrated in Listing 7-4.

Listing 7-4: Stripping all but a select few tags from an HTML file

```
<?
$fh = fopen("science.html", "r");
$allowable = "<br>";
while (!feof($fh)) :
    print fgetss($fh, 2048, $allowable);
endwhile;
fclose($fh);
?>
```

```
Breaking News - Science Alien lifeform discovered August 20, 2000
Early this morning, a strange new form of fungus was found growing in the closet
of an old apartment refrigerator. It is not known if powerful radiation
emanating from the tenant's computer monitor aided in this evolution.
```

As you can see, `fgetss()` can be rather useful for file conversion, particularly when you have a large group of HTML files similarly formatted.

Reading a File into an Array

The `file()` function will read the entire contents of a file into an indexed array. Each element in the array corresponds to a line in the file. Its syntax is:

```
array file (string file [, int use_include_path])
```

If the optional input parameter use_include_path is set to 1, then the file is searched along the include path in the php.ini file (See Chapter 1 for more information about the php.ini file.) Listing 7-5 shows how to use `file()` to read pastry.txt, first shown in Listing 7-1.

Listing 7-5: Reading pastry.txt using file()

```
<?
$file_array = file( 'pastry.txt' );

while ( list( $line_num, $line ) = each( $file_array ) ) :
     print "<b>Line $line_num:</b> " . htmlspecialchars( $line ) . "<br>\n";
endwhile;
?>
```

Cycling through the array, each line is output along with the corresponding line number:

```
Line 0: Recipe: Pastry Dough
Line 1: 1 1/4 cups all-purpose flour
Line 2: 3/4 stick (6 tablespoons) unsalted butter, chopped
Line 3: 2 tablespoons vegetable shortening
Line 4: 1/4 teaspoon salt
Line 5: 3 tablespoons water
```

Redirecting a File Directly to Output

The `readfile()` function reads in a file and outputs it to standard output (in most cases the browser). Its syntax is:

```
int readfile (string file [, int use_include_path])
```

The number of bytes read in is returned to the caller. File may be a file contained in the local file system, an stdio stream, or a remote file obtained via HTTP or FTP. Its specifications for the file input parameter mimic those of the `fopen()` function.

Suppose you had a restaurant review that you wanted to display online. This review, entitled “latorre.txt”, follows:

Restaurant “La Torre,” located in Nettuno, Italy, offers an eclectic blend of style, history, and fine seafood cuisine. Within the walls of the medieval borgo surrounding the city, one can dine while watching the passersby shop in the village boutiques. Comfort coupled with only the freshest seafare make La Torre one of Italy’s finest restaurants.

Executing the following code will result in the entire contents of “latorre.txt” being displayed to standard output:

```
<?
$restaurant_file = "latorre.txt";
// display entire file to standard output
readfile($restaurant_file);
?>
```

Opening a Process File Pointer with popen()

Just as a file can be opened, so can a file pointer to a server process. This is accomplished with the function `popen()`. Its syntax is:

```
int popen (string command, string mode)
```

The input parameter command refers to the system command that will be executed, and *mode* refers to how you would use the `popen()` function to search a file:

```
<?
// open file "spices.txt" for writing purposes
$fh = fopen("spices.txt","w");
// Add a few lines of text
fputs($fh, "Parsley, sage, rosemary\n");
fputs($fh,"Paprika, salt, pepper\n");
fputs($fh,"Basil, sage, ginger\n");
// close the file handle
fclose($fh);
// Open UNIX grep process, searching for "Basil" in spices.txt
$fh =popen("grep Basil < spices.txt", "r");
// output the result of the grep
fpassthru($fh);
?>
```

The resulting output:

```
Basil, sage, ginger
```

The `fpassthru()` function is covered later this chapter in "External Program Execution."

pclose()

After you're done with a file or process, you should close it. The `pclose()` function simply closes the connection to a process designated by filepointer, just as `fclose()` closes a file opened by `fopen()`. Its syntax is:

```
int pclose (int filepointer)
```

The input parameter *filepointer* refers to a previously opened file pointer.

