Server-side
Web Development

Unit 7. PHP and files.

Fidel Oltra, Ricardo Sánchez

IES Jaume II El Just
Tavernes de la Valldigna
Departament d'Informàtica
Curs 2024-25

Index

1 PHP and files

We have seen in previous units how to persist data using session variables and databases. However, sometimes we will need to store certain data in files, without having to create a database. This happens when we only need to create one file with no much data, or even a single object.

Files can have a structure, like JSON or XML, or not. We will go over both types.

However, let's take a look first to the **file system** and how to work with it using PHP.

2 File system

In order to work with files, we need to know how to access to our file system and how to work with files and directories.

2.1 File system size

The **`disk_free_space`** function allows you to know the available space in the file system or in a partition. On the other hand, the **`disk_total_space`** function tells us the total space of the file system or disk partition. In both cases it does it in bytes. On Windows, for example, we would do:

```php
$bytesFree=disk_free_space("C:");
$bytesTotal=disk_total_space("C:");
$gbFree=round($bytesFree/pow(1024,3),2);
$gbTotal=round($bytesTotal/pow(1024,3),2);
echo "You have $gbFree GB free of $gbTotal GB";
```

In Linux we just should change `"C:"` by our file system root or partition, for example `"/"`.

```
You have 166.82 GB free of 476.31 GB
```

Figure 1: Available space with PHP

2.2 Some functions to get information about the file status

These are some functions to check the file status:

Function	Description
`file_exists($fileName)`	Returns true if the file exists
`is_file($fileName)`	Returns true if `$fileName` exists and is a file
`is_link($fileName)`	Returns true if `$fileName` exists and is a link
`is_executable($fileName)`	Returns true if the file exists and is executable
`is_readable($fileName)`	Returns true if the file exists and can be read
`is_writable($fileName)`	Returns true if the file exists and can be written
`is_uploaded_file($fileName)`	Returns true if `$fileName` has been uploaded with HTTP_POST

```
$fileName="ud07test.php";
if(file_exists($fileName)) {
  if(is_file($fileName)) {
    if(is_writable($fileName)) {
     echo "The file $fileName exists and can be written";
    }
  }
  else { echo "$fileName is not a file"; }
}
else { echo "$fileName doesn't exist"; }
```

2.3 Some functions to work with directories / folders

These are some of the most used functions to work with directories:

Function	Description
`is_dir($path)`	Returns true if `$path` exists and is a directory
`mkdir($path, $mode, $recursive)`	Creates a directory with the name specified by `$path`. The `$mode` variable define the Linux permissions, and it's ignored in Windows. If `$recursive` is true, the function allows us to create nested directories.

Function	Description
`rmdir($path)`	Deletes the directory specified by the `$path` variable
`chdir($path)`	Changes to the directory specified by the `$path` variable
`getcwd()`	Returns the current directory
`dirname($path,$levels)`	Returns the parent directory's path that is `$levels` (1 if `$levels` is not specified) up from the current directory
`basename($path,$suffix)`	Returns the last component of a path. If the name component ends with `$suffix` will be cut off
`pathinfo($path,$flags)`	Returns information about the given path. The information returned depends on the `$flags` value
`realpath($path)`	Returns the real path of the directory/file specified in `$path`. If none, returns the path of the current directory

For example:

```php
echo dirname("/usr/local/lib"); // returns "/usr/local"
echo dirname("/usr/local/lib",2); // returns "/usr"
echo basename("C://xampp/htdocs/curs2425"); // returns curs2425
echo basename("C://xampp/htdocs/curs2425","25"); // returns curs24
```

2.4 Some functions to work with permissions and ownership (in Linux)

Function	Description
`chgrp($fileName, $group)`	Changes the group of `$fileName` to the specified group
`lchgrp($linkName, $group)`	Changes the group of the `$linkName` symbolic link

Function	Description
`chmod($fileName, $mode)`	Change the permissions of the `$fileName`
`chown($fileName, $owner)`	Change the owner of the `$fileName`
`lchown($linkName, $owner)`	Change the owner of the `$linkName` symbolic link
`filegroup($fileName)`	Returns the group of the `$fileName` file or directory
`fileowner($fileName)`	Returns the owner of the `$fileName` file
`fileperms($fileName)`	Returns the permissions of the `$fileName` file or directory

2.5 Some functions to make generic operations with files

These are some functions to make usual operations with files like copy, delete or move them.

