Beginning PHP 5 and MySQL: From Novice to Professional

W. JASON GILMORE

Apress®

Beginning PHP 5 and MySQL: From Novice to Professional

Copyright © 2004 by W. Jason Gilmore

ISBN (pbk): 1-893115-51-8

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editor: Chris Mills

Technical Reviewer: Matt Wade

Editorial Board: Steve Anglin, Dan Appleman, Ewan Buckingham, Gary Cornell, Tony Davis, John Franklin, Jason Gilmore, Chris Mills, Steve Rycroft, Dominic Shakeshaft, Jim Sumser, Karen Watterson, Gavin Wray, John Zukowski

Project Managers: Nate McFadden, Laura Cheu

Copy Edit Manager: Nicole LeClerc

Copy Editor: Brian MacDonald

Production Manager: Kari Brooks

Production Editor: Kelly Winquist

Compositor: Diana Van Winkle, Van Winkle Design Group

Proofreader: Linda Seifert, Christy Wagner

Indexer: Kevin Broccoli

Artist: Kinetic Publishing Services, LLC

Cover Designer: Kurt Krames

Manufacturing Manager: Tom Debolski

Distributed to the book trade in the United States by Springer-Verlag New York, Inc., 175 Fifth Avenue, New York, NY 10010 and outside the United States by Springer-Verlag GmbH & Co. KG, Tiergartenstr. 17, 69112 Heidelberg, Germany.

In the United States: phone 1-800-SPRINGER, e-mail `orders@springer-ny.com`, or visit `http://www.springer-ny.com`. Outside the United States: fax +49 6221 345229, e-mail `orders@springer.de`, or visit `http://www.springer.de`.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail `info@apress.com`, or visit `http://www.apress.com`.

The source code for this book is available to readers at `http://www.apress.com` in the Downloads section.

CHAPTER 13

Handling File Uploads

DESPITE THE HYPE, glory, and often rumors of magical powers bestowed upon it almost from birth, the Web is really nothing more than a global file server. More specifically, it's a global file server farm. And although HTML files of course make up the vast majority of all files transferred via the HTTP protocol, it has become quite commonplace to make Word documents, PDFs, executables, MPEGs, and any number of other file types accessible via the Web. Although FTP has historically been the common means for uploading files to a server, such file transfers are becoming increasingly prevalent via a Web-based interface. In this chapter, you'll learn all about PHP's file-upload handling capabilities. In particular, chapter topics include:

- PHP's file upload configuration directives
- PHP's `$_FILES` superglobal array, used to handle file upload data
- PHP's built-in upload functions: `is_uploaded_file()` and `move_uploaded_file()`
- A review of possible values returned from an upload script

As always, numerous real-world examples are offered throughout this chapter, providing you with applicable insight into this topic.

Uploading Files via the HTTP Protocol

The way files are uploaded via a Web browser was officially formalized in around November 1995, when Ernesto Nebel and Larry Masinter of the Xerox Corporation proposed a standardized methodology for doing so within RFC 1867 (`http://www.ietf.org/rfc/rfc1867.txt`.). This memo, which formulated the groundwork for making the additions necessary to HTML to allow for file uploads (subsequently incorporated into HTML 3.0), also offered the specification for a new Internet Media Type, *multipart/form-data*. This new media type was desired, because the standard type used to encode "normal" form values, *application/x-www-form-urlencoded*, is considered too inefficient to handle large quantities of binary data such as that which might be uploaded via such a form interface. An example follows, and a screenshot of the corresponding output is shown in Figure 13-1:

```
<form action="uploadmanager.html" enctype="multipart/form-data" method="post">
     Name:<br /> <input type="text" name="name" value="" /><br />
     Email:<br /> <input type="text" name="email" value="" /><br />
     Homework:<br /> <input type="file" name="homework" value="" /><br />
     <p><input type="submit" name="submit" value="Submit Homework" /></p>
</form>
```

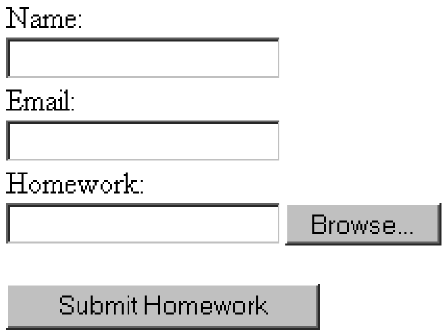

Figure 13-1. HTML form incorporating the "file" input type tag

Understand that this offers only half of the desired result; whereas the `file` input type and other upload-related attributes standardize the way files are sent to the server via an HTML page, no capabilities are offered for determining what happens once that file gets there! The reception and subsequent handling of the uploaded files is a function of an upload handler, created using some server process, or capable server-side language, like Perl, Java, or PHP. We'll devote the remainder of this chapter to this aspect of the upload process.

