Recognizing Newline Characters

The newline character, represented by the \n character sequence (\r\n on Windows), denotes the end of a line within a file. Keep this in mind when you need to input or output information one line at a time. Several functions introduced throughout the remainder of this chapter offer functionality tailored to working with the newline character. Some of these functions include file(), fgetcsv(), and fgets().

Recognizing the End-of-File Character

Programs require a standardized means for discerning when the end of a file has been reached. Thisstandard is commonly referred to as the *end-of-file*, or *EOF*, character. This is such an important conceptthat almost every mainstream programming language offers a built-in function for verifying whether theparser has arrived at the EOF. In the case of PHP, this function is feof(). The feof() function determineswhether a resource's EOF has been reached. It is used quite commonly in file I/O operations. Itsprototype follows:int feof(string *resource*)

An example follows:

```
<?php// Open a text file for reading purposes
$fh = fopen('/home/www/data/users.txt', 'r');
// While the end-of-file hasn't been reached, retrieve the next line
while (!feof($fh)) echo fgets($fh);
// Close the file
fclose($fh);?>
```

6.3 Working with directories in PHP

Organizing related data into entities commonly referred to as *files* and *directories* has long been a coreconcept in the modern computing environment.

Parsing Directory Paths

It's often useful to parse directory paths for various attributes such as the tailing extension name,directory component, and base name. Several functions are available for performing such tasks, all ofwhich are introduced in this section.

Retrieving a Path's Filename

The basename() function returns the filename component of a path. Its prototype follows:

string basename(string *path* [, string *suffix*])

If the optional suffix parameter is supplied, that suffix will be omitted if the returned file namecontains that extension. An example follows:

```
<?php
$path = '/home/www/data/users.txt';
printf("Filename: %s <br />", basename($path));
printf("Filename sans extension: %s <br />", basename($path, ".txt"));?>
```

Executing this example produces the following output:

```
Filename: users.txt
Filename sans extension: users
```

Retrieving a Path's Directory

The dirname() function is essentially the counterpart to basename(), providing the directory componentof a path. Its prototype follows:

string dirname(string *path*)

The following code will retrieve the path leading up to the file name users.txt:

```
<?php
$path = '/home/www/data/users.txt';
printf("Directory path: %s", dirname($path));?>
```

This returns the following:Directory path: /home/www/data

Learning More about a Path

The pathinfo() function creates an associative array containing three components of a path, namely the directory name, the base name, and the extension. Its prototype follows:

array pathinfo(string *path* [, *options*])

Consider the following path:

/home/www/htdocs/book/chapter10/index.html

The pathinfo() function can be used to parse this path into the following four components:

• Directory name: /home/www/htdocs/book/chapter10
• Base name: index.html
• File extension: html
• File name: index

You can use pathinfo() like this to retrieve this information:

```
<?php
$pathinfo = pathinfo('/home/www/htdocs/book/chapter10/index.html');
printf("Dir name: %s <br />", $pathinfo['dirname']);
printf("Base name: %s <br />", $pathinfo['basename']);
printf("Extension: %s <br />", $pathinfo['extension']);
printf("Filename: %s <br />", $pathinfo['filename']);?>
```

This produces the following output:

```
Dir name: /home/www/htdocs/book/chapter10
Base name: index.html
Extension: html
Filename: index
```

The optional $options parameter can be used to modify which of the four supported attributes arereturned. For instance, by setting it to PATHINFO_FILENAME, only the filename attribute will be populatedwithin the returned array. See the PHP documentation for a complete list of supported $options values.