Opening a Socket Connection

PHP does not limit you to working solely with files and processes. You can also manipulate socket connections. A *socket* is a software tool that allows you to make connections with various services offered by some machine.

fsockopen()

The `fsockopen()` function establishes a socket connection to an Internet server via either TCP or UDP. Its syntax is:

```
int fsockopen (string host, int port [, int errnumber [, string errstring [, int
timeout]]])
```

The optional input parameters errnumber and errstring return error information specific to the attempt to connect to the host. Both of these parameters must be specified as reference variables. The other optional input parameter, timeout, can be used to specify the number of seconds the call should wait before the host to respond. Listing 7-6 shows how you might use `fsockopen()` to retrieve information about a server. However, before Listing 7-6, I need to introduce another function, `set_socket_blocking()`.

NOTE *UDP, short for User Datagram Protocol, is a connectionless protocol similar to TCP/IP.*

set_socket_blocking()

The `set_socket_blocking()` function, when the mode is set to false, allows you to obtain control of the timeout setting specified by the server pointed to by file-pointer:

```
set_socket_blocking(int filepointer, boolean mode)
```

The input parameter *filepointer* refers to a previously opened socket pointer, and *mode* refers to the mode that the socket file pointer will be switched to; false for nonblocking mode, true for blocking mode. An example of `fsockopen()` and `set_socket_blocking()` in shown in Listing 7-6.

Listing 7-6: Using fsockopen() to retrieve information about a server

```
<?
function get_the_host($host,$path) {
     // open the host
     $fp = fsockopen($host, 80, &$errno, &$errstr, 30);
     // take control of server timeout
     socket_set_blocking($fp, 1);
     // send the appropriate headers
     fputs($fp,"GET $path HTTP/1.1\r\n");
     fputs($fp,"Host: $host\r\n\r\n");
     $x = 1;
     // grab a bunch of headers
     while($x < 10) :
          $headers = fgets($fp, 4096);
         print $headers;
          $x++;
     endwhile;
     // close the filepointer.
     fclose($fp);
}

get_the_host("www.apress.com", "/");
?>
```

Execution of Listing 7-6 results in the following output:

```
HTTP/1.1 200 OK Server: Microsoft-IIS/4.0 Content-Location:
http://www.apress.com/Default.htm Date: Sat, 19 Aug 2000 23:03:25 GMT Content-
Type: text/html Accept-Ranges: bytes Last-Modified: Wed, 19 Jul 2000 20:25:06
GMT ETag: "f0a6166dbff1bf1:34a5" Content-Length: 1311
```

pfsockopen()

The `pfsockopen()` function is just a persistent version of `fsockopen()` This means that it will not automatically close the connection once the script making use of the command has terminated. Its syntax is:

```
int pfsockopen (string hostname, int port [, int errno [, string errstr [, int
timeout]]])
```

Depending on the exact purpose of your application, it may be more convenient to choose `pfsockopen()` over `fsockopen()`.

External Program Execution

It is also possible to execute programs residing on a server. These functions can be particularly useful when administrating various aspects of the system via a Web browser, in addition to creating more user-friendly system summaries.

exec()

The `exec()` function will execute the program specified by command and return the last line of the command output. Its syntax is:

```
string exec (string command [, string array [, int return_variable]])
```

Note that it will not display the command output, just execute it. It is possible to store all of the command output in the optional input parameter array. Furthermore, if the optional input parameter return_variable is provided in conjunction with array, it will be assigned the status of the executed command.

Listing 7-7 shows how exec can be used to execute the UNIX system function ping.