Function	Description
`copy($sourcePath,$destinationPath)`	Copies the file specified in `$sourcePath` to `destinationPath`
`rename($sourcePath, $destinationPath)`	Renames the specified file, or moves it to the `$destinationPath` if it's different from `$sourcePath`
`unlink($fileName)`	Deletes the file or link specified by `$fileName`
`move_uploaded_file($fileName, $destinationPath)`	Moves the specified file uploaded with HTTP_POST to the `$destinationPath`

2.6 Some functions to get more information about a file

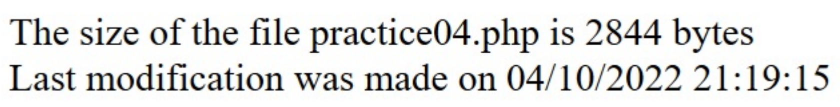
These functions offer information about some file features such as size, last access or last modification, among others.

Function	Description
`fileatime($fileName)`	Returns the date of the last access
`filemtime($fileName)`	Returns the date of the last modification
`filesize($fileName)`	Returns the size of the file in bytes
`filetype($fileName)`	Returns the type of the file (*file*, *dir*, *link*…)
`stat($fileName)`	Returns an array with information about the file
`lstat($linkName)`	Returns an array with information about the symbolic link
`fstat($pointer)`	Returns an array with information about the file opened by the pointer

Let's see an example:

```php
$fileName = "../practice04.php";
$fileSize = filesize($fileName);
$lastChange = date("d/m/Y H:i:s",filemtime($fileName));
echo "The size of the file ".basename($fileName)." is $fileSize bytes";
echo "<br>";
echo "Last modification was made on $lastChange";
```

When we run the script we will get this result:



```
The size of the file practice04.php is 2844 bytes
Last modification was made on 04/10/2022 21:19:15
```

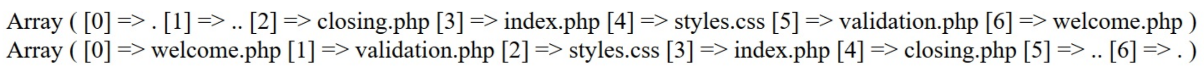
Figure 2: File info

2.7 Reading the content of a directory

We can go through the content of a directory by using the function **`scandir($directory)`**, which returns an array with the files and directories inside **`$directory`**.

```php
$list = scandir("../practice06"); // alfabetical ascending
print_r($list);
echo "<br/>";
$list = scandir("../practice06", SCANDIR_SORT_DESCENDING); // alfabetical
↪  descending
print_r($list);
```

The output will be:



```
Array ( [0] => . [1] => .. [2] => closing.php [3] => index.php [4] => styles.css [5] => validation.php [6] => welcome.php )
Array ( [0] => welcome.php [1] => validation.php [2] => styles.css [3] => index.php [4] => closing.php [5] => .. [6] => . )
```

Figure 3: Result of scandir function

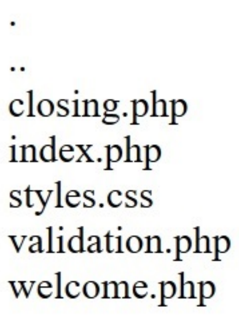
We can use other functions as well:

- **`opendir($directory)`** Opens a directory handle
- **`readdir($dir_handle)`** Reads next entry from the directory opened with **`$dir_handle`**
- **`closedir($dir_handle)`** Closes the directory handle
- **`rewinddir($dir_handle)`** Rewinds the directory handle to the beginning of the directory

This is the same example seen above but using `opendir` and `readdir`:

```php
$directory = "../practice06";
$dirHandler = opendir($directory);
if($dirHandler) {
    while(FALSE !== ($nextFile = readdir($dirHandler))) {
         echo $nextFile."<br/>";
    }
}
closedir($dirHandler);
```

Output:



```
.
..
closing.php
index.php
styles.css
validation.php
welcome.php
```

Figure 4: Directory content with a handler

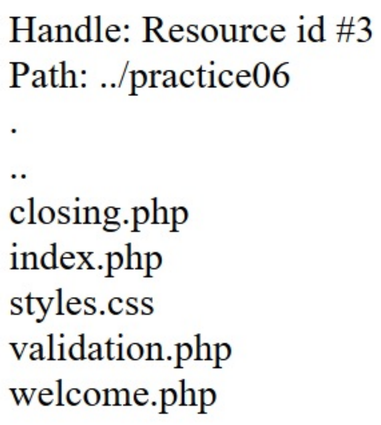
A third way of working with the directory content is using the **`Directory`** class.

