O'REILLY
ONLamp.com
LAMP: THE OPEN SOURCE WEB PLATFORM

Published on ONLamp.com (http://www.onlamp.com/)
See this if you're having trouble printing code examples



Working with Files in PHP

by John Coggeshall
10/03/2002

Welcome back. In my last column, I finished my discussion of using references in PHP to do some pretty powerful things. This week, I'll be changing gears completely and begin my discussion of working with the file system through the use of an extensive set of PHP functions. I'll cover the fundamentals of reading and writing text files first, then cover some of the more advanced features and offer a few great shortcut functions. Let's get started.

Basic File IO

Opening and Closing Files

The most versatile file IO (input and output) function available to the PHP developer is the `fopen()` (or file open) command. This function is the most common starting point for any file operations that occur within most PHP scripts. The syntax for the `fopen()` command is as follows:

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

The `fopen()` function when called returns an integer called a `file reference` (or pointer, depending on personal preference) or zero if the operation failed. This integer value in most cases must be saved into a variable because a great number of the file IO functions available to PHP require it as a parameter when called. As you can see, the `fopen()` function requires two parameters and allows an optional third. The first, the `filename`, obviously represents the name of the file in the file system that will be accessed. This file may or may not actually exist, depending on whether you are creating a new file or working with an existing one. The second required parameter, the `mode`, is used to determine how the file is intended to be accessed. The possible modes of access available in PHP are listed in the table below:

String	Access Mode
`r`	Access the file for reading only, start access from beginning of the file.
`r+`	Access the file for both reading and writing, start access `f` from beginning of the file.
`w`	Access the file for writing only, If the file exists, erase all contents. If file does not exist, an attempt will be made to create the file. In either case access the file from the beginning.
`w+`	Access the file for both reading and writing, If the file exists erase all contents. If the file does not exist, an attempt will be made to create the file. In either case, access the file from the beginning.
	Open the file for writing only. If the file does not exist, an attempt will be made to create it. If

a	the file does exist, access will start from the end of the file (no erasing).
a+	Open the file for reading and writing. If the file does not exist, an attempt will be made to create it. If the file does exist, access will start from the end of the file (no erasing).

As you can see, these modes can be boiled down into three basic groups: reading, writing, and appending. Only one of these strings may be used as the `mode` parameter for the `fopen()` command.

The final optional parameter `use_include_path` is an integer representing a boolean value. If the value of this parameter is true (an integer value of 1,) then when an attempt is made to open a file that does not exist in the specified directory, an attempt will be made to file the file in the directories specified by PHP's include path before further action is taken.

Important note: In order for a file to be accessed or created, the directory and/or file in question must have the appropriate permissions. Also note that access must be granted to the user under which the PHP script will be executed (usually the same username as the web server).

Once a file reference has been created, and any reading/writing done to the file opened is complete, it is necessary that the file be closed before it can be opened again. Although all file references are automatically closed upon the termination of the PHP script which opened them, it is good programming practice to close the files manually. To do this, PHP provides the `fclose()` function. This function takes a single parameter (the `file reference`) and returns a boolean value indicating the success or failure.

For example, the below code will open and close a file in the */tmp/* directory called *test.txt* using the `a+` (Append read/write) mode:

```
<?php
   $fr = fopen('/tmp/test.txt', 'a+');
   if(!$fr) {
      echo "Error! Couldn't open the file.";
   } else {
      // $fr now can be used to represent the opened file
   }
   if(!fclose($fr)) {
      echo "Error! Couldn't close the file.";
   }
?>
```

Related Reading



PHP Cookbook
By David Sklar, Adam Trachtenberg

Read Online--Safari
Search this book on Safari:

Go
Only This Book
Code Fragments only

Reading and Writing Strings

Now that we have discussed opening a file, how do we read any data from it? Although there are many different methods, one of the most basic methods is the `fgets()` function. The syntax for the `fgets()` function is as follows:

```
fgets(int file reference [, int length])
```

As you can see, the only required parameter to the `fgets()` function is the `file reference` returned by the `fopen()` function. The optional parameter represents the number of bytes to read from the file (minus one). If this parameter is not provided, the `fgets()` will read one kilobyte (1024 bytes) from the file until a new line character is read or until the function reaches the end of the file. The return value from the `fgets()` function is the string read from the file. Please note that in the event that the `fgets()` function stops reading because a new line character is encountered, the new line character will be returned as part of the string. This is not true, however, if the end of file is reached. In that case, the end of file character is omitted from the resulting string.

Of course, for this function to work the `file reference` used must have been created by opening a file in the proper mode (any mode allowing reading).

The complement of the `fgets()` function, `fputs()` is the basic function available in PHP for writing a string to a file. Unlike `fgets()`, `fputs()` requires a minimum of two parameters (the `file reference` and the `string` to write) with a third optional parameter representing the amount of the passed string to actually write to the file.

Now that we are familiar with the ways to open, close, read, and write to a file in PHP, let's make a simple counter script. The purpose of this script is simply to count the number of times the script has been executed. When placed on a Web page, this script can be used to count the number of times the page has been viewed. To create this script we'll need to store an string representing the number of times the page has been accessed into a file, then every time the script is executed, we'll read that value, add one, and write it back to the file.

Let's get started.

First, we'll need to open the file. For our purposes, I'll use the file *counter.txt* in the */tmp/* directory for reading:

```
<?php
   $visitors = 0;       // Initialize the visitors to zero
   $fr = fopen('/tmp/counter.txt', 'r');
```

Because the first time the script is executed, the file shouldn't exist we have to first test the `file reference` to see if the `fopen()` command was successful. If not we need to create the file, set the total visitors to one, and write that value into the file:

```
   if(!$fr) {
      $visitors = 1;            // Our first visitor
      $fr = fopen('/tmp/counter.txt','w');
      if(!$fr) {
         echo "Could not create the counter file!";
         exit;
      }
      fputs($fr, $visitors);
      fclose($fr);
```

If we were successful at opening the *counter.txt* file, then our next step is to read the string containing the total visitors from the file, add one to it, and display it to the user:

```
   } else {
      $visitors = fgets($fr);
```

```
      $visitors++;
      echo "You are visitor number: $visitors";
```

Now, we need to close the current `file reference` (as it's in the wrong mode) and re-open the file in writing mode. Once that's done, we can write the new visitor number back to the file, close it, and we're all done:

```
      fclose($fr);
      $fr = fopen('/tmp/counter.txt','w');
      if(!$fr) {
         echo "Could not re-create the counter file!";
         exit;
      }
      fputs($fr, $visitors);
      fclose($fr);
   }
?>
```

That's All for Today

That concludes our introduction to accessing the file system with PHP. Now that you are familiar with some of the fundamental functions used when accessing files, in the coming weeks I'll be continuing our discussion including some shortcuts, dealing with permissions and working with the directory structure. See you then.

Editor's Note, 8 October 2002: due to an unfortunate typo, the previous versions of this article left out the initial argument to the `fputs()` calls. This has now been fixed.

John Coggeshall is a a PHP consultant and author who started losing sleep over PHP around five years ago.

---

*Read more **PHP Foundations** columns.*

Return to the PHP DevCenter.

Copyright © 2009 O'Reilly Media, Inc.