Listing 7-7: Using exec() to ping a server

```
<?
exec("ping -c 5  www.php.net", $ping);
// For Windows, do exec("ping -n 5 www.php.net", $ping);
for ($i=0; $i < count($ping); $i++) :
     print "<br>$ping[$i]";
endfor;
?>
```

```
PING www.php.net (208.247.106.187): 56 data bytes
64 bytes from 208.247.106.187: icmp_seq=0 ttl=243 time=66.602 ms
64 bytes from 208.247.106.187: icmp_seq=1 ttl=243 time=55.723 ms
64 bytes from 208.247.106.187: icmp_seq=2 ttl=243 time=70.779 ms
64 bytes from 208.247.106.187: icmp_seq=3 ttl=243 time=55.339 ms
64 bytes from 208.247.106.187: icmp_seq=4 ttl=243 time=69.865 ms

—- www.php.net ping statistics —-
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max/stddev = 55.339/63.662/70.779/6.783 ms
```

Backticks

An alternative method exists for execution of a system command, in which no predefined function is required. The command can be executed if it is enclosed within backticks (``), and its output subsequently displayed to the browser. An example follows:

```
$output = `ls`;
print "<pre>$output</pre>";
```

This would result in the directory contents from which the script executing these commands resides being output to the browser.

NOTE *The -c 5 (-n 5 for Windows) is a parameter internal to ping that tells it to ping the server x times. The -c means count, and the -n means number.*

If you are interested in simply returning unformatted command output, check out `passthru()`, described next.

passthru()

The `passthru()` function works almost exactly like `exec()`, except that the command output is automatically output. Its syntax is:

```
void passthru (string command [, int return_variable])
```

If the optional input parameter *return_variable* is provided, it will be assigned the command return status.

You can use `passthru()` to view the uptime of the server, for example:

```
passthru("uptime");
1:21PM up 4 days, 23:16, 1 user, load averages: 0.02, 0.01, 0.00
```

fpassthru()

The `fpassthru()` function behaves exactly like `passthru()`, except that it works with file pointers pointing to files or processes opened by `popen()`, `fopen()`, or `fsockopen()`. Its syntax is:

```
int fpassthru (int fp)
```

It will read the entire file or process pointed to by the file pointer and forward it directly to standard output.

system()

You can think of the `system()` function as a hybrid of `exec()` and `passthru()`, executing command and automatically outputting the results and returning the last line of command, the input parameter command being a call to some system command that the server recognizes. Its syntax is:

```
string system (string command [, int return_variable])
```

If the optional input parameter return_variable is provided, it will be assigned the command return status.

The escapeshellcmd() Security Feature

The `escapeshellcmd()` function will escape any potentially dangerous characters that may be supplied by a user (via an HTML form, for example) for reason of executing the `exec()`, `passthru()`, `system()` or `popen()` commands. Its syntax is:

```
string escapeshellcmd (string command)
```

User input should always be treated with some degree of caution, and even more so when users can input commands that may be executed with functions capable of executing system commands. Consider the following:

```
$user_input = 'rm -rf *'; // Input means erase the parent directory and _all_ of
                             Its children.
exec( $user_input);       // execute $user_input!!!
```

Left uncontrolled, such commands could cause disaster. However, if you use `escapeshellcmd()` to escape user input:

```
$user_input = `rm -rf *`;                // Input means erase the parent
                                            directory and _all_ of Its children.
exec( escapeshellcmd($user_input));      // escapes dangerous characters.
```

Keep in mind that the function `escapeshellcmd()` will escape the *, preventing the command from being executed as intended.

NOTE *Because security is such an important issue in the Web environment, I have devoted an entire chapter to it and how it relates to PHP programming. See Chapter 16, "Security," for more information.*

Working with the File System

PHP provides a number of functions geared toward viewing and manipulating server files. Obtaining numerous facts about server files, such as location, owner, and privileges, can be useful.

basename()

The `basename()` function returns the file pointed to by path. Its syntax is:

```
string basename (string path)
```

Here's how you would parse out the base name of a path:

```
$path = "/usr/local/phppower/htdocs/index.php";
$file = basename($path);     // $file = "index.php"
```

This effectively parses the path, returning just the filename.

getlastmod()

The `getlastmod()` function returns the most recent modification date and time of the page in which the function is placed. The syntax:

```
int getlastmod (void)
```

The return value is in the form of a UNIX timestamp and can be formatted using the `date()` function. Here's how you could display the last modified time of a page:

```
echo "Last modified: ".date( "H:i:s a", getlastmod() );
```

stat()

