

File Handling in PHP

Opening and Closing Files

Files are opened in PHP using the **fopen** command. The command takes two parameters, the file to be opened, and the mode in which to open the file. The function returns a file pointer if successful, otherwise zero (false). Files are opened with **fopen** for reading or writing.

```
$fp = fopen("myfile.txt", "r");
```

If **fopen** is unable to open the file, it returns 0. This can be used to exit the function with an appropriate message.

```
if ( !$fp = fopen("myfile.txt", "r") ) )  
    exit("Unable to open the input file.");
```

File Modes

The following table shows the different modes the file may be opened in.

Mode	Description
r	Read Only mode, with the file pointer at the start of the file.
r+	Read/Write mode, with the file pointer at the start of the file.
w	Write Only mode. Truncates the file (effectively overwriting it). If the file doesn't exist, fopen will attempt to create the file.
w+	Read/Write mode. Truncates the file (effectively overwriting it). If the file doesn't exist, fopen will attempt to create the file.
a	Append mode, with the file pointer at the end of the file. If the file doesn't exist, fopen will attempt to create the file.
a+	Read/Append, with the file pointer at the end of the file. If the file doesn't exist, fopen will attempt to create the file.

Note: The *mode* may also contain the letter 'b'. This is useful only on systems which differentiate between binary and text files (i.e. Windows. It's useless on Unix/Linux). If not needed, it will be ignored.

Closing a File

The **fclose** function is used to close a file when you are finished with it.

```
fclose($fp);
```

Reading from Files

You can read from files opened in r, r+, w+, and a+ mode. The **feof** function is used to determine if the end of file is true.

```
if ( feof($fp) )  
    echo "End of file<br>";
```

The **feof** function can be used in a while loop, to read from the file until the end of file is encountered. A line at a time can be read with the **fgets** function:

```
while( !feof($fp) ) {  
    // Reads one line at a time, up to 254 characters. Each line ends with a newline.  
    // If the length argument is omitted, PHP defaults to a length of 1024.  
    $myline = fgets($fp, 255);  
    echo $myline;  
}
```

You can read in a single character at a time from a file using the **fgetc** function:

```
while( !feof($fp) ) {  
    // Reads one character at a time from the file.  
    $ch = fgetc($fp);  
    echo $ch;  
}
```

You can also read a single word at a time from a file using the **fscanf** function. The function takes a variable number of parameters, but the first two parameters are mandatory. The first parameter is the file handle, the second parameter is a C-style format string. Any parameters passed after this are optional, but if used will contain the values of the format string.

```
$listFile = "list.txt"; // See next page for file contents  
if (!$fp = fopen($listFile, "r"))  
    exit("Unable to open $listFile.");  
while (!feof($fp)) {  
    // Assign variables to optional arguments  
    $buffer = fscanf($fp, "%s %s %d", $name, $title, $age);  
    if ($buffer == 3) // $buffer contains the number of items it was able to assign  
        print "$name $title $age<br>\n";  
}
```

Here is the file **list.txt**:

Dave Programmer 34

Sue Designer 21

Lisa Programmer 29

Nigel User 19

You can also store the variables in an array by omitting the optional parameters in **fscanf**:

```
while (!feof($fp)) {  
    $buffer = fscanf($fp, "%s %s %d"); // Assign variables to an array  
    // Use the list function to move the variables from the array into variables  
    list($name, $title, $age) = $buffer;  
    print "$name $title $age<br>\n";  
}
```

You can read in an entire file with the **fread** function. It reads a number of bytes from a file, up to the end of the file (whichever comes first). The **filesize** function returns the size of the file in bytes, and can be used with the **fread** function, as in the following example.

```
$listFile = "myfile.txt";  
if (!$fp = fopen($listFile, "r"))  
    exit("Unable to open the input file, $listFile.");  
$buffer = fread($fp, filesize($listFile));  
echo "$buffer<br>\n";  
fclose($fp);
```

You can also use the **file** function to read the entire contents of a file into an array instead of opening the file with **fopen**:

```
$array = file('filename.txt');
```

Each array element contains one line from the file, where each line is terminated by a newline.

