PHP PROGRAMMING COOKBOOK

Boosting your Web Development Career

WEB CODE GEEKS

WEB CODE GEEKS
WEB DEVELOPERS RESOURCE CENTER

PHP Programming Cookbook

Contents

Copyright (c) Exelixis Media P.C., 2016

All rights reserved. Without limiting the rights under
copyright reserved above, no part of this publication
may be reproduced, stored or introduced into a retrieval system, or
transmitted, in any form or by any means (electronic, mechanical,
photocopying, recording or otherwise), without the prior written
permission of the copyright owner.

Preface

PHP is a server-side scripting language designed for web development but also used as a general-purpose programming language. Originally created by Rasmus Lerdorf in 1994, the PHP reference implementation is now produced by The PHP Group. PHP originally stood for Personal Home Page, but it now stands for the recursive backronym PHP: Hypertext Preprocessor.

PHP code may be embedded into HTML code, or it can be used in combination with various web template systems, web content management systems and web frameworks. PHP code is usually processed by a PHP interpreter implemented as a module in the web server or as a Common Gateway Interface (CGI) executable. The web server combines the results of the interpreted and executed PHP code, which may be any type of data, including images, with the generated web page. (Source: https://en.wikipedia.org/wiki/PHP)

In this ebook, we provide a compilation of PHP based examples that will help you kick-start your own web projects. We cover a wide range of topics, from HTML tables and files uploading, to SOAP clients and Curl command execution. With our straightforward tutorials, you will be able to get your own projects up and running in minimum time.

About the Author

WCGs (Web Code Geeks) is an independent online community focused on creating the ultimate Web developers resource center; targeted at the technical architect, technical team lead (senior developer), project manager and junior developers alike.

WCGs serve the Web designer, Web developer and Agile communities with daily news written by domain experts, articles, tutorials, reviews, announcements, code snippets and open source projects.

You can find them online at https://www.webcodegeeks.com/

Chapter 1

PHP Tutorial for Beginners

PHP (recursive acronym for PHP: Hypertext Preprocessor) is a widely-used open source general-purpose scripting language that is especially suited for web development and can be embedded into HTML. It allows web developers to create dynamic content that interacts with databases. PHP is basically used for developing web based software applications.

PHP is mainly focused on server-side scripting, so you can do anything any other CGI program can do, such as collect form data, generate dynamic page content, or send and receive cookies. Code is executed in servers, that is why you'll have to install a sever-like environment enabled by programs like XAMPP which is an Apache distribution.

1.1 Introduction

Back in 1994, Rasmus Lerdorf unleashed the very first version of PHP. However, now the reference implementation is now produced by The PHP Group. The term PHP originally stood for Personal Home Page but now it stands for the recursive acronym: Hypertext Preprocessor. PHP 4 and PHP 5 are distributed under the PHP Licence v3.01, which is an **Open Source** licence certified by the Open Source Initiative.

1.1.1 Where is PHP used?

Three are the main areas where PHP scripts are used:

- **Server-side scripting**

This is the most used and main target for PHP. You need three things to make this work the way you need it. The PHP parser (CGI or server module), a web server and a web browser. You need to run the web server where. You can access the PHP program output with a web browser, viewing the PHP page through the server. All these can run on your home machine if you are just experimenting with PHP programming.

- **Command line scripting**

You can make a PHP script to run it without any server or browser. You only need the PHP parser to use it this way. This type of usage is ideal for scripts regularly executed using cron (on Linux) or Task Scheduler (on Windows). These scripts can also be used for simple text processing tasks.

- **Writing desktop applications**

PHP may not the very best language to create a desktop application with a graphical user interface, but if you know PHP very well, and would like to use some advanced PHP features in your client-side applications you can also use PHP-GTK to write such programs. You also have the ability to write cross-platform applications this way.

In this article, we'll have a detailed look at the server-side scripting using PHP.

1.1.2 Why PHP?

There stand convincing arguments for all those who wonder why PHP is so popular today:

- **Compatible with almost all servers used nowadays**

A web server is an information technology that processes requests via HTTP, the basic network protocol used to distribute information on the World Wide Web. There exist many types of web servers that servers use. Some of the most important and well-known are: Apache HTTP Server, IIS (Internet Information Services), lighttpd, Sun Java System Web Server etc. As a matter of fact, PHP is compatible with all these web servers and many more.

- **PHP will run on most platforms**

Unlike some technologies that require a specific operating system or are built specifically for that, PHP is engineered to run on various platforms like Windows, Mac OSX, Linux, Unix etc)

- **PHP supports such a wide range of databases**

An important reason why PHP is so used today is also related to the various databases it supports (is compatible with). Some of these databases are: DB++, dBase, Ingres, Mongo, MaxDB, MongoDB, mSQL, Mssql, MySQL, OCI8, PostgreSQL, SQLite, SQLite3 and so on.

- **PHP is free to download and open source**

Anyone can start using PHP right now by downloading it from php.net. Millions of people are using PHP to create dynamic content and database-related applications that make for outstanding web systems. PHP is also open source, which means the original source code is made freely available and may be redistributed and modified.

- **Easy to learn & large community**

PHP is a simple language to learn step by step. This makes it easier for people to get engaged in exploring it. It also has such a huge community online that is constantly willing to help you whenever you're stuck (which actually happens quite a lot).

The graphic below shows a basic workflow of dynamic content being passed to and from the client using PHP:

Figure 1.1: PHP Dynamic Content Interaction

1.2 XAMPP Setup

XAMPP is a free and open source cross-platform web server solution developed by Apache Friends, consisting mainly of the Apache HTTP Server, MariaDB database, and interpreters for scripts written in the PHP and Perl programming languages. In order to make your PHP code execute locally, first install XAMPP.

