File Interaction

by Sophia

WHAT'S COVERED

In this lesson, you will be introduced to the concepts of file input and output (file I/O) from the programming perspective using PHP. You will learn about the process of opening a file stream and using that stream object to read data from the file. You will also learn how PHP can create new files and write data to new and existing files. Lastly, you will learn about the functions available in PHP to convert data objects into plain text for storage.

Specifically, this lesson will cover the following:

1. File Access Using PHP

File handling from the perspective of a programmer involves opening a file as a file stream. The stream is then used to either write data into or read data out of the stream of bits. Once the file has been opened as an in or output stream, we can then begin reading or writing data to or from the file. After the file has been opened, the **file stream object** (also referred to as a file pointer) provides different methods for reading a single character, word, or entire line of text into the application, as well as methods for writing data to the file. Lastly, once we are done writing or reading to or from the file, we then need to close the file stream, which releases the file and nullifies the file pointer itself.

PHP can be used to create, read, write, and remove text-based files within the webserver's file system. File access in PHP is restricted to the files within the webserver and is often further restricted to the directory on the server where the website is located.

However, the ability to create and write data to files on the webserver provides developers with potentially powerful operations. Files created and used with PHP can be simple and used to store user information and preferences, or they can be files containing entire newly created webpages based on the user's information, preferences, and other resources.

You will learn about the convenient PHP readfile() function, which can be easily used to read a file's entire contents and insert it into a variable or an echo command. Additionally, you will see how the PHP fopen() function is used to open files and then use the fread() or fwrite() functions to read data from or write data to the file.

TERM TO KNOW

File Stream Object
A programming object that represents the data going into or being read out of a digital file. Also called *file pointer*.

2. Opening and Reading Files With PHP

When reading from files, we have two options. If we need to read the entire file into our code, we can simply use the readfile() function and provide the file's relative path and filename. This function will quickly open the file and return all of the contents as output. When using the

readfile() function, we are not given the opportunity to process the file's contents incrementally. This means that if we are working with large files, we might run into memory issues as the entire file is loaded into memory at once.

EXAMPLE A file is read using PHP's readfile() function. The contents of "registeredUsernames.txt" are as follows:

```
jDoe
mSmith
tCarter
rSwanson
jSeinfeld
cKramer
```

EXAMPLE The PHP script code to read and output file contents is as follows:

```
<?php
readfile("registeredUsernames.txt");
?>
```

EXAMPLE When the contents are returned by PHP readfile, the output is as follows:

```
jDoe mSmith tCarter rSwanson jSeinfeld cKramer
```

While this is a quick way of outputting the contents of a file, it is apparent that the formatting is ignored, and the content is written to the DOM as a continuous string of text (ignoring newline and tab characters). When you need more control over how the contents of the file are handled, you will instead use a more explicit approach to reading the contents of the file.

In such cases, we need to first open the file and specify what mode we want the file opened in. The fopen() function is used to open a file as a file stream. This function takes up to four arguments, the first two being the filename and the mode. The filename will be a string pointing to the filename, including any relative directories. The mode is a string containing one or two characters.

The fopen() modes are as follows:

Mode	Description
r	This opens a file for reading only. The file pointer starts at the beginning of the file.
w	This opens a file for writing only. It erases the contents of the file or creates a new file if it does not exist. The file pointer starts at the beginning of the file.
a	This opens a file for writing only. The existing data in the file is preserved. The file pointer starts at the end of the file. It creates a new file if the file does not exist.
x	This creates a new file for writing only. It returns "FALSE" and an error if the file already exists.
r+	This opens a file for read/write. The file pointer starts at the beginning of the file.
w+	This opens a file for read/write. It erases the contents of the file or creates a new file if it does not exist. The file pointer starts at the beginning of the file.
a+	This opens a file for read/write. The existing data in the file is preserved. The file pointer starts at the end of the file. It creates a new file if the file does not exist.
x+	This creates a new file for read/write. It returns "FALSE" and an error if the file already exists.

As you can see from the mode options, we can manipulate an existing or nonexistent file in a variety of ways. Whether we need to create a new file or append to the end of an existing file, the variety of modes offers developers options for managing their files.

When we want to control how we read data into our PHP software, we will want to use one of PHP's different functions in the table below to retrieve the desired portions of the file's contents.

Method	Description
fgetc(resource $stream)	This reads the next character from the file and moves the read head ahead one character.
fgetcsv(resource $stream, ?int $length = null, string $separator = ",", string $enclosure = "\"", string $escape = "\\",)	This reads a line of comma-separated values, parses the values, and returns them as an array. The length argument is an integer representing how many bytes to read from the resource or null to remove the limit. The separator argument is a string representing the delimiter between words or values, as in the case of comma-separated documents.
fgets(resource $stream, ?int $length = null)	This reads a single line of text from the file.
fread(resource $stream, int $length)	This reads a specified amount of data from the file.