Handling Uploads with PHP

Successfully managing file uploads via PHP is the result of cooperation between various configuration directives, the `$_FILES` superglobal, and a properly coded Web form. In the following sections, I'll introduce all three topics and conclude with a number of examples.

PHP's File Upload/Resource Directives

Several configuration directives are available for fine-tuning PHP's file upload capabilities. These directives determine whether PHP's file upload support is enabled, the maximum allowable uploadable file size, the maximum allowable script memory allocation, and various other important resource benchmarks. I'll introduce these directives in this section.

file_uploads (boolean)
Scope: `PHP_INI_SYSTEM`, Default value: 1
The `file_uploads` directive determines whether PHP scripts on the server can accept file uploads.

max_execution_time (integer)
Scope: `PHP_INI_ALL`, Default value: 30
The `max_execution_time` directive determines the maximum amount of time, in seconds, that a PHP script will execute before registering a fatal error.

memory_limit (integer)M
Scope: `PHP_INI_ALL`, Default value: 8M
The `memory_limit` directive sets a maximum allowable amount of memory, in megabytes, that a script can allocate. Note that the integer value must be followed by an "M" in order for this setting to work properly. This prevents runaway scripts from monopolizing server memory, and even crashing the server in certain situations. This directive only takes effect if the `--enable-memory-limit` flag was set at compile-time.

upload_max_filesize (integer)M
Scope: `PHP_INI_SYSTEM`, Default value: 2M
The `upload_max_filesize` directive determines the maximum size, in megabytes, of an uploaded file. This directive should be smaller than `post_max_size` (introduced in a later section), because it applies only to information passed via the `file` input type, and not to all information passed via the `POST` instance. Like `memory_limit`, note that an `M` must follow the integer value.

upload_tmp_dir (string)
Scope: `PHP_INI_SYSTEM`, Default value: NULL
Because an uploaded file must be successfully transferred to the server before subsequent processing on that file can begin, a staging area of sorts must be designated for such files as the location where they can be temporarily placed until moved to their final location. This location is specified using the `upload_tmp_dir` directive. For example, suppose you wanted to temporarily store uploaded files in the `/tmp/phpuploads/` directory:

```
upload_tmp_dir = "/tmp/phpuploads/"
```

Keep in mind that this directory must be writable by the user owning the server process. Therefore, if user "nobody" owns the Apache process, then user "nobody" should either be made owner of the temporary upload directory, or should be made a member of the group owning that directory. If this is not done, user "nobody" will be unable to write the file to the directory, unless world write permissions are assigned to the directory.

post_max_size (integer)M
Scope: `PHP_INI_SYSTEM`, Default value: 8M
The `post_max_size` directive determines the maximum allowable size, in megabytes, of information that can be accepted via the `POST` method. As a rule of thumb, this directive setting should be larger than `upload_max_filesize`, to account for any other form fields that may be passed in addition to the uploaded file. Like `memory_limit` and `upload_max_filesize`, note that an `M` must follow the integer value.

The $_FILES Array

The `$_FILES` superglobal is special in that it is the only one of the predefined EGCPFS (Environment, Get, Cookie, Put, Files, Server) superglobal arrays that is two-dimensional. Its purpose is to store a variety of information pertinent to a file (or files) uploaded to the server via a PHP script. In total, five items are available in this array, each of which is introduced in this section.

> **NOTE** *Each of the items introduced in this section makes reference to* `userfile`*. This is simply a placeholder for the name assigned to the file upload form element. Therefore, this value will likely change in accordance to your chosen name assignment.*

$_FILES['userfile']['error']
The `$_FILES['userfile']['error']` array value offers important information pertinent to the outcome of the upload attempt. In total, five return values are possible, one signifying a successful outcome, and four others denoting specific errors which arise from the attempt. The names and meanings of each return value are introduced in the later section, “Upload Error Messages.”

$_FILES['userfile']['name']
The `$_FILES['userfile']['name']` variable specifies the original name of the file, including the extension, as declared on the client machine. Therefore, if you browse to a file named `vacation.jpg`, and upload it via the form, this variable will be assigned the value `vacation.jpg`.