- Function **`dir($directory)`** Creates an object of the Directory class
- Method **`read()`** Reads the next entry of the directory
- Method **`close()`** Closes the handler

Again the same example:

```php
$d = dir("../practice06");
echo "Handle: " . $d->handle . "<br/>";
echo "Path: " . $d->path . "<br/>";
while (false !== ($entry = $d->read() )) {
   echo $entry."<br/>";
}
$d->close();
```

And the output:



```
Handle: Resource id #3
Path: ../practice06
.
..
closing.php
index.php
styles.css
validation.php
welcome.php
```

Figure 5: Directory content with dir()

3 Working with text files

PHP offers some functions to work with text files and do the basic operations of create/open/read/write.

3.1 Opening a text file

To open a text file we will need a **handler**. The handler is created using the function **`fopen($fileName, $mode)`**. The mode can be:

Mode	Description
`r`	Read-only. Place the file pointer to the beginning of the file. If the file doesn't exist, we'll get a warning
`w`	Write-only. Place the file pointer to the beginning of the file. If the file doesn't exist, will be created. If the file exists, it will be overwritten
`a`	Append-only. Place the file pointer to the end of the file. If the file doesn't exist, it will be created
`x`	Like 'w' but if the file exists we'll get a warning
`c`	Like 'w' but the file is not overwritten until we add content
`r+`	Like 'r' but allows writing as well
`w+`	Like 'w' but allows reading as well
`a+`	Like 'a' but allows reading as well

Mode	Description
x+	Like ‘x’ but allows reading as well
c+	Like ‘c’ but allows reading as well

We will usually check if the file exists before opening it with ‘w’, ‘a’ or ‘r’ mode.

```
if (!file_exists($fileName)) {
  // the file doesn't exist
  $f = fopen($fileName,"w"); // so we create the file
} else {
  // the file already exists
  $f = fopen($fileName,"a"); // opened to append content
}
```

3.2 Reading the content of a text file

We can read the content of a text file all at once or by pieces.

3.2.1 Reading all the content of a text file

These are some functions that allow us to read the whole content of a file with a single operation:

Function	Description
`readFile($fileName)`	Reads the whole content of the file and sends it to the default output buffer
`file_get_contents($fileName, $init, $length)`	Reads the content of the file beginning in the `$init` position (optional, by default from the beginning) and for the `$length` number of characters (optional, by default to the end of the file). Returns a string with the read content
`file($fileName)`	Reads the content of the file and returns an array with a position for each file line

Let’s see an example using the three functions:

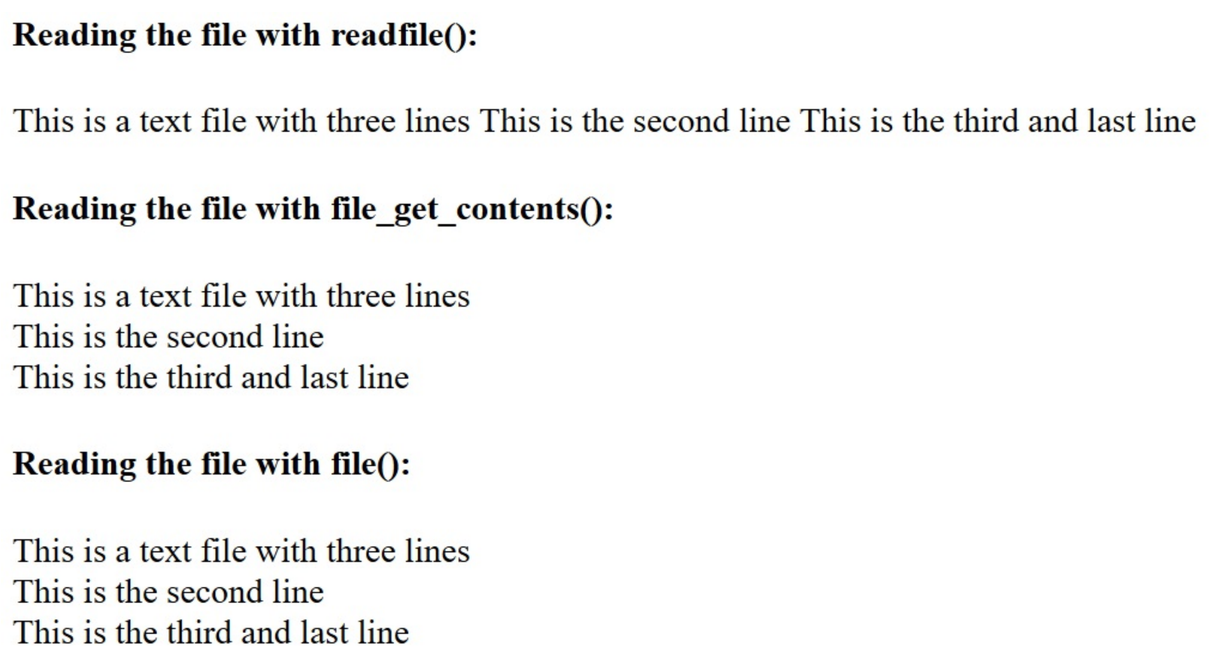
```php
$fileName = "textfile.txt";

echo "<h4>Reading the file with readfile():</h4>";
readfile($fileName);

echo "<h4>Reading the file with file_get_contents():</h4>";
$string = file_get_contents($fileName);
echo nl2br($string); // to change the text line breaks to <br/> tags

echo "<h4>Reading the file with file():</h4>";
$array = file($fileName);
foreach($array as $line) { echo $line."<br>"; }
```