- Download XAMPP
- Install the program (check the technologies you want during installation)
- Open XAMPP and click on "Start" on Apache and MySQL (when working with databases)

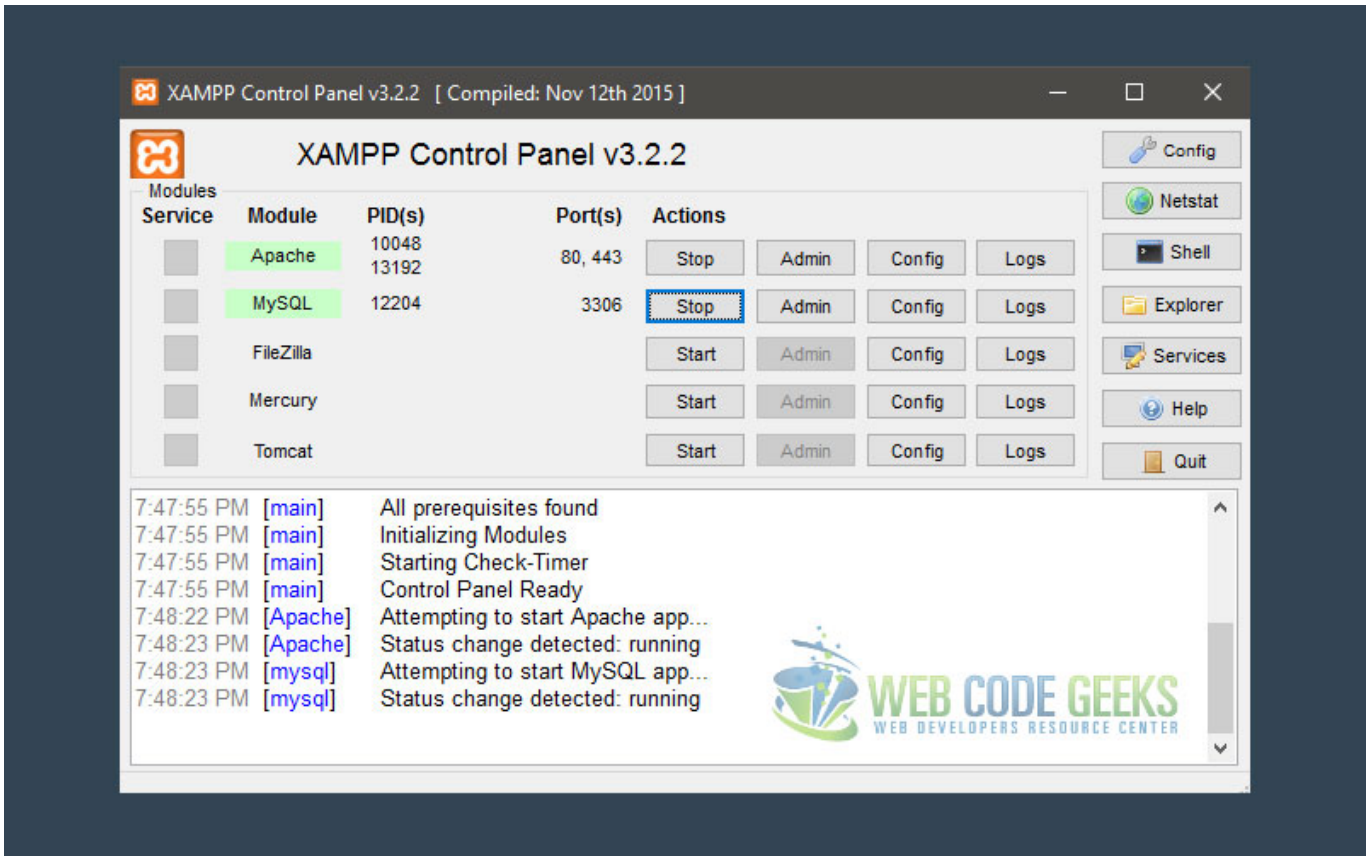

Figure 1.2: XAMPP window after a successful installation with Apache and MySQL enabled

- Place your web project inside the `htdocs` directory. In the common case, if you installed XAMPP directly inside the C: drive of your PC, the path to this folder would be: `C:xampphtdocs`

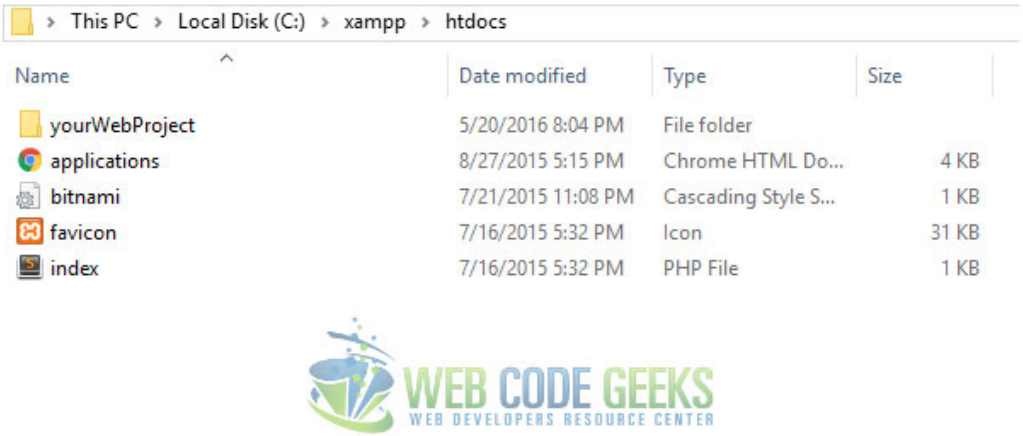

Figure 1.3: XAMPP Directory for Web Projects

To test the services are up and running you can just enter **localhost** in your address bar and expect the welcoming page.

1.3 PHP Language Basics

The aim of this section is to introduce the general syntax you should expect in PHP.

