PHP Upload Magick

Jacob Smits

Requirements

- PHP Version 5.2.6 or higher
- ImageMagick 6.0.6 or higher
- SWF Upload (included)

Installation Instructions

- The root folder of this GIT would be the root folder of your server. Simply copy the PHP folder to your servers root.

- The 'website' folder would be your website's domain folder. Ex: 'mywebsite.com'. Simply copy the contents of this folder to your websites domain folder.

- All paths in the contained files are relative, and should work without much, if any modification. If you want your directory tree to differ from my own, you must change the paths in the contained files.

Basic Upload Tutorial

The first step is to create a simple file upload form in HTML. **The file 'image_upload.html'** is included in the package, so open it now and we'll take a look at a few of the important elements.

'image_upload.html'

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Image Upload</title>
<link rel="stylesheet" href="http://www.thehoppr.com/css/upload.css" type="text/css" />
</head>

<body>

<form enctype="multipart/form-data" action="image_upload_exec.php" method="POST">
    <!-- MAX_FILE_SIZE must precede the file input field -->
    <input type="hidden" name="MAX_FILE_SIZE" value="1572864" />
    <!-- Name of input element determines name in $_FILES array -->
    Send this file: <input id="upload" name="upload" type="file" />
    <input type="submit" value="Send File" />
```

```
</form>

</body>

</html>
```

First thing is to make sure action is set to execute our file 'image_upload_exec.php' and that the method is post. Next we have a hidden input field that our server uses to determine if uploaded files are too large. Use a byte converter if you are unsure as to the size in bytes. Pay special attention to **name=”upload”.** This value is important as it is used in PHP to handle the file upload. I set 'id' to the same value just to avoid any possible bugs that may arise while using this with other scripts.

'image_upload_exec.php'

PHP Upload Magick is an object-based class. If you have never worked with OOP before, you're about to get a crash course. **Open up 'image_upload_exec.php'** now. Basically, our class of functions will work as long as we pass them some values, so we are going to set up our variables. We initiate the class *inside a variable* as a new class. We can then use this variable to point the class to certain pieces of information. Notice how the '→' arrow points to the variable. The class files must also be included at the beginning of our execution script in order to access it. Here I link to them by using my servers root. Your structure may vary depending on how you (or your provider) has set up your server.

```
<?php

include ("$DOCUMENT_ROOT/../php/classes/php_magick_upload/file_upload_class.php");
include ("$DOCUMENT_ROOT/../php/classes/php_magick_upload/image_manip_class.php");
$file = new image_upload;
```

Next, we set up our variables for the class.

```
//File Upload Vars

$file->theFile = $_FILES['upload']['name'];
$file->tempFile = $_FILES['upload']['tmp_name'];
$file->uploadName = 'upload';
$file->maxSize = 1572864;
$file->extensions = array(".jpg", ".png", ".gif", ".bmp");
$file->newName = "image";
$file->fileNum = 1;
$file->uploadDir = "$DOCUMENT_ROOT/../mywebsite.com/testing/results/";
$file->conFormat = ".jpg"; //Even know this is an image manipulation function, it needs to reside in the file upload vars so the renaming function works.
```

Detailed explanations of these variables can be found in the class files themselves, so I'll be brief here. $_FILES is the array PHP stores the information about your uploaded file. To make our class work, you need to set the variables 'theFile', 'tempFile', and 'uploadName' to what you named your input field in the HTML form (see bold in script above) In our case, we named that input field **'upload'** so we set these three variables accordingly. Leave the second values of the $_FILES array to '[name]' and '[tmp_name]'.

Next we have the max size variable. This should be set to the same value as in your HTML form. Extensions are the allowed extensions separated by ', ' (a comma and a space). Here we are accepting your standard image file types. While this script is mainly intended for image uploading, there's nothing wrong here with accepting just about any file type. 'newName' is set if you want to rename the file to a different name upon upload. This function also checks if the file name already exists in the upload directory, and if so, begins to attach numbers to the end of the new file name you've chosen. Finally, set the uploadDir to where you want to place the final file.

**Note that conFormat is the format you want to convert image files to. It resides in file_upload_class to avoid a bug with renaming files in which the renaming function will not be able to find the files in the folder that have already been uploaded.

So, with those important values set, you can now perform the upload function. This function has two flags, rename and validate. The validate function ensures the file is of proper extension. Set to true if you want to perform these functions, false if you dont.