EXAMPLE When a file is read using PHP's fgets() function, the contents of "registeredUsernames.txt" are as follows:

```
jDoe
mSmith
tCarter
rSwanson
jSeinfeld
cKramer
```

EXAMPLE The PHP script code to read each line into a variable to be echoed is as follows:

```
<?php
      $fo = fopen("registeredUsernames.txt", "r");

      if (!$fo)
      {   echo "file could not be located."; }
      else
      {
        while (($buffer = fgets($fo)) !== false)
        {
          echo "$buffer <br>";
        }
      }
?>
```

EXAMPLE The output is as follows:

```
jDoe
mSmith
tCarter
rSwanson
```

```
jSeinfeld
cKramer
```

As you can see in the example, since we read one line at a time using fgets(), we are able to store it in a variable ($buffer) and then wrap it in any additional HTML code needed to provide a proper output. In this scenario, we are able to add a line break
 just after each line, and since each name is on its own line, we received a vertical list of usernames. However, what if a single line contained multiple individual values related to one entity? The fgets() function reads the entire line and would require us to then manually process the string of text to detect and separate each value. In such cases, we can use the fgetcsv() function, which would read a single line from the file and use the separator character to identify and parse the different values and return them as elements of an array.

EXAMPLE When fgetcsv() is used to get an array of values, the contents of “registeredUsernames.txt” are as follows:

```
jDoe,Manager,17.50,Sales
mSmith,CFO, 21.05,Accounting
tCarter,Coordinator,18.75,Operations
rSwanson,Coordinator,19.50,Operations
jSeinfeld,Comedian,17.85,Entertainment
cKramer,Comedian,17.85,Entertainment
```

EXAMPLE The PHP script code using fgetcsv() to read and parse values one line at a time into an array and generate an HTML table is as follows:

```
<?php
      $fo = fopen("registeredUsernames.txt", "r");

      if (!$fo)
      {   echo "file could not be located."; }
      else
      {
        echo "<table class='border'><tr><th>Name</th><th>Title</th><th>Pay Rate</th><th>Department</th></tr>";
        while (($buffer = fgetcsv($fo, null, ',')) !== false)
        {
          echo "<tr><td>$buffer[0]</td><td>$buffer[1]</td><td>$$buffer[2]</td><td>$buffer[3]</td> ";
        }
        echo "</table>";
      }
?>
```

EXAMPLE The HTML table with interpolated data from the file is returned. The output is as follows:

Name	Title	Pay Rate	Department
jDoe	Manager	$17.50	Sales
mSmith	CFO	$21.05	Accounting
tCarter	Coordinator	$18.75	Operations
rSwanson	Coordinator	$19.50	Operations
jSeinfeld	Comedian	$17.85	Entertainment
cKramer	Comedian	$17.85	Entertainment

In the above example, we added comma-separated data values for each user in the registeredUsernames.txt file. We then open the file, echo the start of an HTML table that includes appropriate headers, and then use a while loop that uses the fgetcsv() function with no size limit and uses a comma as the separator. This returns an array of the individual values on that line and stores it in $buffer. From there, we parse the four values into a table row. This step continues until we have no more lines of text to read. The loop ends, and we echo the closing of the HTML table.

We often use loops to handle reading data from a file and then other loops for processing the data. We can read one character or one line at a time, check it to see if it is what we were looking for, and then add it to a collection. In the previous example, we used a loop to read in each line as an array of values and, at the same time, parse the values into the newly generated HTML content.

On the flip side, when we have complex objects in the memory that need to be written to a file, we are likely to use a set of loops to convert each object's key and values into structured text, such as a comma-separated file. This way, we can save complex objects from the memory into long-term storage. We can then use the data read in from a file and pass it into an object constructor function to reinitialize each object back into the memory.

---

3. Creating and Writing Data to Files With PHP

Creating files using PHP is relatively easy since many of the modes of the fopen() function will automatically create the targeted file if it does not already exist. The following file modes will automatically create a file:

- w
- w+
- a
- a+
- x
- x+

In any of the above modes, if the specified file does not exist, PHP will attempt to create the file and then open it for writing.
As for writing data to a file, PHP only provides one function called fwrite().

KEY CONCEPT

One aspect of PHP is that there are several aliases for different functions and objects throughout the language. For example, there is an alias function for fwrite() called fputs(). The fputs() function is simply an alias of fwrite() and operates in exactly the same way.