1.3.1 Escaping to PHP

There are four ways the PHP parser engine can differentiate PHP code in a webpage:

- **Canonical PHP Tags**

This is the most popular and effective PHP tag style and looks like this:

```
<?php...?>
```

- **Short-open Tags**

These are the shortest option, but they might need a bit of configuration, and you might either choose the `--enable-short-tags` configuration option when building PHP, or set the `short_open_tag` setting in your php.ini file.

```
<?...?>
```

- **ASP-like Tags**

In order to use ASP-like tags, you'll need to set the configuration option in the php.ini file:

```
<%...%>
```

- **HTML script Tags**

You can define a new script with an attribute language like so:

```
<script language="PHP">...</script>
```

1.3.2 Commenting PHP

Just like other languages, there are several ways to comment PHP code. Let's have a look at the most useful ones:

Use # to write single-line comments

```
<?
      # this is a comment in PHP, a single line comment
?>
```

Use // to also write single-line comments

```
<?
      // this is also a comment in PHP, a single line comment
?>
```

Use /* ...*/ to write multi-line comments

```
<?
      /*  this is a multi line comment
          Name: Web Code Geeks
          Type: Website
          Purpose: Web Development
      */
?>
```

1.3.3 Hello World

The very basic example of outputting a text in PHP would be:

```
<?
      print("Hello World");
      echo "Hello World";
      printf("Hello World");
?>
```

The result of the above statements would be the same: "Hello World". But why are there three different ways to output?

- **print** returns a value. It always returns 1.
- **echo** can take a comma delimited list of arguments to output.
- **printf** is a direct analog of C's `printf()`.

1.3.4 Variables in PHP

Any type of variable in PHP starts with a leading dollar sign ($) and is assigned a variable type using the = (equals) sign. The value of a variable is the value of its most recent assignment. In PHP, variables do not need to be declared before assignment. The main data types used to construct variables are:

- **Integers** - whole numbers like 23, 1254, 964 etc
- **Doubles** - floating-point numbers like 46.2, 733.21 etc
- **Booleans** - only two possible values, true or false
- **Strings** - set of characters, like *Web Code Geeks*
- **Arrays** - named and indexed collections of other values
- **Objects** - instances of predefined classes

The following snippet shows all of these data types declared as variables:

```
<?
      $intNum = 472;
      $doubleNum = 29.3;
      $boolean = true;
      $string = 'Web Code Geeks';
      $array = array("Pineapple", "Grapefruit", "Banana");

      // creating a class before declaring an object variable
      class person {
        function agePrint() {
            $age = 5;
            echo "This person is $age years old!";
        }
      }
      // creating a new object of type person
      $object = new person;
?>
```

1.3.5 Conditional Statements in PHP

Conditional statements are used to execute different code based on different conditions.

The If statement

The **if** statement executes a piece of code if a condition is true. The syntax is:

```
if (condition) {
    // code to be executed in case the condition is true
  }
```

A practical example would be:

```
<?php
  $age = 18;

  if ($age < 20) {
    echo "You are a teenager";
  }
  ?>
```

Because the condition is true, the result would be:

```
You are a teenager
```

The If...Else statement

The If...Else statement executed a piece of code if a condition is true and another piece of code if the condition is false. The syntax is:

```
if (condition) {
    // code to be executed in case the condition is true
  }
  else {
    // code to be executed in case the condition is false
  }
```

An example of an If...Else statement would be:

```
<?php
  $age = 25;

  if ($age < 20) {
    echo "You are a teenager";
  }
  else {
    echo "You are an adult";
  }
  ?>
```

Because the condition is false, the result in this case would be:

```
You are an adult
```

The If...Elseif...Else statement

This kind of statement is used to define what should be executed in the case when two or more conditions are present. The syntax of this case would be:

```
if (condition1) {
    // code to be executed in case condition1 is true
  }
```

```
  elseif (condition2) {
    // code to be executed in case condition2 is true
  }
  else {
    // code to be executed in case all conditions are false
  }
```

Again, a simple example to demonstrate this:

```
<?php
$age = 3;

if ($age < 10) {
  echo "You are a kid";
} elseif ($age < 20) {
  echo "You are a teenager";
} else {
  echo "You are an adult";
}
?>
```

The result, as you might expect, would be:

```
You are a kid
```

1.3.6 Loops in PHP

In PHP, just like any other programming language, loops are used to execute the same code block for a specified number of times. Except for the common loop types (for, while, do...while), PHP also support `foreach` loops, which is not only specific to PHP. Languages like Javascript and C# already use **foreach** loops. Let's have a closer look at how each of the loop types works.

The for loop

The `for` loop is used when the programmer knows in advance how many times the block of code should be executed. This is the most common type of loop encountered in almost every programming language.

```
for (initialization; condition; step){
       // executable code
}
```

An example where we use the for loop would be:

```
for ($i=0; $i < 5; $i++) {
      echo "This is loop number $i";
  }
```

The result of this code snippet would be:

```
This is loop number 0
    This is loop number 1
    This is loop number 2
    This is loop number 3
    This is loop number 4
```

The while loop

The `while` loop is used when we want to execute a block of code as long as a test expression continues to be true.

```
while (condition){
       // executable code
    }
```

An example where we use the while loop would be:

```
$i=0; // initialization
  while ($i < 5) {
      echo "This is loop number $i";
      $i++; // step
  }
```

The result of this code snippet would be just the same as before:

```
This is loop number 0
    This is loop number 1
    This is loop number 2
    This is loop number 3
    This is loop number 4
```

To have a clearer idea of the flow of these two loops, look at the graphic below:

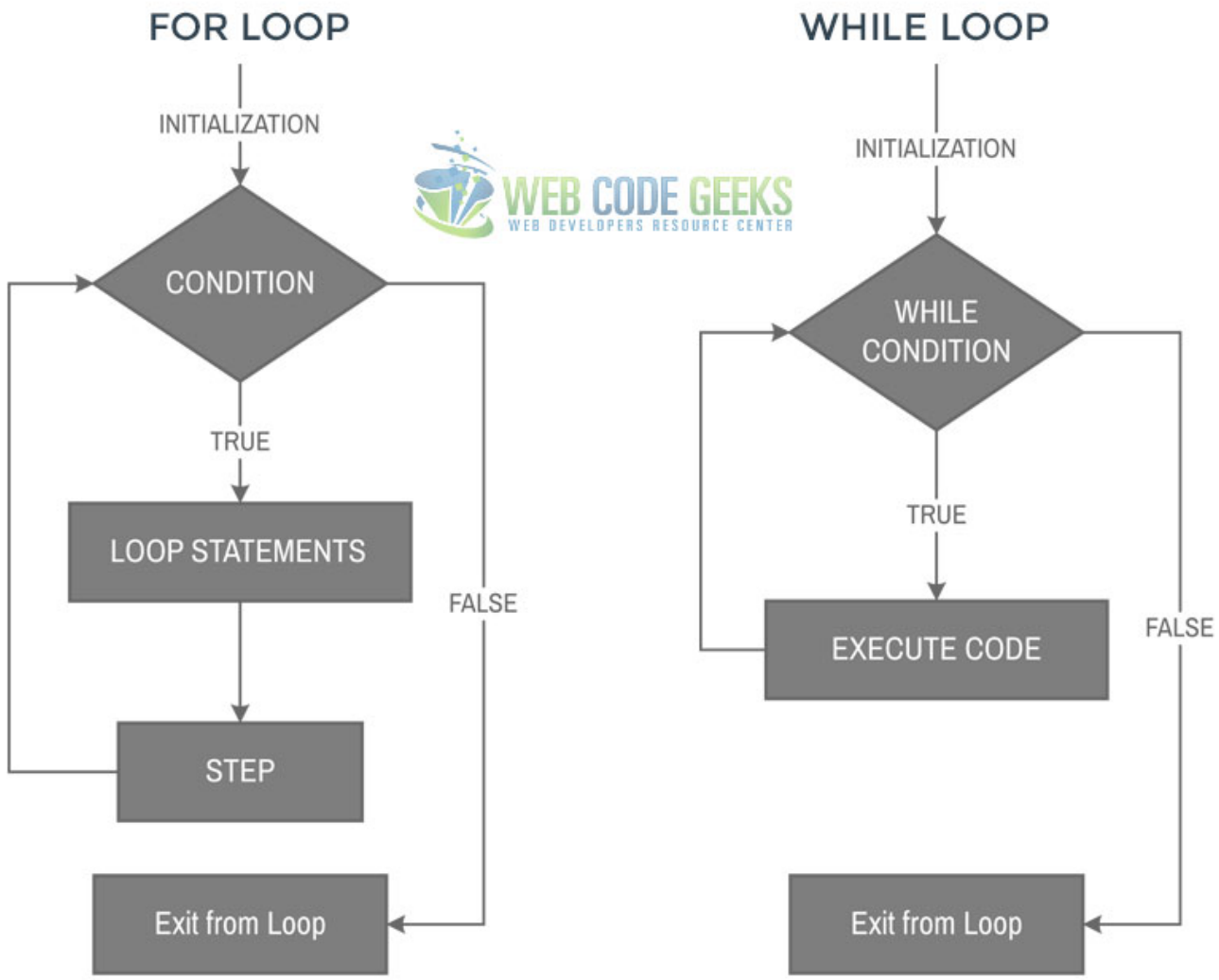

Figure 1.4: PHP for and while Loops

The do. . . while loop

The `do...while` loop is used when we want to execute a block of code at least once and then as long as a test expression is true.

```
do {
       // executable code
    }
    while (condition);
```

An example where we use the do...while loop would be:

```
$i = 0; // initialization
  do {
      $i++; // step
      echo "This is loop number $i";
  }
  while ($i < 5);  // condition
```

This time the first loop number would be 1, because the first `echo` was executed only after variable incrementation:

```
This is loop number 1
    This is loop number 2
    This is loop number 3
    This is loop number 4
    This is loop number 5
```

The foreach loop

The `foreach` loop is used to loop through arrays, using a logic where for each pass, the array element is considered a `value` and the array pointer is advanced by one, so that the next element can be processed.

```
foreach (array as value) {
       // executable code
    }
```

An example where we use the foreach loop would be:

```
$var = array('a','b','c','d','e');  // array declaration

  foreach ($var as $key) {
      echo "Element is $key";
  }
```

This time the first loop number would be 1, because the first `echo` was executed only after variable incrementation:

```
Element is a
    Element is b
    Element is c
    Element is d
    Element is e
```

1.4 PHP Arrays

Arrays are used to store multiple values in a single variable. A simple example of an array in PHP would be:

```
<?php
$languages = array("JS", "PHP", "ASP", "Java");
?>
```

Array elements are accessed like this: `$arrayName[positionIndex]`. For the above example we could access "PHP" this way: `$languages[1]`. Position index is 1 because in programming languages the first element is always element 0. So, PHP would be 1 in this case.

There are three types of arrays in PHP:

Indexed Arrays

We can create these kind of arrays in two ways shown below:

```
<?php
  $names = array("Fabio", "Klevi", "John");
  ?>
```

```
<?php
  // this is a rather manual way of doing it
  $names[0] = "Fabio";
  $names[1] = "Klevi";
  $names[2] = "John";
  ?>
```

An example where we print values from the array is:

```
<?php
  $names = array("Fabio", "Klevi", "John");
  echo "My friends are " . $names[0] . ", " . $names[1] . " and " . $names[2];
  ?>
```

```
// RESULT
 My friends are Fabio, Klevi and John
```

Looping through an indexed array is done like so:

```
<?php
    $names = array("Fabio", "Klevi", "John");
    $arrayLength = count($names);

    for($i = 0; $i < $arrayLength; $i++) {
        echo $names[$i];
        echo "";
    }
  ?>
```

This would just print the values of the array.