The `stat()` function returns a comprehensive indexed array of information concerning the file designated by filename:

```
array stat (string filename)
```

The indexed values correspond to the following pieces of information:

0	Device
1	Inode
2	Inode protection mode
3	Number of links
4	Owner user ID
5	Owner group ID
6	Inode device type
7	Byte size
8	Last access time
9	Last modification time
10	Last change time
11	File system I/O block size
12	Block allocation

Therefore, if you wanted the last access time of the filename in question, you would call element 8 of the returned array. Consider this example:

```
$file = "datafile.txt";
list ($dev, $inode, $inodep, $nlink, $uid, $gid, $inodev, $size, $atime, $mtime,
$ctime, $bsize) = stat($file);

print "$file is $size bytes. <br>";
print "Last access time: $atime <br>";
print "Last modification time: $mtime <br>";
```

```
popen.php is 289 bytes.
Last access time: August 15 2000 12:00:00
Last modification time: August 15 2000 10:07:18
```

In this example, I used `list()` to explicitly name each piece of returned information. Of course, you could also just return an array and then use an iterative loop to display each piece of information as necessary. As you can see, `stat()` can be particularly useful when you need to retrieve various information about a file.

Displaying and Modifying File Characteristics

All files on UNIX-based systems have three basic characteristics:

- Group membership
- Ownership
- Permissions

Each of these characteristics can be changed through its respective PHP functions. The functions described in this section will not work on Windows-based systems.

NOTE *If you are new to the UNIX operating system, a great resource for learning about the UNIX file system characteristics is at http://sunsite.auc.dk/linux-newbie/FAQ2.htm. Section 3.2.6 in particular addresses group, ownership, and permission issues.*

chgrp()

The `chgrp()` function will attempt to change the group of the file denoted by filename to group. Its syntax is:

```
int chgrp (string filename, mixed group)
```

filegroup()

The `filegroup()` function returns the group ID of the owner of a file specified by filename, or false should some error occur. Its syntax is:

```
int filegroup (string filename)
```

chmod()

The `chmod()` function changes the mode of filename to permissions. Its syntax is:

```
int chmod (string filename, int permissions)
```

The permissions must be specified in octal mode. The following example shows that `chmod()` is particular about the permissions input parameter:

```
chmod ("data_file.txt", g+r); // This will not work
chmod ("data_file.txt", 766); // This will not work
chmod ("data_file.txt", 0766); // This will work
```

fileperms()

The `fileperms()` function returns the permissions of a file specified by filename, or false should some error occur. Its syntax is:

```
int fileperms (string filename)
```

chown()

The `chown()` function attempts to change the ownership of a filename to user. Only the superuser can change the ownership of a file. Its syntax is:

```
int chown (string filename, mixed user)
```

fileowner()

The `fileowner()` function returns the user ID of the owner of the file specified by filename. Its syntax is:

```
int fileowner (string filename)
```

Copying and Renaming Files

Other useful system functions that can be performed via a PHP script are copying and renaming files on the server. The two functions capable of doing so are `copy()` and `rename()`.

copy()

You can easily make a copy of a file much in the same way as you would with the UNIX cp command. This is done with PHP's `copy()` function. Its syntax is:

```
int copy (string source, string destination)
```

The `copy()` function will attempt to copy a file by the name of source to a file named destination, returning true on success and false otherwise. If destination does not exist, `copy()` will create it. Here's how to back up a file with copy():

```
$data_file = "data1.txt";
copy($data_file, $data_file'.bak') or die("Could not copy $data_file");
```

rename()

A file can be renamed with the `rename()` function, returning true on success and false otherwise. Its syntax is:

```
bool rename (string oldname, string newname)
```

Here's how you would use the `rename()` function for renaming a file:

```
$data_file = "data1.txt";
rename($data_file, $data_file'.old') or die("Could not rename $data_file");
```

Deleting Files

unlink()

You can delete a file with the `unlink()` function. Its syntax is:

```
int unlink (string file)
```

If you are using PHP on a Windows system, you may have problems with this function. If so, you can use the previously discussed `system()` function, deleting a file with a call to the DOS del function:

```
system ("del filename.txt");
```

Working with Directories

You can modify and traverse directories just as you are able to modify and traverse files. A typical non-Windows directory structure might look similar to the one displayed in Listing 7-8.