Determining a File's Last Access Time

The fileatime() function returns a file's last access time in Unix timestamp format or FALSE on error. Its prototype follows:int fileatime(string *filename*)

Determining a File's Last Changed Time

The filectime() function returns a file's last changed time in Unix timestamp format or FALSE on error. Its prototype follows:int filectime(string *filename*)

Determining a File's Last Modified Time

The filemtime() function returns a file's last modification time in Unix timestamp format or FALSE otherwise. Its prototype follows:int filemtime(string *filename*)

```
<?php
$file = '/var/www/htdocs/book/chapter10/stat.php';
printf("File last accessed: %s", date("m-d-y g:i:sa", fileatime($file)));
printf("File inode last changed: %s", date("m-d-y g:i:sa", filectime($file)));
echo "File last updated: ".date("m-d-y g:i:sa", filemtime($file));?>
```

Determining a File's Size

The filesize() function returns the size, in bytes, of a specified file. Its prototype follows:int filesize(string *filename*)

Calculating a Disk's Free Space

The function disk_free_space() returns the available space, in bytes, allocated to the disk partitionhousing a specified directory. Its prototype follows:float disk_free_space(string *directory*)

Calculating Total Disk Size
The disk_total_space() function returns the total size, in bytes, consumed by the disk partition housinga specified directory. Its prototype follows:float disk_total_space(string *directory*)
If you use this function in conjunction with disk_free_space(), it's easy to offer useful spaceallocation statistics:

```
<?php
$file = '/www/htdocs/book/chapter1.pdf';
$bytes = filesize($file);
$kilobytes = round($bytes/1024, 2);
printf("File %s is $bytes bytes, or %.2f kilobytes", basename($file), $kilobytes);
$partition = '/usr';
// Determine total partition space
$totalSpace = disk_total_space($partition) / 1048576;
// Determine used partition space
$usedSpace = $totalSpace - disk_free_space($partition) / 1048576;
printf("Partition: %s (Allocated: %.2f MB. Used: %.2f MB.)",
$partition, $totalSpace, $usedSpace);?>
```

The process required for reading a directory's contents is quite similar to that involved in reading a file.

Opening a Directory Handle
Just as fopen() opens a file pointer to a given file, opendir() opens a directory stream specified by a path.
Its prototype follows:resource opendir(string *path* [, resource *context*])

Closing a Directory Handle
The closedir() function closes the directory stream. Its prototype follows:void closedir(resource *directory_handle*)

Parsing Directory Contents
The readdir() function returns each element in the directory. Its prototype follows:
string readdir([resource *directory_handle*])
Among other things, you can use this function to list all files and child directories in a givendirectory:

```
<?php
$dh = opendir('/usr/local/apache2/htdocs/');
while ($file = readdir($dh))
echo "$file <br />";
closedir($dh);?>
```

Sample output follows:

```
.
..
articles
images
news
test.php
```

Note that readdir() also returns the . and .. entries common to a typical Unix directory listing. Youcan easily filter these out with an if statement:if($file != "." AND $file != "..")
If the optional *directory_handle* parameter isn't assigned, then PHP will attempt to read from thelast link opened by opendir().

Reading a Directory into an Array
The scandir() function, introduced in PHP 5, returns an array consisting of files and directories found indirectory or returns FALSE on error. Its prototype follows:array scandir(string *directory* [,int *sorting_order* [, resource *context*]])

Setting the optional sorting_order parameter to 1 sorts the contents in descending order, overridingthe default of ascending order. Executing this example (from the previous section)

```
<?php
print_r(scandir('/usr/local/apache2/htdocs'));?>
```

returns the following
Array ([0] => . [1] => .. [2] => articles [3] => images[4] => news [5] => test.php)

6.1 Creating and deleting a file
6.5 Opening a file for writing, reading, or appending

Opening a File
The fopen() function binds a file to a handle. Once bound, the script can interact with this file via thehandle. Its prototype follows:
resource fopen(string *resource*, string *mode* [, int *use_include_path*[, *resource context*]])

While fopen() is most commonly used to open files for reading and manipulation, it's also capableof opening resources via a number of protocols, including HTTP, HTTPS, and FTP.
The *mode*, assigned at the time a resource is opened, determines the level of access available to thatresource. The various modes are defined in below table

Mode	Description
R	Read-only. The file pointer is placed at the beginning of the file.
r+	Read and write. The file pointer is placed at the beginning of the file.
W	Write only. Before writing, delete the file contents and return the file pointer to the beginningof the file. If the file does not exist, attempt to create it.
w+	Read and write. Before reading or writing, delete the file contents and return the file pointerto the beginning of the file. If the file does not exist, attempt to create it.
A	Write only. The file pointer is placed at the end of the file. If the file does not exist, attempt tocreate it. This mode is better known as Append.
a+	Read and write. The file pointer is placed at the end of the file. If the file does not exist,attempt to create it. This process is known as *appending to the file*.
x	Create and open the file for writing only. If the file exists, fopen() will fail and an error of levelE_WARNING will be generated.
x+	Create and open the file for writing and writing. If the file exists, fopen() will fail and an errorof level E_WARNING will be generated.