Associative Arrays

Associative arrays are arrays which use named keys that you assign. Again, there are two ways we can create them:

```
<?php
  $namesAge = array("Fabio"=>"20", "Klevi"=>"16", "John"=>"43");
  ?>
```

```
<?php
  // this is a rather manual way of doing it
  $namesAge['Fabio'] = "20";
  $namesAge['Klevi'] = "18";
  $namesAge['John'] = "43";
  ?>
```

An example where we print values from the array is:

```
<?php
  $namesAge = array("Fabio"=>"20", "Klevi"=>"16", "John"=>"43");
  echo "Fabio's age is " . $namesAge['Fabio'] . " years old.";
  ?>
```

```
// RESULT
Fabio's age is 20 years old.
```

Looping through an associative array is done like so:

```
<?php
    $namesAge = array("Fabio"=>"20", "Klevi"=>"16", "John"=>"43");

    foreach($namesAge as $i => $value) {
        echo "Key = " . $i . ", Value = " . $value;
        echo "";
    }
  ?>
```

```
Key = Fabio, Value = 20
  Key = Klevi, Value = 16
  Key = John, Value = 43
```

Multidimensional Arrays

This is a rather advanced PHP stuff, but for the sake of this tutorial, just understand what a multidimensional array is. Basically, it is an arrays the elements of which are other arrays. For example, a three-dimensional array is an array of arrays of arrays. An example of this kind of array would be:

```
<?php
    $socialNetowrks = array (
    array("Facebook", "feb", 21),
    array("Twitter","dec", 2),
    array("Instagram","aug", 15));
    ?>
```

1.5 PHP Functions

Functions are a type of procedure or routine that gets executed whenever some other code block calls it. PHP has over 1000 built-in functions. These functions help developers avoid redundant work and focus more on the logic rather than routine things. Apart from its own functions, PHP also let's you create your own functions. The basic syntax of a function that we create is:

```
<?php
function functionName($argument1, $argument2...) {
  // code to be executed
}

functionName($argument1, $argument2...); // function call
?>
```

Every function needs a name, optionally has one or more arguments and most importantly, defines some kind of procedure to be followed within the body, that is, code to be executed. Let's see some basic functions:

```
<?php
function showGreeting() { // function definition
  echo "Hello Chloe!";    // what this function does
}
```

```
showGreeting(); // function call
?>
```

This function would just print the message we wrote:

```
Hello Chloe!
```

Another example would be a function with arguments:

```
<?php
function greetPerson($name) { // function definition with arguments
  echo "Hi there, ".$name;    // what this function does
}

greetPerson("Fabio"); // function call
greetPerson("Michael");
?>
```

This time we'd have:

```
Hi there, Fabio
Hi there, Michael
```

More than one argument can be used whenever needed:

```
<?php
function personProfile($name, $city, $job) { // function definition with arguments
  echo "This person is ".$name." from ".$city.".";
  echo "";
  echo "His/Her job is ".$job.".";
}

personProfile("Fabio", "Tirana", "Web Dev");
echo "";
personProfile("Michael", "Athens", "Graphic Designer");
echo "";
personProfile("Xena", "London", "Tailor");
?>
```

The result would include the printed message together with the arguments:

```
This person is Fabio from Tirana.
His/Her job is Web Dev.
This person is Michael from Athens.
His/Her job is Graphic Designer.
This person is Xena from London.
His/Her job is Tailor.
```

In PHP, just like in many other languages, we can tell functions to return a value upon executing the code. Such example would be:

```
<?php
function difference($a, $b) { // function definition with arguments
  $c = $a - $b;
  return $c;
}

echo "The difference of the given numbers is: ".difference(8, 3);
?>
```

The result would be:

```
The difference of the given numbers is: 5
```

PHP also provides quite useful functions for developers to use. One of them is the `mail()` function. Have a detailed look of how you can use this function to send e-mails **in this article**

1.6 Connecting to a Database

There are four ways you can generally consider when you want to connect to a previously created database. Below, we'll explain how you can use each of them beginning with the easiest one.

1.6.1 Connecting to MySQL Databases

The syntax for connecting to a MySQL database would be:

```
<?php
$username = "your_name";
$password = "your_password";
$hostname = "localhost";

//connection to the database
$dbConnect = mysql_connect($hostname, $username, $password)
or die("Unable to connect to MySQL");
echo "Connected to MySQL";

//select a specific database
$dbSelect = mysql_select_db("dbName", $dbConnect)
or die("Could not select dbName");
?>
```

Considering your entered information is correct, you'd be successfully connected to the right database and ready to start writing and test your queries. Else, the respective error message would appear as defined by the `die` function. However, do keep in mind that the mysql extension is deprecated and will be removed in the future, so the next methods can be used for databases.

1.6.2 Connecting to MySQLi Databases (Procedurial)

The MySQLi stands for MySQL improved. The syntax for connecting to a database using MySQLi extension is:

```
<?php
$username = "your_name";
$password = "your_password";
$hostname = "localhost";

//connection to the database
$dbConnect = mysqli_connect($hostname, $username, $password)

// another way of checking if the connection was successful
if (!$dbConnect) {
  die ("Connection failed: " . mysqli_connect_error());
}
echo "Connected successfully";

//select a specific database
mysqli_select_db($dbConnect, "dbName")
?>
```

This is a good way to start, because it is easy to understand and gets the job done. However, object oriented logic that we'll see below is what everyone should be getting into because of the other components of programming being used in this paradigm and also because it is kind of more structured way of doing things.

1.6.3 Connecting to MySQLi databases (Object-Oriented)

Although the functionality is basically the same, this is another way, the object-oriented way of connecting to a database using the MySQLi extension.

```
$username = "your_name";
$password = "your_password";
$hostname = "localhost";

// create connection
$dbConnect = new mysqli($hostname, $username, $password);

// check connection
if ($dbConnect->connect_error) {
    die("Connection failed: " . $dbConnect->connect_error);
}
echo "Connected successfully";

// select a specific database
$mysqli->select_db("dbName");
```

Even in this case, you can check to see if the database was successfully selected, but it is a matter of choice. Object-Oriented MySQLi is not a different MySQLi as far as code functionality is concerned, but rather a different way/logic of writing it.

