

3.8: File Interaction

We can use PHP to create and read files in a variety of formats, allowing us to create whole new pages, store information in document form, copy files, and plenty more. The first step in this process is creating a new file, or opening an existing file, that we want to work with. To do this, we use the function `fopen()` to declare the file's location and how we intend to interact with its contents.

File Permissions

PHP follows the Unix/Linux approach to file permissions, which is more granular to what Microsoft and Apple users are typically used to. In this approach, a particular file can have different permission levels depending on if the person editing the file is the owner, belongs to the same system group as the owner, or falls into the “anyone else” category. Within these three categories we can also specify whether or not the person is allowed to read, write, or execute the file, in any combination.

One of the methods used to depict permissions is with a string of letters and dashes, using R, W, and X to represent read write and execute. In this approach, three groupings of these letters are strung together in the order of owner, group, other, by read, write and execute. A file that everyone has full permissions would be represented by `rw-rw-rwx`, while a file where the owner can do anything, others in his group can read and execute, and anyone else can execute would be shown as `rx-rx-r-`. The dashes here indicate that the permission is lacking. Group membership refers to the group your account is associated with on the server, which can be anything the server is told to recognize like administrators, users, guests, professor, student, and so on. If the owner of our imaginary file was in the administrator group, other administrators could read and execute the file, where anyone in any other group would only be able to execute it without seeing its contents.

Understanding this structure is important to understanding why file open methods are necessary, and can also help us understand problems opening files when appropriate permissions are not used. Any time we open a file in PHP we need to use one of the following methods, which will determine what PHP lets us do with the file.

Table 3.8.1 PHP File Methods

mode	Description
<code>'r'</code>	Open for reading only; place the file pointer at the beginning of the file.
<code>'r+'</code>	Open for reading and writing; place the file pointer at the beginning of the file.
<code>'w'</code>	Open for writing only; place the file pointer at the beginning of the file and truncate the file to zero length. If the file does not exist, attempt to create it.
<code>'w+'</code>	Open for reading and writing; place the file pointer at the beginning of the file and truncate the file to zero length. If the file does not exist, attempt to create it.
<code>'a'</code>	Open for writing only; place the file pointer at the end of the file. If the file does not exist, attempt to create it.
<code>'a+'</code>	Open for reading and writing; place the file pointer at the end of the file. If the file does not exist, attempt to create it.
<code>'x'</code>	Create and open for writing only; place the file pointer at the beginning of the file. If the file already exists, the fopen() call will fail by returning FALSE and generating an error of level E_WARNING . If the file does not exist, attempt to create it. This is equivalent to specifying O_EXCL O_CREAT flags for the underlying open(2) system call.
<code>'x+'</code>	Create and open for reading and writing; otherwise it has the same behavior as <code>'x'</code> .

mode	Description
<code>'c'</code>	Open the file for writing only. If the file does not exist, it is created. If it exists, it is neither truncated (as opposed to <code>'w'</code>), nor the call to this function fails (as is the case with <code>'x'</code>). The file pointer is positioned on the beginning of the file. This may be useful if it is desired to get an advisory lock (see flock()) before attempting to modify the file, as using <code>'w'</code> could truncate the file before the lock was obtained (if truncation is desired, ftruncate() can be used after the lock is requested).
<code>'c+'</code>	Open the file for reading and writing; otherwise it has the same behavior as <code>'c'</code> .

[PHP.net \[CC-A 3.0\]](#)

File Actions

Assuming we want our file in the same folder as our page and that the web server has permission to create files in that location, we can start a new file with the following:

1. `$handler = fopen("ourFile.txt", 'x');`

If all was successful in creating our new file, the `$handler` variable would now represent the system's position in our open file as a reference. If the file already existed, we would have received an error (this keeps us from accidentally overwriting a file we wanted). If no permissions errors cropped up, you can now add content to your file. If you do have permission errors, you will need to change folders to one your web server can write to, or give your server permissions on that folder. Since this is an operating system task, you will need to find instructions on how to achieve this based on your OS type, version, and web server settings.

We can now add whatever we want to our file, so long as it results in valid content for the file type we are creating. For example, any HTML placed in a text file will appear as plain text, not a web page. We would need to create our file as `ourFile.html` for that type of content to render correctly. If we had a large block of text already stored in a variable called `content`, we can add it to our file by using `fwrite()`. Each time we call `fwrite`, the variable passed to it will be appended to what we have already sent. If we had opened an existing file, the content might be appended (`'a'`) or overwrite what exists (`'w+'`) depending on how we opened it. When we are done, we need to close the file, which actually writes the content using the `$handler` variable and saves it in our folder:

1. `fwrite($content);`
2. `fwrite($moreContent);`
3. `fclose($handler);`

If we browse to our file from our operating system, you should be able to open it in a text editor and see the text you stored in your `$content` variable.

Uploading Files

In order to allow users to upload files to our server, we have to create a folder that allows the web server to write to it, and make the following changes in our `php.ini` file:

1. `File_uploads = on`
2. `Upload_tmp_dir = [location of our upload folder]`
3. `Upload_max_filesize = [size in megs, i.e. 5M = 5 megs]`

After making these changes and restarting our web server, our users will be able to use upload form elements, which we create using an input with a type attribute of `file`:

1. `<input type="file" name="userUpload" id="userUpload">`

On the page that processes our form we can access the file (and information describing it) by using the reserved PHP array `$_FILES`:

1. `echo "File name:" . $_FILES["userUpload"]["name"] . "
";`

```
2. echo "Type:" . $_FILES["userUpload"]["type"] . "<br>";  
3. echo "Size:" . ($_FILES["userUpload"]["size"] / 1024) . "kB<br>";  
4. echo "Stored in:" . $_FILES["userUpload"]["tmp_name"];
```

3.8: File Interaction is shared under a [not declared](#) license and was authored, remixed, and/or curated by LibreTexts.

- **3.8: File Interaction** by [Michael Mendez](#) has no license indicated.