Writing to Files

The **fwrite** function is used to write a string, or part of a string to a file. The function takes three parameters, the file handle, the string to write, and the number of bytes to write. If the number of bytes is omitted, the whole string is written to the file. If you want the lines to appear on separate lines in the file, use the `\n` character. **Note:** Windows requires a carriage return character as well as a new line character, so if you're using Windows, terminate the string with `\r\n`. The following example logs the visitor to a file, then displays all the entries in the file.

```
<?php  
$logFile = "stats.txt";  
$fp = fopen($logFile, "a+"); // Open the file in append/read mode  
$userDetails = $_SERVER['HTTP_USER_AGENT']; // Create a string containing user details  
if (isset($_SERVER['HTTP_REFERER']))  
    $userDetails .= " {$_SERVER['HTTP_REFERER']}<br>\r\n";  
else  
    $userDetails .= "<br>\r\n";
```

```

fwrite($fp, "$userDetails"); // Write the user details to the file
rewind($fp); // Move the file pointer to the start of the file
$entries = 0;
while(!feof($fp)) { // Display each line in the file
    $buffer = fgets($fp, 1024);
    if ($buffer != "") {
        echo $buffer;
        $entries++;
    }
} // Show a summary
echo "There are $entries entries in the log file<br>";
fclose ($fp);
?>

```

Locking Files

boolean **flock** (resource *fp*, integer operation)

PHP supports a portable way of locking complete files in an advisory way (which means all accessing programs have to use the same way of locking or it will not work). If there is a possibility that more than one process could write to a file at the same time then the file should be locked.

flock() operates on *fp* which must be an open file pointer. *operation* is one of the following:

- To acquire a shared lock (reader), set *operation* to **LOCK_SH**
- To acquire an exclusive lock (writer), set *operation* to **LOCK_EX**
- To release a lock (shared or exclusive), set *operation* to **LOCK_UN**
- If you don't want **flock()** to block while locking, add **LOCK_NB** to **LOCK_SH** or **LOCK_EX**

When obtaining a lock, the process may block. That is, if the file is already locked, it will wait until it gets the lock to continue execution. **flock()** allows you to perform a simple reader/writer model which can be used on virtually every platform (including most Unix derivatives and even Windows). **flock()** returns **TRUE** on success and **FALSE** on error (e.g. when a lock could not be acquired).

Here is a script that writes to a log file with the **fputs** function and then displays the log file's contents:

```

<?php
$fp = fopen("/tmp/log.txt", "a");
flock($fp, LOCK_EX); // get lock
fputs($fp, date("h:i A l F dS, Y\n")); // add a single line to the log file
flock($fp, LOCK_UN); // release lock
fclose($fp);
echo "<pre>"; // dump log
readfile("/tmp/log.txt");
echo "</pre>\n";
?>

```

Other Useful File Functions

The **opendir** function returns a directory handle; **closedir** closes a directory; **readdir** reads a directory entry.

```
$myDirectory = opendir("."); // use the current directory
while($entryname = readdir($myDirectory)) {
    echo "<tr>";
    echo "<td>$entryname</td>";
    echo "<td align='right'>";
    echo filesize($entryname); // the filesize function returns the file size in bytes
    echo "</td>";
    echo "</tr>\n";
}
closedir($myDirectory);
```

The **is_dir** function tests if a filename refers to a directory:

```
if(is_dir($filename))
    echo $filename . " is a directory";
```

The **is_file** function determines if a filename refers to a file:

```
if(is_file($filename))
    echo $filename . " is a file";
```

The **is_readable** function returns TRUE if the file exists and it is readable, otherwise it returns false. On Unix/Linux this is determined by the file's permissions. On Windows, TRUE is always returned if the file exists.

```
if(is_readable($filename))
    echo $filename . " is readable";
```

The **is_writable** function determines whether the server will allow you to write data to the file before you attempt to open it:

```
if(is_writable('../quotes.txt')) {
    // attempt to open the file and write to it
} else {
    echo 'The file is not writable';
}
```

Note: there are many more file functions. Please refer to PHP documentation for an exhaustive list.