1.6.4 Connecting to PDO Databases

PDO stands for **PHP Data Objects** and is a consistent way to access databases, which promises much easier portable code. PDO is more like a data access layer which uses a unified API rather than an abstraction layer. The syntax for connecting to a database using PDO is:

```
$username = "your_name";
$password = "your_password";
$hostname = "localhost";

// try to create connection
try {
    $dbConnect = new PDO("mysql:host=$hostname;dbname=myDB", $username, $password);
    // set the PDO error mode to exception
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    echo "Connected successfully";
}
// show an error if the connection was unsuccessful
catch(PDOException $e) {
    echo "Connection failed: " . $e->getMessage();
}
```

PDO is widely used today for a bunch of advantages it offers. Among them, we can mention that PDO allows for prepared statements and rollback functionality which makes it really consistent, it throws catchable exceptions which means better error handling and uses blind parameters in statements which increases security.

1.7 PHP Form Handling

In HTML, forms are used to collect user input/information. But, as far as HTML goes, it just provides the graphical interface and the ability to write on the input fields. However, the aim we collect user information is because we need to process this information. That usually means saving the inputs to a database so that they can be accessed later for several purposes. In this section, we will not send information to the database, as that requires an active database that is already set up and includes knowledge form SQL language, but we can retrieve information that the user has given us. That being said, it is important to

have a way to get information, becuase what you use it for and where you want to show/save it depends. Let's have a look at the HTML form below:

```
<form method="POST" action="wcg.php">
    <input type="text" name="name" placeholder="Name">
    <input type="email" name="email" placeholder="E-Mail">
    <input type="number" name="age" placeholder="Age">
    <input type="radio" name="gender" value="Male"> Male
    <input type="radio" name="gender" value="Female"> Female
    <input type="submit" name="submit" value="Submit">
</form>
```

This form is just a regular one, and includes inputs for name, e-mail, age and gender. This information will be subject of a **print** when the Submit button is clicked. That just proves that we got the information from the user. For now, let's see what we got and fill the form:

Figure 1.5: HTML form we just created

Next, we check each input to make sure the user has written/chosen something, and the input is not empty. We do this using two well-known functions in PHP, the `isset()` and `empty()`. After making sure we have this right, we can also validate fields. In this case, only the name input may need some sort of validation to make sure one simply doesn't write numbers in there, while other inputs are more or less validated from HTML5, in the case of email and age.

```
<?php
if (isset($_POST["name"]) && !empty($_POST["name"])) {
    $name = $_POST["name"];
    if (!preg_match("/^[a-zA-Z ]*$/",$name))
    echo "Name: Only letters and whitespace allowed";
    else
    echo "Name: ".$_POST["name"]."";
}
if (isset($_POST["email"]) && !empty($_POST["email"])) {
    echo "E-Mail: ".$_POST["email"]."";
}
if (isset($_POST["age"]) &&!empty($_POST["age"])) {
```

```
    echo "Age: ".$_POST["age"]."";
}
if (isset($_POST["gender"]) && !empty($_POST["gender"])) {
    echo "Gender: ".$_POST["gender"];
}
?>
```

Validation is a must for all fields when considering real web projects. We recommend having a closer look to our article on **PHP Form Validation Example**

If the fields have been set, information about each field will be printed. After clicking the "Submit" button, the result we'd have is:

```
Name: Alex Brown
E-Mail: alex.brown@gmail.com
Age: 28
Gender: Male
```

As you may have noticed by now, we used this `$_POST["name"]` to get information the user posted.

What about implementing a login form in PHP? Have a look at our **Login Form in PHP Example**.

1.8 PHP Include & Require Statements

PHP code can get cluttered, which means that if you want to later change something, it becomes a hard task to do. Include and require statements are two almost identical statements that help in an important aspect of coding, the organization of code, and making it more readable and flexible. The include/require statement copies all text, code or any other markup from one existing file to the file using the statement. In a simple viewpoint, do consider these statements like this:

Figure 1.6: PHP Include Statement

The include and require statements are the same, except upon failure of code execution where:

- **require** will produce a fatal error (E_COMPILE_ERROR) and stop the script from executing
- **include** will only produce a warning (E_WARNING) and the script will continue

The syntax of these two statements is as follows:

```
<?php
include 'file.php'; // in case of include
require 'file.php'; // in case of require
?>
```

Let's now see a real-world example where we use a header and footer using include and require into our main file:

header.php

```
<header>
<nav>
    <ul>
        <li><a href="#">Home</a></li>
        <li><a href="#">Profile</a></li>
        <li><a href="#">About</a></li>
        <li><a href="#">Contact</a></li>
```

```
    </ul>
</nav>
</header>
// styling to make the menu look right
<style type="text/css">
    ul, li {
        list-style-type: none;
        display: inline-block;
        margin-right: 1em;
        padding: 0;
    }
</style>
```

footer.php

```
<footer>All Rights Reserved. Do not reproduce this page.</footer>
```

In our main file, we'll use `require` for the header.php and `include` for the footer.php file:

index.php

```
<?php require 'header.php'; ?>

<body>
    <h2>Main Content Goes Here</h2>
</body>

<?php include 'footer.php' ?>
```

The result, as you might have an idea by now, would be the whole code being shown as one:

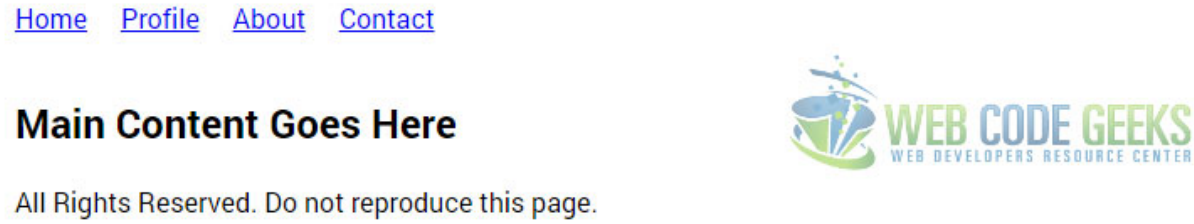

Figure 1.7: The result of using include and require in our main file

1.9 Object Oriented Concepts

Object oriented programming is a programming language model in the center of which are objects. Let's see some of the core concepts of classes and objects before we see actual examples of them:

- **Class**

A class is a predefined (by a programmer) data type, part of which are local data like variables, functions etc.

