File Version Management in PHP

This is an article on *File Version Management in PHP* in *PHP*.

Management of files

Manipulating files is a basic necessity for programmers and PHP gives you a great deal of tools for creating, uploading, and editing files. When you are manipulating files you must be very careful because you can do a lot of damage if you do some errors. Common errors include editing the wrong file, filling a hard-drive with garbage data, and accidentally deleting a file's contents. Before you can do anything with a file it has to exist! So we need to create file in PHP.

Creating files in PHP

The fopen function needs two important pieces of information to operate correctly. First, we must supply it with the name of the file that we want it to open. Secondly, we must tell the function what we plan on doing with that file .

Since we want to create a file, we must supply a file name and tell PHP that we want to write to the file.

Note: We have to tell PHP we are writing to the file, otherwise it will not create a new file.

Code: PHP

```
$ourFileName = "firstFile.txt";
$ourFileHandle = fopen($ourFileName, 'w') or die("can't open file");
fclose($ourFileHandle);
```

The file "firstFile.txt" should be created in the same directory where this PHP code resides. PHP will see that "firstFile.txt" does not exist and will create it after running this code.

There's a lot of information in those three lines of code.
1. $ourFileName = "firstFile.txt";

Here we create the name of our file, "firstFile.txt".

2. $ourFileHandle = fopen($ourFileName, 'w') or die("can't open file");

This bit of code actually has two parts.

First we use the function fopen and give it two arguments: our file name and we inform PHP

that we want to write by passing the character "w".

Second, the fopen function returns what is called a file handle, which will allow us to manipulate the file. We save the file handle into the $ourFileHandle variable.

3. fclose($ourFileHandle);

We close the file that was opened. fclose takes the file handle that is to be closed.

Permissions to PHP files

If you are trying to get this program to run and you are having errors, you might want to check that you have granted your PHP file access to write information to the hard drive. Setting permissions is most often done with the use of an FTP program to execute a command called CHMOD.

Use CHMOD to allow the PHP file to write to disk, thus allowing it to create a file.

Different Ways to Open a File

Below are the three ways to open a file .

Read: 'r'

Open a file for read only use. The file pointer begins at the front of the file.

Write: 'w'

Open a file for write only use. The data in the file is erased and you will begin writing data at the beginning of the file. The file pointer begins at the start of the file.

Append: 'a'

Open a file for write only use. However, the data in the file is preserved and you begin will writing data at the end of the file. The file pointer begins at the end of the file.

A file pointer is PHP's way of remembering its location in a file. When you open a file for reading, the file pointer begins at the start of the file. This makes sense because you will usually be reading data from the front of the file.

However, when you open a file for appending, the file pointer is at the end of the file, as you most likely will be appending data at the end of the file. When you use reading or writing functions they begin at the location specified by the file pointer.

File Close Description

In PHP it is not system critical to close all your files after using them because the server will close all files after the PHP code finishes execution. However the programmer is still free to make errors (i.e. editing a file that you forgot to close). You should close all files after you have finished with them because it's a good programming practice .

We had a call to the function fclose to close down a file after we were done with it.

Here we will repeat that example and discuss the importance of closing a file.

Code: PHP

```php
$ourFileName = "firstFile.txt";
$ourFileHandle = fopen($ourFileName, 'w') or die("can't open file");
fclose($ourFileHandle);
```

The function fclose requires the file handle that we want to close down.

In our example we set our variable "$fileHandle" equal to the file handle returned by the fopen function.

After a file has been closed down with fclose it is impossible to read, write or append to that file unless it is once more opened up with the fopen function.

File Upload

A very useful aspect of PHP is its ability to manage file uploads to your server. Allowing users to upload a file to your server opens a whole can of viruses, so please be careful when enabling file uploads.

Before you can use PHP to manage your uploads, you must first build an HTML form that lets users select a file to upload.

Code: HTML

```html
<form enctype="multipart/form-data" action="uploader.php" method="POST">
<input type="hidden" name="MAX_FILE_SIZE" value="100000" />
Choose a file to upload: <input name="Myuploadedfile" type="file"  />
<br /> <input type="submit" value="Upload File" /> </form>
```

Here is a brief description of the above code:

`enctype="multipart/form-data"` - Necessary for our to-be-created PHP file to function properly.

`action="uploader.php"` - The name of our PHP page that will be created.

`method="POST"` - Informs the browser that we want to send information to the server using POST.

`input type="hidden" name="MA...` - Sets the maximum allowable file size, in bytes, that can be uploaded.

We have set the max file size to 100KB in this example.

`input name="Myuploadedfile"` - uploadedfile is how we will access the file in our PHP script.

Save that form code into a file and call it upload.html.

After the user clicks submit, the data will be posted to the server and the user will be redirected to uploader.php. This PHP file is going to process the form data and do all the work. Now that we have the right HTML form we can begin to code the PHP script that is going to handle our uploads. Typically, the PHP file should make a decision with all uploads keep the file or throw it away.

A file might be thrown away from many reasons, including:

1.The file is too large and you do not want to have it on your server.
2.You wanted the person to upload a picture and they uploaded something else.
3.There were problems uploading the file and so you can't keep it.

When the uploader.php file is executed, the uploaded file exists in a temporary storage area on the server. If the file is not moved to a different location it will be destroyed.

To save our precious file we are going to need to make use of the $_FILES array..

The $_FILES array is where PHP stores all the information about files.

There are two elements of this array that we will need to understand Myuploadedfile is the reference we assigned in our HTML form. We will need this to tell the $_FILES array which file we want to play around with.

$_FILES['uploadedfile']['name'] - name contains the original path of the user uploaded file.
$_FILES['uploadedfile']['tmp_name'] - tmp_name contains the path to the temporary file that resides on the server.

The file should exist on the server in a temporary directory with a temporary name.

Now we can finally start to write a basic PHP upload manager script. Here is how we would get the temporary file name, choose a permanent name, and choose a place to store the file.

Code: PHP

```php
// Where the file is going to be placed
 $target_path = "uploads/";
 /* Add the original filename to our target path.
 Result is "uploads/filename.extension" */
 $target_path = $target_path . basename( $_FILES['uploadedfile']['name']);
 $_FILES['uploadedfile']['tmp_name'];
```

You will need to create a new directory in the directory where uploader.php resides, called "uploads", as we are going to be saving files there. We now have all we need to successfully save our file to the server.

$target_path contains the path where we want to save our file to. Now all we have to do is call the move_uploaded_file function and let PHP do its magic.

The move_uploaded_file function needs to know

1) The path of the temporary file
2) The path where it is to be moved to.

Code: PHP

```php
$target_path = "uploads/";
$target_path = $target_path . basename( $_FILES['uploadedfile']['name']);
if(move_uploaded_file($_FILES['uploadedfile']['tmp_name'], $target_path))
{
    echo "The file ".  basename( $_FILES['uploadedfile']['name']). " has been
uploaded";
}
```

```
else
{
    echo "There was an error uploading the file, please try again!";
}
```

If the upload is successful, then you will see the text

"The file filename has been uploaded".

This is because $move_uploaded_file returns true if the file was moved, and false if it had a problem.

If there was a problem then the error message **"There was an error uploading the file, please try again!"** would be displayed.

Source: http://www.go4expert.com/articles/file-version-management-php-t3068/