$_FILES['userfile']['size']
The `$_FILES['userfile']['size']` variable specifies the size, in bytes, of the file uploaded from the client machine. Therefore, in the case of the `vacation.jpg` file, this variable could plausibly be assigned a value like `5253`, or roughly 5 kilobytes.

$_FILES['userfile']['tmp_name']
The `$_FILES['userfile']['tmp_name']` variable specifies the temporary name assigned to the file once it has been uploaded to the server. This is the name of the file assigned to it while stored in the temporary directory (specified by the PHP directive `upload_tmp_dir`).

$_FILES['userfile']['type']
The $_FILES['userfile']['type'] variable specifies the mime-type of the file uploaded from the client machine. Therefore, in the case of the vacation.jpg file, this variable would be assigned the value image/jpeg. If a PDF were uploaded, then the value application/pdf would be assigned.

Because this variable sometimes produces unexpected results, you should explicitly verify it yourself from within the script.

PHP's Upload Functions

In addition to the host of file-handling functions made available via PHP's file system library (see Chapter 10 for more information), PHP offers two functions specifically intended to aid in the upload process, is_uploaded_file() and move_uploaded_file(). Each function is introduced in this section.

is_uploaded_file()

```
boolean is_uploaded_file(string filename)
```

The is_uploaded_file() function determines whether a file specified by the input parameter *filename* was uploaded using the POST method. This function is intended to prevent a potential attacker from manipulating files not intended for interaction via the script in question. For example, consider a scenario in which uploaded files were made immediately available for viewing via a public site repository. Say an attacker wanted to make a file somewhat juicier than boring old class notes available for his perusal, say /etc/passwd. So rather than navigate to a class notes file as would be expected, the attacker instead types /etc/passwd directly into the form's file upload field.

Now consider the following uploadmanager.php script:

```
<?php
     copy($_FILES['classnotes']['tmp_name'],
              "/www/htdocs/classnotes/".basename($classnotes));
?>
```

The result in this poorly written example would be that the /etc/passwd file is copied to a publicly-accessible directory. (Go ahead, try it. Scary, isn't it?) To avoid such a problem, use the is_uploaded_file() function to ensure that the file denoted by the form field, in this case classnotes, is indeed a file that has been uploaded via the form. Revising the uploadmanager.php code:

```
<?php
if (is_uploaded_file($_FILES['classnotes']['tmp_name'])) {
     copy($_FILES['classnotes']['tmp_name'],
              "/www/htdocs/classnotes/".$_FILES['classnotes']['name']);
} else {
     echo "<p>Potential script abuse attempt detected.</p>";
}
?>
```

In the revised script, `is_uploaded_file()` checks whether the file denoted by `$_FILES['classnotes']['tmp_name']` has indeed been uploaded. If the answer is yes, the file is copied to the desired destination Otherwise, an appropriate error message is displayed.

move_uploaded_file()

```
boolean move_uploaded_file(string filename, string destination)
```

The `move_uploaded_file()` function was introduced in version 4.0.3 as a convenient means for moving an uploaded file from the temporary directory to a final location. Although `copy()` works equally well, `move_uploaded_file()` offers one additional feature that this function does not: It will check to ensure that the file denoted by the *`filename`* input parameter was in fact uploaded via PHP's HTTP POST upload mechanism. If the file has indeed not been uploaded, the move will fail and a `FALSE` value will be returned. Because of this, you can forego using `is_uploaded_file()` as a precursor condition to using `move_uploaded_file()`.

Using `move_uploaded_file()` is quite simple. Consider a scenario in which you want to move the uploaded class notes file to the directory `/www/htdocs/classnotes/`, while also preserving the filename as specified on the client:

```
move_uploaded_file($_FILES['classnotes']['tmp_name'],
                   "/www/htdocs/classnotes/".$_FILES['classnotes']['name']);
```

Of course you could rename the file to anything you wish when it's moved. It's important, however, that you properly reference the file's temporary name within the first (source) parameter.

Upload Error Messages

Like any other application component involving user interaction, you need a means to assess the outcome, successful or otherwise. How do you definitively know that the file-upload procedure was successful? And if something goes awry during the upload process, how do you know what caused the error? Thankfully, sufficient information for determining the outcome, and in the case of an error, the reason for the error, is provided in `$_FILES['userfile']['error']`.