- **Object**

An object is an instance of a class. Objects can only be created after a class has been define. A programmer can create several objects.

- **Constructor**

A constructor refers to the concept where all data and member functions are encapsulated to form an object. A destructor, on the other hand, is called automatically whenever an object is deleted or goes out of scope.

1.9.1 PHP Classes

Let's now define a new clas in PHP.

```
<?php
  class myClass {          // use 'class' followed by the name of the class
    var $var1;             // declared an undefined variable
    var $var2 = 5;         // declared a number defined variable
    var $var3 = "string"; // declared a string defined variable

    function myFunction ($argument1, $argument2) {  // function definition
      // function code here
    }
  }
  ?>
```

But that is just how the syntax looks like. Below, we give an example of a real-world class, for example class `Vehicle`:

```
<?php
  class Vehicle {
    var $brand;       // just a declared undefined variable
    var $speed = 80;  // a declared and defined variable

    function setSpeed($speedValue) {  // a function to change speed
      $this->speed = $speedValue;     // this will replace speed with our value
    }

    function setBrand($brandName) {   // a function to change brand
        $this->brand = $brandName;    // this will set a brand name
    }

    function printDetails(){          // a function to print details
        echo "Vehicle brand is: ".$this->brand;
        echo "";
        echo "Vehicle speed is: ".$this->speed;
    }
  }

  $myCar = new Vehicle;       // an instance of our Vehicle class (an object)
  $myCar->setBrand("Audi");   // calling the function setBrand to define a brand
  $myCar->setSpeed(120);      // calling the function setSpeed to change speed
  $myCar->printDetails();     // calling the printDetails function to see details
  ?>
```

The result of this code would be:

```
Vehicle brand is: Audi
Vehicle speed is: 120
```

As you can see, the speed is changed to the latest value set by us.

1.9.2 PHP Constructor Function

PHP provides a special function called `__construct()` to define a constructor, which can take as many arguments as we want. Constructors are called automatically whenever an object is created. Let's see how we adopt this into our previously created class. The following code snipper is found inside the class:

```
<?php
  class Vehicle {
  function __construct ($brandName, $speedValue) {
    $this->brand = $brandName;  // initialize brand
    $this->speed = $speedValue; // initialize speed
    }
      function printDetails(){
          echo "Vehicle brand is: ".$this->brand;
          echo "
";
          echo "Vehicle speed is: ".$this->speed;
          echo "

";
      }
  }

    $car1 = new Vehicle("Toyota", 130);
    $car2 = new Vehicle ("Ferrari", 450);

    $car1->printDetails();
    $car2->printDetails();
  ?>
```

The result now is:

```
Vehicle brand is: Toyota
Vehicle speed is: 130

Vehicle brand is: Ferrari
Vehicle speed is: 450
```

PHP contains such object oriented rules just like other languages like Java, and as you go on learning it in further details, you'll see that you can do more with those variables we declare inside a class, you can set scope to them so that they can only be accessed within a class, a function etc. Also, other concepts like methods overriding is normal when using PHP, as declaring the same method for the second time but with different arguments or code logic, will make the program execute only the latest method it finds in the file.

1.10 Conclusion

PHP is the fundamental back-end development technology regarding usage and learning curve. It combines such features that make for interactive web pages that people love to use everyday. In this article, we detailed the basic concepts to give you a clear idea of how it can be used in different aspects and information that you might as well relate to other programming languages like loops and conditionals, as most languages recognize these concepts and make use of them.

In the early days, everyone used PHP in a rather procedurial style, as a way to get things done when you don't know a lot, but as you learn more and more, you'll love working with object-oriented PHP which will make your programming life way much easier and sustainable in big projects. It is important to understand that there are many ways pages could be made dynamic in PHP, and different developers might as well have a totally distinct code that does the same task, that is some kind of flexibility that you get to decide how you are going to implement certain functions (randomly called procedures in PHP). So, just give it a try, you have nothing to lose.

1.11 Download the source code

Download You can download the full source code of this example here: **PHP-Tutorial-Samples**

Chapter 2

Upload Script Example

In this example, we are going to see how to make file uploads with PHP, that is, how to upload a file from the client side, to the server side.

The process itself is pretty simple, but several checks have to be done to ensure that the file upload is made safely, i.e., to neutralise any hypothetical malicious misuse; and to be able to detect every possible error to act accordingly, so, this is something that must be done carefully.

For this tutorial, we will use:

- Ubuntu (14.04) as Operating System.
- Apache HTTP server (2.4.7).
- PHP (5.5.9).

2.1 Preparing the environment

2.1.1 Installation

Below, are shown the commands to install Apache and PHP:

```
sudo apt-get update
sudo apt-get install apache2 php5 libapache2-mod-php5
sudo service apache2 restart
```

2.1.2 PHP configuration

Is necessary to configure PHP to allow file uploads. With your favourite editor, open the `php.ini` file:

```
sudo vi /etc/php5/apache2/php.ini
```

And check that the `file_uploads` directive value is set to `On`:

```
file_uploads = On
```

If you want, you can change the `upload_max_filesize` directive, to change the maximum size allowed for upload.

Don't forget to restart Apache after doing any change.

Note: if you want to use Windows, installing XAMPP is the fastest and easiest way to install a complete web server that meets the prerequisites.