Listing 7-8: A typical directory structure

```
drwxr-xr-x  4 root  wheel    512  Aug 13  13:51   book/
drwxr-xr-x  4 root  wheel    512  Aug 13  13:51   code/
-rw-r—r—    1 root  wheel    115  Aug  4   09:53  index.html
drwxr-xr-x  7 root  wheel   1024  Jun 29  13:03   manual/
-rw-r—r—    1 root  wheel    19   Aug 12  12:15  test.php
```

dirname()

The `dirname()` function operates as the counterpart to `basename()`, returning the directory element of path. Its syntax is:

```
string dirname (string path)
```

Here's an example of using `basename()` to parse the base name of a path:

```
$path = "/usr/local/phppower/htdocs/index.php";
$file = basename($path);     // $file = "/usr/local/phppower/htdocs"
```

You can also use `dirname()` in conjunction with the predefined variable *$SCRIPT_FILENAME* to obtain the complete path of the script executing the command:

```
$dir = dirname($SCRIPT_FILENAME);
```

is_dir()

The `is_dir()` function verifies that the file designated by filename is a directory:

```
bool is_dir (string filename)
```

Refer to Listing 7-8 to understand the following example:

```
$isdir = is_dir("index.html"); // returns false

$isdir = is_dir("book"); // returns true
```

mkdir()

The `mkdir()` function has the same purpose as the UNIX command mkdir(), creating a new directory. Its syntax is:

```
int mkdir (string pathname, int mode)
```

The pathname specifies the path in which the directory is to be created. Don't forget to include the directory name at the end of this path! The mode is the file permission setting to which the newly created directory should be set.

opendir()

Just as `fopen()` opens a file pointer to a given file, `opendir()` will open a directory stream specified by directory_path. Its syntax is:

```
int opendir (string directory_path)
```

closedir()

The `closedir()` function will close the directory stream pointed to by directory_handle. Its syntax is:

```
void closedir (int directory_handle)
```

readdir()

The `readdir()` function returns each element in a given directory. Its syntax is:

```
string readdir (int directory_handle)
```

Using it, we can easily list all files and child directories in a given directory:

```
$dh = openddir('.');
while ($file = readdir($dh)) :
     print "$file <br>";
endwhile;
closedir($dh);
```

chdir()

The `chdir()` function operates just like the UNIX cd function, changing to the file directory specified by directory. Its syntax is:

```
int chdir (string directory)
```

Assume that you were currently sitting at the directory. You could change to and subsequently output the contents of the book/directory as follows:

```
$newdir = "book";
chdir($newdir) or die("Could not change to directory ($newdir)");
$dh = opendir('.');
print "Files:";
while ($file = readdir($dh)) :
     print "$file <br>";
endwhile;
closedir($dh);
```

rewinddir()

The `rewinddir()` function will reset the directory pointer pointed to by directory_handle. Its syntax is:

```
void rewinddir (int directory_handle)
```

Project 1: A Simple Access Counter

This simple access counter will keep count of the number of visits to the page in which the script is inserted. Before checking out the code in Listing 7-9, take a moment to review the pseudocode:

1. Assign `$access` the name of the file in which you would like to store the counter.
2. Use `file()` to read the contents of `$access` into the `$visits` array. The @ preceding the function acts to suppress any potential errors (such as a nonexistent file).
3. Assign the first (and only) element of the `$visits` array to `$current_visitors`.
4. Increment `$current_visitors` by 1.
5. Open the `$access` file for writing, placing the file pointer at the beginning of the file.
6. Write `$current_visitors` to the `$access` file.
7. Close the file handle pointing to the `$access` file.