The output:



```
Reading the file with readfile():

This is a text file with three lines This is the second line This is the third and last line

Reading the file with file_get_contents():

This is a text file with three lines
This is the second line
This is the third and last line

Reading the file with file():

This is a text file with three lines
This is the second line
This is the third and last line
```

Figure 6: Reading the content of a file

3.2.2 Reading the content of a text file by pieces

Sometimes we will need to read the file line by line, word by word or even char by char.

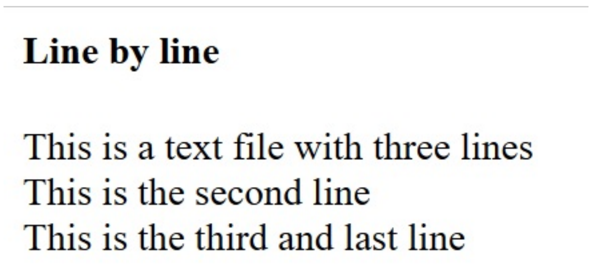
If we have opened the file with some of the modes that allow reading, we can use these functions:

Function	Description
`fread($fileHandler, $size)`	Read `$size` bytes of the file opened with the handler and returns the content
`fgets($fileHandler)`	Reads and returns a line from the file opened with the handler, and moves the pointer to the next line
`fgetc($fileHandler)`	Reads and returns a byte from the file opened with the handler, and moves the pointer to the next char
`fgetcsv($fileHandler, $length, $separator, $enclosure)`	Reads a line from the file opened with the handler, and search for CSV fields using the `$separator` char (optional) and the `$enclosure` char (optional) and then returns an array
`feof($fileHandler)`	Returns true if we reach the end of the file

Let's read our example file line by line:

```php
$fileName = "textfile.txt";
echo "<h4>Line by line</h4>";
$f = fopen($fileName,"r");
while (!feof($f)) {
    $line = fgets($f);
    echo nl2br($line);
}
fclose($f);
```

The output:



```
Line by line

This is a text file with three lines
This is the second line
This is the third and last line
```

Figure 7: File content

3.2.3 Reading CSV files

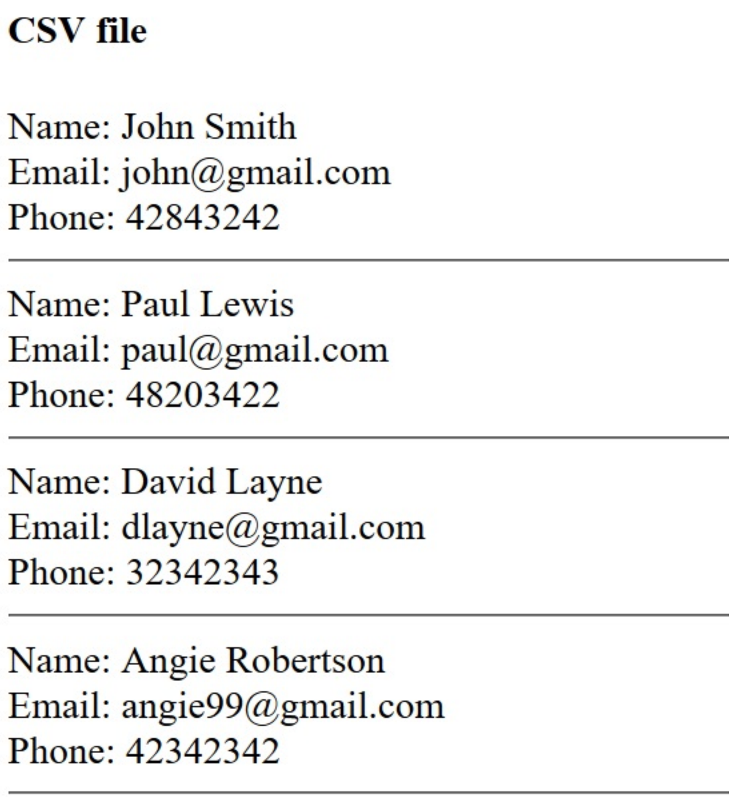
Now, let's take a look to this file:

```
"John","Smith","john@gmail.com","42843242"
"Paul","Lewis","paul@gmail.com","48203422"
"David","Layne","dlayne@gmail.com","32342343"
"Angie","Robertson","angie99@gmail.com","42342342"
```