The first opens a read-only handle to a text file residing on the localserver:
$fh = fopen('/var/www/users.txt', 'r');

The next example demonstrates opening a write handle to an HTML document:
$fh = fopen('/var/www/docs/summary.html', 'w');

The next example refers to the same HTML document, except this time PHP will search for the file inthe paths specified by the include_path directive (presuming the summary.html document resides in thelocation specified in the previous example, include_path will need to include the path/usr/local/apache/data/docs/):
$fh = fopen('summary.html', 'w', 1);

The final example opens a read-only stream to a remote index.html file:
$fh = fopen('http://www.example.com/', 'r');
Of course, keep in mind fopen() only readies the resource for an impending operation. Other thanestablishing the handle, it does nothing; you'll need to use other functions to actually perform the readand write operations.
Closing a File

Good programming practice dictates that you should destroy pointers to any resources once you'refinished with them. The fclose() function handles this for you, closing the previously opened filepointer specified by a file handle, returning TRUE on success and FALSE otherwise. Its prototypefollows:
boolean fclose(resource *filehandle*)
The filehandle must be an existing file pointer opened using fopen() or fsockopen().

Deleting files: unlink() function is used to delete files. The term "unlink" is used because one can think of these filenames as links that join the files to the directory you are currently viewing. The function returns true on success or false on failure.
Syntax: unlink (string $filename); example : unlink ($samplefile);

```
<?php
$fn = './backup/test.bak';
if(unlink($fn)){
echo sprintf("The file %s deleted successfully",$fn);
}else{
echo sprintf("An error occurred deleting the file %s",$fn);}
```

6.6 Writing Data to the file

Writing a String to a File

The fwrite() function outputs the contents of a string variable to the specified resource. Its prototypefollows:
int fwrite(resource *handle*, string *string* [, int *length*])
If the optional length parameter is provided, fwrite() will stop writing when length characters havebeen written. Otherwise, writing will stop when the end of the string is found. Consider this example:

```
<?php
// Data we'd like to write to the subscribers.txt file
$subscriberInfo = 'Jason Gilmore|jason@example.com';
// Open subscribers.txt for writing
$fh = fopen('/home/www/data/subscribers.txt', 'a');
// Write the data
fwrite($fh, $subscriberInfo);
// Close the handle
fclose($fh);?>
```

6.2 Reading and writing text files
6.7 Reading character

Reading from a File

PHP offers numerous methods for reading data from a file, ranging from reading in just one character ata time to reading in the entire file with a single operation. Many of the most useful functions areintroduced in this section.

Reading a File into an Array

The file() function is capable of reading a file into an array, separating each element by the newlinecharacter, with the newline still attached to the end of each element. Its prototype follows:
array file(string *filename* [int *use_include_path* [, resource *context*]])
Although simplistic, the importance of this function can't be overstated, and therefore it warrants asimple demonstration. Consider the following sample text file named users.txt:

```
Ale ale@example.com
Nicole nicole@example.com
Laura laura@example.com
```

The following script reads in users.txt and parses and converts the data into a convenient Webbasedformat.
Notice file() provides special behavior because unlike other read/write functions, youdon't have to establish a file handle in order to read it:

```
<?php
// Read the file into an array
$users = file('users.txt');
// Cycle through the array
foreach ($users as $user) {
// Parse the line, retrieving the name and e-mail address
list($name, $email) = explode(' ', $user);
// Remove newline from $email
$email = trim($email);
// Output the formatted name and e-mail address
echo "<a href=\"mailto:$email\">$name</a><br /> ";          }          ?>
```

This script produces the following HTML output:

```
<a href="ale@example.com">Ale</a><br />
<a href="nicole@example.com">Nicole</a><br />
<a href="laura@example.com">Laura</a><br />
```

Like fopen(), you can tell file() to search through the paths specified in the include_pathconfiguration parameter by setting *use_include_path* to 1.