UPLOAD_ERR_OK (Value = 0)
A value of 0 is returned if the upload is successful.

UPLOAD_ERR_INI_SIZE (Value = 1)
A value of 1 is returned if there is an attempt to upload a file whose size exceeds the specified by the `upload_max_filesize` directive.

UPLOAD_ERR_FORM_SIZE (Value = 2)
A value of 2 is returned if there is an attempt to upload a file whose size exceeds the value of the `MAX_FILE_SIZE` directive, which can be embedded into the HTML form.

> **NOTE** *Keep in mind that, because the* `MAX_FILE_SIZE` *directive is embedded within the HTML form, it can easily be modified by an enterprising attacker. Therefore, always use PHP's server-side settings (*`upload_max_filesize`, `post_max_filesize`*) to ensure that such predetermined absolutes are not surpassed.*

UPLOAD_ERR_PARTIAL (Value = 3)
A value of 3 is returned if a file was not completely uploaded. This might occur if a network error occurs that results in a disruption of the upload process.

UPLOAD_ERR_NO_FILE (Value = 4)
A value of 4 is returned if the user submits the form without specifying a file for upload.

File Upload Examples

Now that the groundwork has been set regarding the basic concepts, it's time to consider a few practical examples.

A Simple Upload Example

The first example actually implements the class notes example referred to throughout this chapter. To formalize the scenario, suppose that a professor invites students to post class notes to his Web site, the idea being that everyone might have something to gain from such a collaborative effort. Of course, credit should nonetheless be given where credit is due, so each file upload should be renamed to the last name of the student. In addition, only PDF files are accepted. Listing 13-1 offers an example:

Listing 13-1. A Simple File Upload Example

```
<form action="uploadmanager.php" enctype="multipart/form-data" method="post">
     Last Name:<br /> <input type="text" name="name" value="" /><br />
     Class Notes:<br /> <input type="file" name="classnotes" value="" /><br />
     <p><input type="submit" name="submit" value="Submit Notes" /></p>
</form>

<?php
/* Set a few constants */
define ("FILEREPOSITORY","/home/www/htdocs/class/classnotes/");

/* Make sure that the file was POSTed. */
if (is_uploaded_file($_FILES['classnotes']['tmp_name'])) {

     /* Was the file a PDF? */
     if ($_FILES['classnotes']['type'] != "application/pdf") {
          echo "<p>Class notes must be uploaded in PDF format.</p>";
     } else {
```

```
/* move uploaded file to final destination. */
          $name = $_POST['name'];

          $result = move_uploaded_file($_FILES['classnotes']['tmp_name'],
          FILEREPOSITORY."/$name.pdf");

          if ($result == 1) echo "<p>File successfully uploaded.</p>";
               else echo "<p>There was a problem uploading the file.</p>";

     } #endIF

} #endIF
?>
```

CAUTION *Remember that files are both uploaded and moved under the guise of the Web server daemon owner. Failing to assign adequate permissions to both the temporary upload directory and the final directory destination for this user will result in failure to properly execute the file upload procedure.*

Categorizing Uploaded Files by Date

The professor, delighted by the students' participation in the class notes project, has decided to move all class correspondence online. His current project involves providing an interface that will allow students to submit their daily homework via the Web. Like the class notes, the homework is to be submitted in PDF format, and will be assigned the student's last name as its file name when stored on the server. Because homework is due daily, the professor wants both a means for automatically organizing the assignment submissions by date, and also a means for ensuring that the class slackers can't sneak homework in after the deadline, which is 11:59:59 p.m. daily.

The script offered in Listing 13-2 automates all of this, minimizing administrative overhead for the professor. In addition to ensuring that the file is a PDF, and automatically assigning it the student's specified last name, the script will also create new folders daily, each following the naming convention MM-DD-YYYY.