This is a CSV formatted file, so we should use the `fgetcsv()` function instead.

```php
$fileName = "csvfile.txt";
if(file_exists($fileName)) {
    echo "<h4>CSV file</h4>";
    $f = fopen($fileName,"r");
    while (!feof($f)) {
        $array = fgetcsv($f);
        echo "Name: ".$array[0]." ".$array[1]."<br>";
        echo "Email: ".$array[2]."<br>";
        echo "Phone: ".$array[3]."<br>";
        echo "<hr>";
    }
    fclose($f);
}
else { echo "<h4>The file $fileName doesn't exist</h4>"; }
```

The output:



```
CSV file

Name: John Smith
Email: john@gmail.com
Phone: 42843242

Name: Paul Lewis
Email: paul@gmail.com
Phone: 48203422

Name: David Layne
Email: dlayne@gmail.com
Phone: 32342343

Name: Angie Robertson
Email: angie99@gmail.com
Phone: 42342342
```

Figure 8: Reading a CSV file

Another way to read a CSV file:

```php
<?php
$row = 1;
if (($handle = fopen("test.csv", "r")) !== FALSE) {
    while (($data = fgetcsv($handle, 1000, ",")) !== FALSE) {
        $num = count($data);
        echo "<p> $num fields in line $row: </p>\n";
        $row++;
        for ($c=0; $c < $num; $c++) {
            echo $data[$c] . "<br>\n";
        }
    }
    fclose($handle);
}
?>
```

The default separator is the **comma** `,` and the default enclosure is **double quote** `"`.

3.3 Writing on a text file

To write on a text file we should open it with a mode that allows writing or appending content.

To write content on a text file we can use the function `fwrite()` or its alias `fputs()`.

```php
$fileName = "newfile.txt";
$f = fopen($fileName,"w"); // the file is created with write permission
fwrite($f,"Line number 01".PHP_EOL);
fwrite($f,"Line number 02".PHP_EOL);
fclose($f);

$f = fopen($fileName,"a"); // now the file is opened to append lines
fwrite($f,"Line number 03".PHP_EOL);
fwrite($f,"Line number 04".PHP_EOL);
fclose($f);
```

> The **`PHP_EOL`** predefined constant is used for adding an end of line.

If you run the script you can check by yourself that the file `newfile.txt` has been created with the specified content.

> The function `fflush()` can be used to force the writing of the buffered output, just in case we miss the last line in the file

3.4 Reading and writing with the same handler

To read and write a file using a single handler, we should open the file with a mode that allow both operations: w+, r+, a+…

The operations are very similar, but as we can read and write without closing the file and opening it again, we need to know how to move the pointer if needed.

Function	Description
`rewind($fileHandler)`	Rewinds the pointer to the beginning of the file

Function	Description
`fseek($fileHandler,$offset,$init)`	Moves the pointer `$offset` bytes from the point specified by `$init`: **SEEK_CUR** (current plus offset),**SEEK_END** (end-of-file plus offset) or, by default, from the beginning of the file plus offset

We can add a 5th line to our example file, and then rewind the pointer to display the updated content.

```php
$fileName = "newfile.txt";
$f = fopen($fileName,"a+"); // to add content at the end of the file
fwrite($f,"Line number 05".PHP_EOL);
rewind($f);
while (!feof($f)) {
  echo fgets($f)."<br>";
}
fclose($f);
```

You can check that a new line has been added to the file.

4 PHP and JSON files

JSON files are plain text files, so we can work with them using the functions seen above. In fact, we can use those functions along with `json_decode()`, `json_encode()` ans `json_validate()` to work with JSON content. Let's see how to do it.

4.1 Creating a JSON file

As JSON format works with **key->value** pairs, we can easily create the JSON file from an associative array with the keys and values we need.