```
//Perform functions

$file->upload(true, true);
```

Image Manipulation Tutorial

Now that we've covered how to upload files using PHP Upload Magick, let's move right into using the image manipulation function of the script and setting the variables needed to perform it. In our file **'image_upload_exec'** you will see a second set of variables underneath our file upload variables.

'image_upload_exec.php'

```
//Image Manipulation Vars

$file->thumbDir = "$DOCUMENT_ROOT/../mywebsite.com/testing/results/thumb/";
$file->constrain = true;
$file->sharpen = true;
$file->enlarge = true;
$file->max_X = 800;
$file->max_Y = 600;
$file->min_X = 300;
$file->min_Y = 300;
```

**Again, the best way to get familiar with this class library is to look through the class files themselves. Every variable and function is annotated with notes on it's usage there.

First we set the directory our thumbnails are to be saved in, if you so choose to create them. Then, we set constrain to true, which will force all our resizing functions to keep our aspect ratio in regards to matching max X and Y values. If constrain is on, it wont stretch or squash the image to fit the max sizes exactly. In most situations, you're probably going to want to keep this set to true. Sharpen is really used in conjunction with enlarging the image to minimum sizes. Enlarging images results in blurriness,

so a slight sharpen filter is applied. The min and max values should be pretty self-explanatory. Max values will not allow an image to exceed either value, and min values will force an image to be at least one of those values (again, it's a portrait/landscape thing).

All the vars to make the script function are now set. Now all we have to do is run our manipulation function, and the class will perform it to our file after it is saved on the server. If you do not want to run image manipulations, then do not run the function. This function has three flags: resize, createThumb, and convert. Convert will convert the uploaded image from one of the allowed types, to a specified type that you set in the File Upload Vars. That variable resides in that class to avoid bugs with renaming files.

```
//Perform functions

$file->upload(true, true);
$file->perform_manipulations(true, true, true);

//Error Output

$errormsg = $file->error_text();
echo $errormsg;
```

Lets quickly look at the last lines of code. If the script at any point catches an error, it outputs it to an array. All we are doing here is using a function to encode each individual error into a line of HTML, and then echoing that string. As of right now, the script also adds success statements to this same array. The final script will have more dynamic error reporting, including fatal errors that kill the script and delete the uploaded file.

Congratulations! You've completed this tutorial and thus have set up a basic image upload and manipulation. Next we will cover how to execute this script using the very end-user-friendly SWF Upload!

Multiple File Uploads Using SWF Upload

This script can very easily be combined with SWF Upload to download multiple files at once. SWF Upload uses Jquery and Flash to upload files one at a time. Included are all the neccissary files to get it working on your server. Note that I am not an expert on this particular piece of code. I only figured out the bare minimum in order to get the thing working, so please refer to it's documentation if you want to learn more about this powerful tool .

SWF Upload Documentation

We only need to change the name of our input field in our original execution script to get this working. Open up **'multiple_upload_exec'** to see the changes.

'multiple_upload_exec.php'

```
<?php

//Include upload classes
include ("$DOCUMENT_ROOT/../php/classes/php_magick_upload/file_upload_class.php");
```

```
include ("$DOCUMENT_ROOT/../php/classes/php_magick_upload/image_manip_class.php");
$file = new image_upload;

//File Upload Vars

$file->theFile = $_FILES['Filedata']['name'];
$file->tempFile = $_FILES['Filedata']['tmp_name'];
$file->uploadName = 'Filedata';
$file->maxSize = 1572864;
$file->renameFile = true;
$file->extensions = array(".jpg", ".png", ".gif", ".bmp");
$file->newName = "image";
$file->uploadDir = "$DOCUMENT_ROOT/../mywebsite.com/testing/results/";
$file->conFormat = ".jpg";

//Image Manipulation Vars

$file->constrain = true;
$file->thumbDir = "$DOCUMENT_ROOT/../mywebsite.com/testing/results/thumb/";
$file->sharpen = true;
$file->max_X = 800;
$file->max_Y = 600;
$file->min_X = 300;
$file->min_Y = 300;

//Perform functions
$file->upload(true, true);
$file->perform_manipulations(true, true, true);

//Error Output
$errormsg = $file->error_text();
header("HTTP/1.1 200 Uploaded File was Successful");
echo $errormsg;

?>
```

Notice we've changed our $_FILES fields to **'FilesData'.** This is the default name SWF Upload uses. We *are* going to use the rename function so you get an idea of how it works. The only other thing we add is a header that is returning a success code to ensure that our script messages are returned to the uploader.

Now, open up **'multiple_upload.php'.** The most important thing you should note about this web page file is that the links to the script files are correct for your server configuration. If you've kept my file structure, you should be fine. The next lines are all the settings for SWF Upload. Most of these are self-explanatory and I'm not going to go into any sort of depth in this tutorial, so please view their documentation to learn more about it. What you need to pay attention to is that, again, the links to the execution file and the button file are correct. The HTML that follows the scripts is formatted so SWF Upload can attach all it's elements accordingly. Do not change anything that looks important, especially the class elements.

That's it! Pretty simple huh? SWF Upload will only allow a certain amount of uploads and the users can only select files ending in the extensions you choose for it as well. It will then fire our upload script one at a time for each file.

Thanks for downloading my class and I hope you can put it to all sorts of good use!