

Reading and Writing Files

PHP comes with a powerful and flexible file manipulation API, which enables developers to view and modify file attributes, read and list directory contents, alter file permissions, retrieve file contents into a variety of native data structures, and search for files based on specific patterns. The following sections discuss reading and writing files, and retrieving file information.

Reading Data from a File

To begin with, let's consider the process of opening a file and reading its contents.

Create and run the following PHP script (remember to alter the value of the `$file` variable to an actual file on your system that is readable by the web server):

```
<?php
// set file to read
$file = '/home/web/projects.txt';

// open file
$fh = fopen($file, 'r') or die('Could not open file!');
// read file contents
$data = fread($fh, filesize($file)) or die('Could not read file!');
// close file
fclose($fh);
// print file contents
echo $data;
?>
```

A review of the previous script will reveal the three basic steps to reading data from a file:

1. Open the file and assign it a file handle: PHP needs a file handle to read data from a file. This file handle can be created with the `fopen()` function, which accepts two arguments: the name and path to the file, and a string indicating the mode in which the file is to be opened ('r' for read).
2. Interact with the file via its handle and extract its contents into a PHP variable. If the `fopen()` function is successful, it returns a file handle— `$fh`—which can be used for further interaction with the file. This file handle is used by the `fread()` function, which reads the file and places its contents into a variable. The second

argument to `fread()` is the number of bytes to be read. You can usually obtain this information through the `filesize()` function, which returns the size of the file in bytes.

3. Close the file. Once you're done with the file, it's a good idea to close it with `fclose()`, to avoid using up memory. This last step is not strictly necessary, but it's a good habit to develop.

The following example demonstrates this:

```
<?php
// set file to read
$file = '/home/web/projects.txt';
// read file into array
$data = file($file) or die('Could not read file!');
// loop through array and print each line
foreach ($data as $line)
{
echo $line;
}
?>
```

Another way to do this is with the `file_get_contents()` function, new in PHP 4.3.0 and PHP 5.0, which reads the entire file into a string. Here's an

example:

```
<?php
// set file to read
$file = '/home/web/projects.txt';
// read file into string
$data = file_get_contents($file) or die('Could not read file!');
// print contents
echo $data;
?>
```

Writing Data to a File

The steps involved in writing data to a file are almost identical to those involved in

reading it: open the file and obtain a file handle, use the file handle to write data to it,

and close the file. There are two differences:

1. You must `fopen()` the file in write mode ('w' for write).
2. Instead of using the `fread()` function to read from the file handle, use the `fwrite()` function to write to it

```
<?php
// set file to write
$file = '/tmp/dummy.txt';
// open file
$fh = fopen($file, 'w') or die('Could not open file!');
// write to file
fwrite($fh, 'Hello, file!') or die('Could not write to file');
// close file
fclose($fh);
?>
```

An alternative here is the `file_put_contents()` function, new in PHP 5.0, which takes a string and writes it to a file in a single line of code. The next example illustrates this:

```
<?php
// set file to write
$file = '/tmp/dump.txt';
// write to file
file_put_contents($file, 'Hello, file!') or die('Could not write to file');
?>
```

Creating a Session and Registering Session Variables

In PHP, the `session_start()` function is used to create a client session and generate a session ID. Once a session has been created, it becomes possible to register any number of *session variables*; these are regular variables which can store textual or numeric information and can be manipulated by standard PHP functions, but are unique to each client. In a PHP script, session variables may be registered as key-value pairs in the special `$_SESSION` associative array.

To see how sessions and session variables work, examine the following script, which creates a new client session and registers two session variables:

```
<?php
// first page
// create a session
session_start();
// register some session variables
$_SESSION['username'] = 'deathsbane';
$_SESSION['role'] = 'admin';
?>
```

On subsequent pages, calls to the `session_start()` function re-create the prior session environment by restoring the values of the `$_SESSION` associative

Destroying a Session

To destroy an extant session—for example, on user logout—reset the `$_SESSION` array, and then use the `session_destroy()` function to erase session data.

```
<?php
// re-create session
session_start();
// reset session array
$_SESSION = array();
// destroy session
session_destroy();
?>
```