Listing 7-9: A simple access counter

```
<?
// script: simple access counter
// purpose: uses a file to keep track of visitor count.
$access = "hits.txt";               // name this file whatever you want
$visits = @file($access);           // feed file into array
$current_visitors = $visits[0];     // extract first (and only) element from array
++$current_visitors;                // increment visitor count
$fh = fopen($access, "w");          // open "hits.txt" and place file pointer at
                                       beginning of file
@fwrite($fp, $current_visitors);    // write new visitor count to "hits.txt"
fclose($fh);                        // close filepointer to "hits.txt"
?>
```

Project 2: A Site Map Generator

The script in Listing 7-10 produces a site map of all folders and files on a server, starting from a specified directory. The site map is staggered through the calculation of indentation values through several functions defined in this and previous chapters. Before checking out the code, take a moment to review the pseudocode:

1. Declare a few necessary variables: parent directory, folder graphic location, page title, and server OS flag (Windows or non-Windows).

2. Declaration of `display_directory()` function, which parses and formats a directory for display in the browser.

3. Concatenate the directory passed in as `$dir1` to `$dir`, producing the correct directory path.

4. Open the directory and read its contents. Format the directory name and files and display them to the browser.

5. If the file in question is a directory, recursively call `display_directory()` with the file passed in as the new directory to parse. Also calculate specific indentation value for formatting purposes.

If the file in question is a file, format it as a link to itself, in addition to calculating a specific indentation value for formatting purposes.