Listing 13-2. Categorizing Uploaded Files by Date

```
<form action="homework.php" enctype="multipart/form-data" method="post">
     Last Name:<br /> <input type="text" name="name" value="" /><br />
     Homework:<br /> <input type="file" name="homework" value="" /><br />
     <p><input type="submit" name="submit" value="Submit Notes" /></p>
</form>

<?php
# Set a constant
define ("FILEREPOSITORY","/home/www/htdocs/class/homework/");
if (isset($_FILES['homework'])) {
     if (is_uploaded_file($_FILES['homework']['tmp_name'])) {
          if ($_FILES['homework']['type'] != "application/pdf") {
               echo "<p>Homework must be uploaded in PDF format.</p>";
          } else {

               /* Format date and create daily directory, if necessary. */
               $today = date("m-d-Y");
               if (! is_dir(FILEREPOSITORY.$today)) {
                    mkdir(FILEREPOSITORY.$today);
               }

               /* Assign name and move uploaded file to final destination. */
               $name = $_POST['name'];
               $result = move_uploaded_file($_FILES['homework']['tmp_name'],
                                        FILEREPOSITORY.$today."/"."$name.pdf");
               /* Provide user with feedback. */
               if ($result == 1) echo "<p>File successfully uploaded.</p>";
                    else echo "<p>There was a problem uploading the homework.</p>";

          }

     }
}
?>
```

Although this code could stand a bit of improvement, it accomplishes what the professor set out to do. Although it does not prevent students from submitting late homework, that homework will be placed in the folder corresponding with the current date as specified by the server clock.

NOTE *Fortunately for the students, PHP will overwrite previously submitted files, allowing them to repeatedly revise and resubmit homework as the deadline nears.*

Handling Multiple File Uploads

The professor, always eager to push his students to the outer limits of insanity, has decided to require the submission of two daily homework assignments. Always striving for a streamlined submission mechanism, the professor would like both assignments to be submitted via a single interface, and named `student-name1` and `student-name2`. The dating procedure used in the previous listing will be reused in this script. Therefore, the only real puzzle here is to figure out how to submit multiple files via a single form interface.

Earlier in this chapter I mentioned that the `$_FILES` array is unique because it is the only predefined variable array that is two-dimensional. This is not without reason; the first element of that array represents the file input name, therefore if multiple file inputs exist within a single form, each can be handled separately without interfering with the other. This concept is demonstrated in Listing 13-3.

Listing 13-3. Handling Multiple File Uploads

```
<form action="multiplehomework.php" enctype="multipart/form-data" method="post">
     Last Name:<br /> <input type="text" name="name" value="" /><br />
     Homework #1:<br /> <input type="file" name="homework1" value="" /><br />
     Homework #2:<br /> <input type="file" name="homework2" value="" /><br />
     <p><input type="submit" name="submit" value="Submit Notes" /></p>
</form>

<?php
/* Set a constant */
define ("FILEREPOSITORY","/home/www/htdocs/class/homework/");
if (isset($_FILES['homework'])) {
     if (is_uploaded_file($_FILES['homework1']['tmp_name']) &&
         is_uploaded_file($_FILES['homework2']['tmp_name'])) {

          if (($_FILES['homework1']['type'] != "application/pdf") ||
              ($_FILES['homework2']['type'] != "application/pdf")) {

                echo "<p>All homework must be uploaded in PDF format.</p>";

         } else {
               /* Format date and create daily directory, if necessary. */
              $today = date("m-d-Y");

              if (! is_dir(FILEREPOSITORY.$today))
                    mkdir(FILEREPOSITORY.$today);

              /* Name and move homework #1 */
              $filename1 = $_POST['name']."1";

              $result = move_uploaded_file($_FILES['homework1']['tmp_name'],
                             FILEREPOSITORY.$today."/"."$filename1.pdf");

              if ($result == 1) echo "<p>Homework #1 successfully uploaded.</p>";
              else echo "<p>There was a problem uploading homework #1.</p>";

              /* Name and move homework #2 */
              $filename2 = $_POST['name']."2";

              $result = move_uploaded_file($_FILES['homework2']['tmp_name'],
                             FILEREPOSITORY.$today."/"."$filename2.pdf");

              if ($result == 1) echo "<p>Homework #2 successfully uploaded.</p>";
              else echo "<p>There was a problem uploading homework #2.</p>";

          }
     }
}
?>
```

Although this script is a tad longer due to the extra logic required to handle the second homework assignment, it only differs slightly from Listing 13-2. However, there is one very important matter to keep in mind when working with this or any other script that handles multiple file uploads; the combined file size cannot exceed the `upload__max_size` or `post_max_size` configuration directives.

Summary

Transferring files via the Web eliminates a great many inconveniences otherwise posed by firewalls and FTP servers and clients. It also enhances an application's ability to easily manipulate and publish non-traditional files. In this chapter you learned just how easy it is to add such capabilities to your PHP applications. In addition to offering a comprehensive overview of PHP's file upload features, several practical examples were discussed.

In the next chapter, one of my favorite topics of Web development is introduced in excruciating detail: tracking users via session-handling.