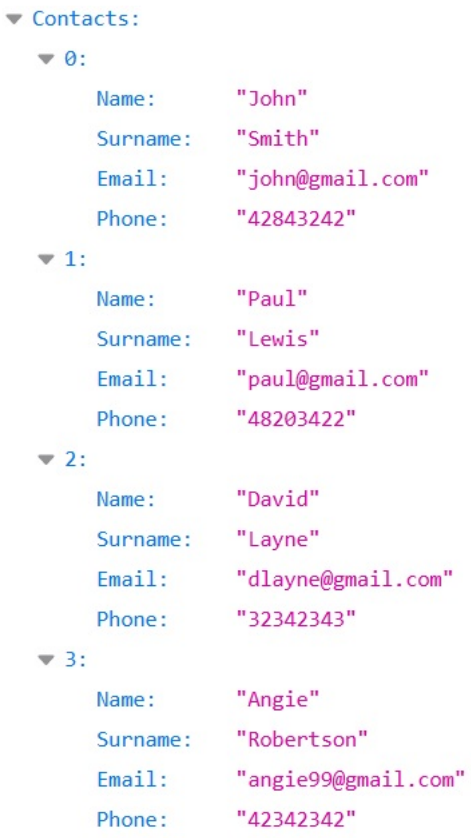
Let's create again our contacts file, but this time as a JSON file:

```
$contacts=array("Contacts"=>array(
    array("Name"=>"John", "Surname"=>"Smith", "Email"=>"john@gmail.com",
    ↪  "Phone"=>"42843242"),
    array("Name"=>"Paul", "Surname"=>"Lewis", "Email"=>"paul@gmail.com",
    ↪  "Phone"=>"48203422"),
    array("Name"=>"David", "Surname"=>"Layne", "Email"=>"dlayne@gmail.com",
    ↪  "Phone"=>"32342343"),
    array("Name"=>"Angie", "Surname"=>"Robertson",
    ↪  "Email"=>"angie99@gmail.com", "Phone"=>"42342342")
));

$jsonFile = json_encode($contacts);
$bytes = file_put_contents("contacts.json", $jsonFile);
if($bytes==0) {
  echo "Write error";
} else {
  echo "Succesful!";
}
```

> The **`file_put_contents($filename, $data)`** function is identical to calling `fopen()`, `fwrite()` and `fclose()` successively to write the `$data` data to the `$filename` file.

You can check that the `contacts.json` file has been created and is well formed.



```
▼ Contacts:
  ▼ 0:
      Name:      "John"
      Surname:   "Smith"
      Email:     "john@gmail.com"
      Phone:     "42843242"
  ▼ 1:
      Name:      "Paul"
      Surname:   "Lewis"
      Email:     "paul@gmail.com"
      Phone:     "48203422"
  ▼ 2:
      Name:      "David"
      Surname:   "Layne"
      Email:     "dlayne@gmail.com"
      Phone:     "32342343"
  ▼ 3:
      Name:      "Angie"
      Surname:   "Robertson"
      Email:     "angie99@gmail.com"
      Phone:     "42342342"
```

Figure 9: JSON file

4.2 Reading a JSON file.

Reading and working with a JSON file is very easy too, because we just need to do the reverse operation: convert the JSON file to an associative array. In fact, the JSON file is converted into an object that has a property, the one we defined as the first key ("Contacts", in this case), that contains the array.

```
$fileName = "contacts.json";
if(file_exists($fileName)) {
    $jsonFile = file_get_contents($fileName);
    $data = json_decode($jsonFile); // now $data is an object, not an array
    // but the property "contacts" is the array we are looking for
    foreach($data->Contacts as $contact) {
        foreach($contact as $key=>$value) {
          echo "<strong>$key: </strong>$value ";
        }
        echo "<br>";
    }
} else {
  echo "The file $fileName doesn't exist";
}
```

4.3 Validating JSON

Since **PHP 8.3** we have the function `json_validate()` that checks if a string contains valid JSON content:

```
var_dump(json_validate('{ "test": { "foo": "bar" } }')); //Outputs:
↪   bool(true)
var_dump(json_validate('{ "": "": "" } }'));  //Outputs: bool(false)
```

We can rewrite the previous example validating the Json file before decode it:

```
$fileName = "contacts.json";
if(file_exists($fileName)) {
    $jsonFile = file_get_contents($fileName);
    if(json_validate($jsonFile)) {
      $data = json_decode($jsonFile);
      ...
```

Before PHP 8.3, we can simulate the behavior of `json_validate()` with a custom function:

```php
function json_validate(string $string): bool {
    json_decode($string);
    return json_last_error() === JSON_ERROR_NONE;
}

if(json_validate($jsonFile)) {
  ...
```

5 Transfering files

Sometimes we need to transfer the files generated to or from the server. Let's see the most common ways.

5.1 Uploading files

To upload a file we need a form with an input of type file:

```html
<form method="post" action="destination.php" enctype="multipart/form-data">
  <label> Select file to upload:
    <input type="file" name="fileToUpload">
  </label>
  <input type="submit" value="Upload file" name="submitFile">
</form>
```

The form must have the **POST method** and the **enctype** (*encryption type*) ***"multipart/form-data"***. Without these requirements, the file upload will not work.