Reading File Contents into a String Variable

The file_get_contents() function reads the contents of a file into a string. Its prototype follows:

string file_get_contents(string *filename* [, int *use_include_path* [, resource *context* [, int*offset* [, int *maxlen*]]]])

By revising the script from the preceding section to use this function instead of file(), you get thefollowing code:

```
<?php
// Read the file into a string variable
$userfile= file_get_contents('users.txt');
// Place each line of $userfile into array
$users = explode("\n", $userfile);
// Cycle through the array
foreach ($users as $user) {
// Parse the line, retrieving the name and e-mail address
list($name, $email) = explode(' ', $user);
// Output the formatted name and e-mail address
printf("<a href='mailto:%s'>%s</a><br />", $email, $name);          }          ?>
```

The *use_include_path* and *context* parameters operate in a manner identical to those defined in thepreceding section. The optional *offset* parameter determines the location within the file where thefile_get_contents() function will begin reading. The optional *maxlen* parameter determines themaximum number of bytes read into the string.

Reading a Specific Number of Characters

The **fgets**() function returns a certain number of characters read in through the opened resourcehandle, or everything it has read up to the point when a newline or an EOF character is encountered. Itsprototype follows:

string fgets(resource *handle* [, int *length*])

If the optional *length* parameter is omitted, 1,024 characters is assumed. In most situations, thismeans that fgets() will encounter a newline character before reading 1,024 characters, therebyreturning the next line with each successive call. An example follows:

```
<?php
// Open a handle to users.txt
$fh = fopen('/home/www/data/users.txt', 'r');
// While the EOF isn't reached, read in another line and output it
while (!feof($fh)) echo fgets($fh);
```

```
// Close the handle
fclose($fh);?>
```

Reading a File One Character at a Time

The fgetc() function reads a single character from the open resource stream specified by handle. If theEOF is encountered, a value of FALSE is returned. Its prototype follows:

string fgetc(resource *handle*)

Ignoring Newline Characters

The fread() function reads length characters from the resource specified by handle. Reading stops whenthe EOF is reached or when length characters have been read. Its prototype follows:

string fread(resource *handle*, int *length*)

Note that unlike other read functions, newline characters are irrelevant when using fread(), makingit useful for reading binary files. Therefore, it's often convenient to read the entire file in at once usingfilesize() to determine the number of characters that should be read in:

```
<?php
$file = '/home/www/data/users.txt';
// Open the file for reading
$fh = fopen($file, 'r');
// Read in the entire file
$userdata = fread($fh, filesize($file));
// Close the file handle
fclose($fh);      ?>
```

Reading in an Entire File

The **readfile**() function reads an entire file specified by filename and immediately outputs it to the output buffer, returning the number of bytes read. Its prototype follows:

int readfile(string *filename* [, int *use_include_path*])

Enabling the optional use_include_path parameter tells PHP to search the paths specified by theinclude_path configuration parameter. This function is useful if you're interested in simply dumping anentire file to the browser:

```
<?php
$file = '/home/www/articles/gilmore.html';
// Output the article to the browser.
$bytes = readfile($file);?>
```

```
echo readfile("webdictionary.txt");
```

Like many of PHP's other file I/O functions, remote files can be opened via their URL if theconfiguration parameter fopen_wrappers is enabled.

6.4 Checking for existence of file

PHP | file_exists( ) Function

The file_exists() function in PHP is an inbuilt function which is used to check whether a file or directory exists or not. The path of the file or directory you want to check is passed as a parameter to the file_exists() function which returns True on success and False on failure.

file_exists($path)

Parameters: The file_exists() function in PHP accepts only one parameter *$path*. It specifies the path of the file or directory you want to check.

Return Value: It returns True on success and False on failure.

Errors And Exception:

The file_exists() function returns False if the path specified points to non-existent files.

For files larger than 2gb, some of the filesystem functions may give unexpected results since PHP’s integer type is signed and many platforms use 32bit integers.
Example: echo file_exists('/user01/work/gfg.txt');