This is a result of the PHP language going through changes over time and needing to maintain **backward compatibility**. Backward compatibility means that when a development language changes with the release of new versions, the changes have been made in such a way that applications written in the old version of the language will still operate.

Let us take a look at how we can manually serialize and then deserialize object instances into a text file using PHP. We will start with our PHP class objects that represent our employees.

EXAMPLE PHP class and instances

```
<?php
      class employee {
        public $name;
        public $title;
        public $payRate;
        public $department;

        function __construct($name, $title, $payRate, $department) {
          $this->name = $name;
          $this->title = $title;
```

```
        $this->payRate = $payRate;
        $this->department = $department;
      }
    }

    $staff = array(
      new employee("jDoe", "Manager", 17.5, "Sales"),
      new employee("mSmith", "CFO", 21.05, "Accounting"),
      new employee("tCarter", "Coordinator", 18.75, "Operations"),
      new employee("rSwanson", "Coordinator", 19.5, "Operations"));
?>
```

For demonstration purposes, we kept the class simple and only used public attributes. We then instantiated four employees into the $staff array. Next, we will open a write file stream to a new text file called “registeredUsers.dat.” Note that even though we chose a .dat file extension, the file will still be a plain text file.

EXAMPLE PHP manually serializes objects.

```
<?php
$dataout = fopen('registeredUsers.dat', 'w');

      foreach($staff as $emp)
      {
        fwrite($dataout, $emp->name . "," . $emp->title . "," . $emp->payRate . "," . $emp->department . "\n");
      }
    fclose($dataout);
?>
```

The segment opens the file in write-only mode and then enters a foreach loop, which takes each employee object and parses the attribute values into a single, comma-separated string within the fwrite() function. The resulting file is a comma-separated list of employees, with one employee on each line of the file.

EXAMPLE Sample PHP data file from objects

```
jDoe,Manager,17.5,Sales
mSmith,CFO,21.05,Accounting
tCarter,Coordinator,18.75,Operations
rSwanson,Coordinator,19.5,Operations
```

Next, we can read the text back in and then pass the values into the employee constructor to reinstate them into the memory.

EXAMPLE Sample PHP manually reinstates objects from a text file.

```
<?php
    $newStaff = array();

    $datain = fopen('registeredUsers.dat', 'r');
    while (($buffer = fgetcsv($fo, null, ',')) !== false)
    {
      array_push($newStaff, new employee($buffer[0], $buffer[1], $buffer[2], $buffer[3]));
    }

    fclose($datain);
?>
```

Finally, we created an empty array, opened the file in read mode, and then used fgetcsv() to retrieve each employee. We then use array_push to add a new element to the $newStaff array and then parse the employee's data values into the employee constructor. This effectively reinstated each employee back into the memory as employee objects.

TERM TO KNOW

Backward Compatibility
The ability of new software or hardware to support software or hardware that was created using older versions of the same software or hardware.

3a. Using PHP Serializer

When we want to write complex objects to file storage, there are pre-built programming libraries and packages that can be incorporated into your project that allow you to **serialize** data objects into JSON, XML, and other text-based structures. Serializing data simply means converting a virtual object into a string of text for storage and retrieval. This can greatly reduce the effort required on the part of the developer to write simple and complex data structures to a file. These libraries also allow you to **deserialize** information using their built-in methods. This is the opposite process where we read structured text data from a file and then use a deserialize function to reinstate the objects in memory.

PHP includes native functions for serializing and deserializing data into a generic string format. To serialize data using PHP's native functions, simply make a call to serialize() and pass in the data structure that you want it to serialize.

EXAMPLE The PHP serialize() function

```
<?php
      class employee {
        public $name;
        public $title;
        public $payRate;
        public $department;

        function __construct($name, $title, $payRate, $department) {
          $this->name = $name;
          $this->title = $title;
          $this->payRate = $payRate;
          $this->department = $department;
        }
      }

      $staff = array(
        new employee("jDoe", "Manager", 17.5, "Sales"),
        new employee("mSmith", "CFO", 21.05, "Accounting"),
        new employee("tCarter", "Coordinator", 18.75, "Operations"),
        new employee("rSwanson", "Coordinator", 19.5, "Operations"));

        $strData = serialize($staff);
        $dataOut = fopen("employeeData.dat", "w");
        fwrite($dataOut, $stfData);
        fclose($dataOut);
?>
```