> **Important**: Before uploading files, probably you need to increase the limits for uploading files in `php.ini`, otherwise the files won't be uploaded:
>
> ```
> upload_max_filesize = 50M
> post_max_size = 50M
> ```
>
> After increasing the limits, restart the apache server.

The files are transferred in the superglobal array **`$_FILES`**, which has the next elements:

```
Array
(
  [fileToUpload] => Array
    (
      [name] => file.txt //Name of the file
      [full_path] => file.txt //Full path of all files, used when upload a folder
      [type] => text/csv // File type
      [tmp_name] => /tmp/phpp6nt4o // Temporary path of the file
      [error] => 0 //0 means no errors
      [size] => 903 // the file size in bytes
    )
)
```

When a file is uploaded, it's stored in a temporary path, indicated by the field `tmp_name`. If we only need to read the file, we can do it directly. But if we want to store the file in the server, we need to copy the file to the destination folder.

Before that, we need to make some checks:

```php
$target_dir = "uploads/"; //Destination folder. Check the permissions!
$target_file = $target_dir . basename($_FILES["fileToUpload"]["name"]); //Path of the file in the server
$uploadOk = true;
$fileType = strtolower(pathinfo($target_file,PATHINFO_EXTENSION)); //File extension in lower case

// Check if file already exists
if (file_exists($target_file)) {
  echo "Sorry, file already exists.";
  $uploadOk = false;
}

// Check file size
if ($_FILES["fileToUpload"]["size"] > 5000000) { //5MB
  echo "Sorry, your file is too large.";
  $uploadOk = false;
}

// Allow certain file formats
```

```php
if ($fileType != "csv") {
    echo "Sorry, only CSV files are allowed.";
    $uploadOk = false;
}
```

Then, if there were no errors, copy the file to the destination folder, with the **`move_uploaded_file ($filename, $destination)`** function:

```php
// Check if $uploadOk is set to 0 by an error
if (!$uploadOk) {
  echo "Sorry, your file was not uploaded.";
// if everything is ok, try to upload file
} else {
  if (move_uploaded_file($_FILES["fileToUpload"]["tmp_name"],
  ↪  $target_file)) {
    echo "File uploaded";
  } else {
    echo "Sorry, there was an error uploading your file.";
  }
}
```

Instead of moving the file, we can do other operations, such as reading the file and inserting their elements in a database.

5.2 Downloading files

The easiest way to download a file is to providing a link to it:

```php
echo "<a href="$fileName"> Download file </a>";
```

But if we want to automatic download a file (for example, after a data export) we need to create an HTTP header and use the **`readfile()`** function:

```php
if (file_exists($fileName)) {
  //Define header information
  header('Content-Description: File Transfer');
  header('Content-Type: application/octet-stream');
  header("Cache-Control: no-cache, must-revalidate");
  header("Expires: 0");
  header('Content-Disposition: attachment; filename="' .
  ↪  basename($fileName) . '"');
  header('Content-Length: ' . filesize($fileName));
  header('Pragma: public');

  //Clear system output buffer
  flush();
  //Read the file
  readfile($fileName);
  //Delete the file from server
  unlink($fileName);
}
```

> **Remember**: The `header` function must be used before showing any content on your page, otherwise that content will be added to the download file, corrupting the request's header.

5.3 Storing images

It is very common to store images (or any other file types) in our app. There are 2 different aproaches:

- Storing the images in the server's disk file system: it's better when we need to store a large quantity of images, like in galleries. This approach reduces the maintenance of the database, but requires a good synchronization between database and file system content.
- Storing the images in the database: it's better when each image is associated with an entity instance (for example, a contact's profile picture). It requires less maintenance, but the database size increases significantly, making backups to take a long time.

For the first approach, follow the process explained in the **Uploading files** section. Once the file is stored in the file system, store the file's route in the database, in a VARCHAR field.

For storing the file in the database we need a LONGBLOB field on the table where we want to store the image:

```
+----------+--------------+------+-----+---------+----------------+
| Field    | Type         | Null | Key | Default | Extra          |
+----------+--------------+------+-----+---------+----------------+
...
| image    | longblob     | YES  |     | NULL    |                |
+----------+--------------+------+-----+---------+----------------+
```

To upload the image, we need a form with a file input and a submit button. We also have added an `img` tag to show the image:

```
<form method="post" action="<?php echo
 ↪  htmlspecialchars($_SERVER["PHP_SELF"]);?>"
 ↪  enctype="multipart/form-data">
    <!-- Other inputs -->

    <label>Image: <br>
        <img width="100px" src="data:image/jpg;charset=utf8;base64,<?=
         ↪  !is_null($contact->getImage()) ?
         ↪  base64_encode($contact->getImage()) : '' ?>"
             alt="login picture"> <span class="error"> <?= $imageErr; ?>
 ↪  </span> <br>
        <input type="file" name="image" >
    </label>

    <input type="submit" name="submit" value="Save">
</form>
```

Note how we get the image from a `$contact` object and encode it in order to be shown.