Listing 7-10: The sitemap.php sample program

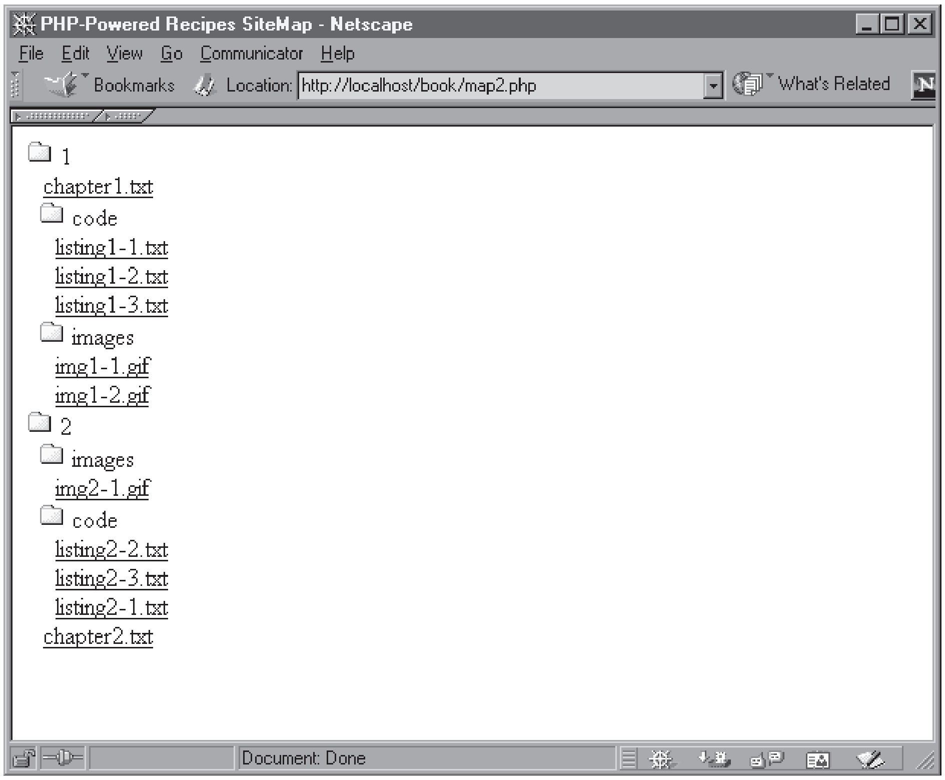
```
<?
// file: sitemap.php
// purpose: display a map of entire site structure

// From which parent directory should the sitemap begin?
$beg_path = "C:\Program Files\Apache Group\Apache\htdocs\phprecipes";

// What Is the location of the folder graphic?
// This path should be *relative* to the Apache server root directory
$folder_location = "C:\My Documents\PHP for Programmers\FINAL
CHPS\graphics\folder.gif";

// What should be displayed in the sitemap title bar?
$page_name = "PHPRecipes SiteMap";
```

```
// Will this script be used on a Windows or non-Windows server?
// (0 for Windows; 1 for non-Windows)
$using_linux = 0;
// function: display_directory
// purpose: Parses a directory specified by $dir1 and formats directory and file
structure.
// This function is recursively called.

function display_directory($dir1, $folder_location, $using_linux, $init_depth) {

// update the directory path
$dir .= $dir1;
$dh = opendir($dir);

while ($file = readdir($dh)) :
     // do not display the "." and ".."in each directory.
     if ( ($file != ".") && ($file != "..") ) :

          if ( $using_linux == 0 ) :
               $depth = explode("\\", $dir);
          else :
               $depth = explode("/", $dir);
          endif;
          $current_depth = sizeof($depth);

          // Build path In accordance with what OS Is being used.
          if ($using_linux == 0) :
               $tab_depth = $current_depth - $init_depth;
               $file = $dir."\\".$file;
          else :
               $file = $dir."/".$file;
          endif;

          // Is $file a directory?
          if ( is_dir($file) ) :
               $x = 0;
               // calculate tab depth
               while ( $x < ($tab_depth * 2) ) :
                    print "&nbsp;";
                    $x++;
              endwhile;

               print "<img src=\"$folder_location\" alt=\"[dir]\">
".basename($file)."<br>";
```

```
               // Increment the &nbsp; count

               // Recursive call to display_directory() function
              display_directory($file, $folder_location, $using_linux,
              // $init_depth);

          // Not dealing with a directory
          else :
               // Build path In accordance with what OS Is being used.
               if ($using_linux == 0) :
                    $tab_depth = ($current_depth - $init_depth) - 2;

                    $x = 0;
                    // calculate tab depth
                    while ( $x < (($tab_depth * 2) + 5) ) :
                         print "&nbsp;";
                         $x++;
                    endwhile;
                    print "<a href =
\"".$dir."\\".basename($file)."\">".basename($file)."</a> <br>";
               else :
                    print "<a href =
\"".$dir."/".basename($file)."\">".basename($file)."</a> <br>";
               endif;

          endif; // Is_dir(file)
     endif; // If ! "." or ".."

endwhile;

// close the directory
closedir($dh);
}
?>
<html>
<head>
<title> <? print "$page_name"; ?> </title>
</head>
<body bgcolor="#ffffff" text="#000000" link="#000000" vlink="#000000"
alink="#000000">
<?
// calculate Initial tab depth
if ($using_linux == 0) :
         $depth = explode("\\", $beg_path);
```

```
else :
    $depth = explode("/", $beg_path);
endif;
$init_depth = sizeof($depth);
display_directory($beg_path, $folder_location, $using_linux, $init_depth);
?>
</body>
</html>
```

Executing this script on the directory pointing to the folder I am using to organize a few of the chapters of this book displays the output shown in Figure 7-1.


PHP-Powered Recipes SiteMap - Netscape
File Edit View Go Communicator Help
Bookmarks
Location: http://localhost/book/map2.php
What's Related
Document: Done

```
1
 chapter1.txt
 code
  listing1-1.txt
  listing1-2.txt
  listing1-3.txt
 images
  img1-1.gif
  img1-2.gif
2
 images
  img2-1.gif
 code
  listing2-2.txt
  listing2-3.txt
  listing2-1.txt
 chapter2.txt
```

Figure 7-1. Using sitemap.php to display the structure of a server directory

Gilmore_07 12/4/00 1:06 PM Page 173

What's Next?

This chapter introduced many aspects of PHP's file-handling functionality, in particular:

- Verifying a File's Existence
- Opening I/O and closing I/O
- Writing to and reading from a file
- Redirecting a file directly to output
- External program execution
- Working with the file system

These topics set the stage for the next chapter, "Strings and Regular Expressions," as string manipulation and I/O manipulation go hand in hand when you are developing PHP-enabled Web applications. With that said, let's forge ahead!