EXAMPLE The employeeData.dat contents are as follows:
a:4:{i:0;O:8:"employee":4:{s:4:"name";s:4:"jDoe";s:5:"title";s:7:"Manager";s:7:"payRate";d:17.5;s:10:"department";s:5:"Sales";}i:1;O:8:"employee":4:{s:4:"name";s:6:"mSmith";s:5:"title";s:3:"CFO";s:7:"payRate";d:21.05;s:10:"department";s:10:"Accounting";}i:2;O:8:"employee":4:

{s:4:"name";s:7:"tCarter";s:5:"title";s:11:"Coordinator";s:7:"payRate";d:18.75;s:10:"department";s:10:"Operations";}i:3;O:8:"employee":4:{s:4:"name";s:8:"rSwanson";s:5:"title";s:11:"Coordinator";s:7:"payRate";d:19.5;s:10:"department";s:10:"Operations";}}

As you can see from the output, the serialize() function does serialize our complex data objects and even keeps track of the object's type (i.e., "employee"), the object's attribute's data types, and, of course, the individual attribute values. Now, let us read the data back in from the data file and use the unserialize() function to reinstate the array of employees.

In this example, we will use the fread() function to read in the entire file. Recall that you saw the use of the readfile() function, which can read the entire file. However, readfile() sends the data directly to the output, which is not what we need. Instead, we will use fread(), which takes two arguments, the file pointer, and the length in bytes to read. This function requires both arguments, and since we want the entire file, we will use the filesize() to get the correct number of bytes to read.

EXAMPLE The PHP unserialize() function

```
<?php

        $datain = fopen("employeeData.dat", "r");
        $strData = fread($datain, filesize($datain));
        fclose($datain);

        $staff = unserialize($strData);

        echo gettype($staff) . "<br>";
        echo gettype($staff[0]) . "<br>";
        echo get_class($staff[0]) . "<br>";

?>
```

EXAMPLE The expected output is as follows:

```
array
object
employee
```

In the example above, we opened an employee data tile, read it into the $strData buffer using fread($datain, filesize($datain)), and then passed the returned string through the unserialize() function to reinstate the array of employee objects. Next, we called the gettype() function on the $staff array itself, which returned "array." We did the same operation on one of the array's elements and received "object" in return. Lastly, we used a function called get_class() to identify the class name of the objects, which returned the correct class name of "employee."

KEY CONCEPT

You may have noticed that the native PHP serializer automatically keeps track of individual object class types. As a result, when you load an object back into your program using unserialize(), you can access the object's methods as well. So, if your employee class contained a toTable() method to echo the employee's data to an HTML table, after loading the objects from the data file, you will have access to the toTable() method.

However, if you use the PHP JSON serializer (json_encode()), you will lose this class data and will have to manually reinitialize each object using its own correct class constructor. This is due to the fact that JSON does not have a mechanism for tracking class types and instead only serializes the object's data values. When you use json_decode() to read the data back in, this method reinstates the objects as generic PHP objects, and they will have no association with the employee class itself. This is where the developer would have to then pass the deserialized data through a custom function, which parses the data back into the correct class constructor.

So, why use JSON? The primary reason you might choose the JSON serializer over PHP's native serializer is cross-platform compatibility. Since JSON is widely used across multiple platforms, if we needed our data to be accessible from other systems that may not use PHP, then

JSON would be a better option. If you know that your data will only be used within your application and on a PHP-compatible webserver, then you are safe to use the native serializer.

TERMS TO KNOW

Serialize
The process of converting programing objects and data structures into the plain-text equivalent for long-term storage.

Deserialize
The process of converting serialized plain-text data into the programming equivalent in the memory.

SUMMARY

In this lesson, you learned about the basic concepts of *file access using PHP*. ***Furthermore, you learned how to*** *open and read files using PHP* ***functions. You also learned how to use PHP to*** *create and write data to a file*. ***Lastly, you learned about the process of serializing and how you can use*** *PHP to serialize data* ***into a PHP-specific text format and JSON text file.***

Source: This Tutorial has been adapted from "The Missing Link: An Introduction to Web Development and Programming " by Michael Mendez. Access for free at https://open.umn.edu/opentextbooks/textbooks/the-missing-link-an-introduction-to-web-development-and-programming. License: Creative Commons attribution: CC BY-NC-SA.

TERMS TO KNOW

Backward Compatibility
The ability of new software or hardware to support software or hardware that was created using older versions of the same software or hardware.

Deserialize
The process of converting serialized plain-text data into the programming equivalent in the memory.

File Stream Object
A programming object that represents the data going into or being read out of a digital file. Also called *file pointer*.

Serialize
The process of converting programing objects and data structures into the plain-text equivalent for long-term storage.