When the form or another script gets the POST request, we need to check the image (size and filetype), generate the error message, if any, and store the image in the `$contact` object:

```
if (count($_FILES) > 0) {
  if (isset($_FILES['image']['tmp_name']) && $_FILES['image']['size'] > 0)
   ↪  {
      $fileType =  strtolower(pathinfo(basename($_FILES['image']['name']),
       ↪  PATHINFO_EXTENSION));

      // Check file size
      if ($_FILES["image"]["size"] > 5000000) { //5MB
          $imageErr = "* File too large. Max 5MB.";
          $err = true;
      }

      //Chek type
      if(!in_array($fileType, ['jpg','png','jpeg','gif', 'bmp']) ){
          $imageErr = '* Only JPG, JPEG, PNG, BMP & GIF files are
           ↪  allowed.';
          $err = true;
      }

      if (!$err) {
          $imgData = file_get_contents($_FILES['image']['tmp_name']);
          $contact->setImage($imgData);
      }
  }
}
```

The methods for store the image in the database are similar to any other data types:

```
public static function insert($object): int
{
  $conn = DBConnection::connectDB();
  if (!is_null($conn)) {
      $stmt = $conn->prepare("INSERT INTO contacts (image) VALUES
       ↪  (:image)");
      $stmt->execute(['image'=>$object->getImage()]);
      echo $stmt->rowCount(); //Return the number of rows affected
  }
  return 0;
}

public static function update($object): int
{
    $conn = DBConnection::connectDB();
    if (!is_null($conn)) {
        $stmt = $conn->prepare("UPDATE contacs SET image=:image WHERE
         ↪  id=:id");
        $stmt->execute(['id'=>$object->getId(),
         ↪  'image'=>$object->getImage()]);
        return $stmt->rowCount(); //Return the number of rows affected
    }
    return 0;
}
```

Don't forget to increase the `php.ini` upload max límits.

6 Cookies

`HTTP cookies` are small files created by a web server while a user is browsing a website and placed on the user's computer or other device by the user's web browser. Cookies are placed on the device used to access a website, and more than one cookie may be placed on a user's device during a session.

Cookies serve useful and sometimes essential functions on the web. They enable web servers to store stateful information, such as items added in the shopping cart in an online store, the user name or the user's preferred language.

6.1 Setting cookies with PHP

To set (store) a cookie we use the function **`setcookie`**:

```
setcookie(
  string $name,
  string $value = "",
  int $expires_or_options = 0,
  string $path = "",
  string $domain = "",
  bool $secure = false,
  bool $httponly = false
): bool
```

The most important arguments are:

- **name**: The name of the cookie.
- **value**: The value of the cookie. This value is stored on the client's computer; do not store sensitive information. Assuming the name is 'cookiename', this value is retrieved through $_COOKIE['cookiename']
- **expires_or_options**: The time the cookie expires. This is a Unix timestamp. If set to 0, or omitted, the cookie will expire at the end of the session (when the browser closes).

If `setcookie()` successfully runs, it will return `true`. This does not indicate whether the user accepted the cookie.

We usually set a cookie in this way:

```
setcookie("UserName", $userName, time() + 3660*24*30); //expires in 30 days
```

> **Note**: The `setcookie()` function must appear BEFORE the `<html>` tag.

To **modify a cookie**, just set (again) the cookie using the `setcookie()` function:

```
setcookie("UserName", $newUserName, time() + 3660*24*30);
```

To **delete a cookie**, use the `setcookie()` function with an expiration date in the past:

```
// set the expiration date to one hour ago
setcookie("UserName", $userName, time() - 3660);

//or
setcookie("UserName", $userName, 1);
```

With `setcookie()` the value of the cookie is automatically URLencoded (replacing HTML special chars) when sending the cookie, and automatically decoded when received. To prevent URLencoding, use **`setrawcookie()`** instead.

When debbugging, you can see the cookies for each page in your browser's **developer console**, storage section.

6.2 Reading cookies

We can read the cookies with the **`$_COOKIE`** superglobal:

```
if(isset($_COOKIE["UserName"])) {
  $userName = $_COOKIE["UserName"];
}
```

> **Tip**: When working with cookies, it's a good idea to test the script in a private window on the browser.