2.2 Upload form

First, we need to create a form, where the file will be attached. It should be similar to this:

upload_form.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Upload form</title>
</head>
<body>
    <form action="upload.php" method="post" enctype="multipart/form-data">

            <label for="file">Select a file to upload:</label>
            <input type="file" name="file" id="file">


            <input type="submit" value="Upload file!">

    </form>
</body>
</html>
```

There are several things to take into account:

- Form `action` **must** point to the PHP file that will perform the upload (this is quite obvious).
- Submission method **must** be `POST`.
- Form `enctype` attribute **must** be specified, with `multipart/form-data` value. This is not to make the form encode the characters.

2.2.1 Accepting only certain type of files

It is possible to restrict the uploading files to only expected file types. This can be made in several ways:

- By file extension. For example, if we are expecting a document, we can restrict the extensions to `.pdf` and `.doc`.
- By file type. We can directly say that we are expecting an image, an audio or/and a video.
- By media type, specifying the MIME type. An example could be `text/plain`, if we are expecting plain files.

All of these can be combinated, so we could say, for example, that we are expecting a document file, and it can be accepted in pdf, doc, docx and odt and format.

Applying this last example to the form above (line 11), the result would be the following:

upload_form.html

```
<input type="file" name="file" id="file" accept=".pdf,.doc,.docx,.odt">
```

As you can see, the conditions are sepparated by comma.

Note: keep always in mind that **client-side validation is never enough**. Every validation we do in the client-side, it should be done also in the server - remember that here is the person in the client-side who has the control, not you!

2.3 PHP upload script

Once the form is submitted, is time for the server to handle the upload. The following script will do the job:

upload.php

```
<?php

define('UPLOAD_DIRECTORY', '/var/php_uploaded_files/');

$uploadedTempFile = $_FILES['file']['tmp_name'];
$filename = $_FILES['file']['name'];
$destFile = UPLOAD_DIRECTORY . $filename;

move_uploaded_file($uploadedTempFile, $destFile);
```

Easy, isn't it? We just define the target directory where the file will be stored, we get the file PHP uploaded temporarily, original file's name, and we move that file to the defined target directory, with its name.

But, as we said in the introduction, in this task we must focus the efforts into the security. This script does not check:

- File size. We should set a maximum allowed size (what if an user is trying to upload a file bigger than server's available space?)
- File type. If we are expecting, for example, a document, receiving an executable file would be quite suspicious, wouldn't it?
- That the file was properly uploaded to the server, and moved to the target directory. We can't take anything for granted. With that script, we won't notice if an error occurs in the upload.

2.3.1 A secure file upload script

Let's update our script to make a safe file upload, supposing that we expect only document-type files, as we defined 2.1 section, of pdf, doc, docx and odt format:

upload.php

```
<?php

define('UPLOAD_DIRECTORY', '/var/php_uploaded_files/');
define('MAXSIZE', 5242880); // 5MB in bytes.

// Before PHP 5.6, we can't define arrays as constants.
$ALLOWED_EXTENSIONS = array('pdf', 'doc', 'docx', 'odt');
$ALLOWED_MIMES = array('application/pdf', // For .pdf files.
    'application/msword', // For .doc files.
    'application/vnd.openxmlformats-officedocument.wordprocessingml.document', // For .docx ↩
        files.
    'application/vnd.oasis.opendocument.text', // For .odt files.
);

/**
 * Checks if given file's extension and MIME are defined as allowed, which are defined in
 * array $ALLOWED_EXTENSIONS and $ALLOWED_MIMES, respectively.
 *
 * @param $uploadedTempFile The file that is has been uploaded already, from where the MIME
 *     will be read.
 * @param $destFilePath The path that the dest file will have, from where the extension ↩
    will
 *     be read.
 * @return True if file's extension and MIME are allowed; false if at least one of them is ↩
    not.
 */
```

```
function validFileType($uploadedTempFile, $destFilePath) {
    global $ALLOWED_EXTENSIONS, $ALLOWED_MIMES;

    $fileExtension = pathinfo($destFilePath, PATHINFO_EXTENSION);
    $fileMime = mime_content_type($uploadedTempFile);

    $validFileExtension = in_array($fileExtension, $ALLOWED_EXTENSIONS);
    $validFileMime = in_array($fileMime, $ALLOWED_MIMES);

    $validFileType = $validFileExtension && $validFileMime;

    return $validFileType;
}

/**
 * Handles the file upload, first, checking if the file we are going to deal with is actually an
 * uploaded file; second, if file's size is smaller than specified; and third, if the file is
 * a valid file (extension and MIME).
 *
 * @return Response with string of the result; if it has been successful or not.
 */
function handleUpload() {
    $uploadedTempFile = $_FILES['file']['tmp_name'];
    $filename = basename($_FILES['file']['name']);
    $destFile = UPLOAD_DIRECTORY . $filename;

    $isUploadedFile = is_uploaded_file($uploadedTempFile);
    $validSize = $_FILES['file']['size'] <= MAXSIZE && $_FILES['file']['size'] >= 0;

    if ($isUploadedFile && $validSize && validFileType($uploadedTempFile, $destFile)) {
        $success = move_uploaded_file($uploadedTempFile, $destFile);

        if ($success) {
            $response = 'The file was uploaded successfully!';
        } else {
            $response = 'An unexpected error occurred; the file could not be uploaded.';
        }
    } else {
        $response = 'Error: the file you tried to upload is not a valid file. Check file type and size.';
    }

    return $response;
}

// Flow starts here.

$validFormSubmission = !empty($_FILES);

if ($validFormSubmission) {
    $error = $_FILES['file']['error'];

    switch($error) {
        case UPLOAD_ERR_OK:
            $response = handleUpload();
            break;

        case UPLOAD_ERR_INI_SIZE:
            $response = 'Error: file size is bigger than allowed.';
            break;
```

```
        case UPLOAD_ERR_PARTIAL:
            $response = 'Error: the file was only partially uploaded.';
            break;

        case UPLOAD_ERR_NO_FILE:
            $response = 'Error: no file could have been uploaded.';
            break;

        case UPLOAD_ERR_NO_TMP_DIR:
            $response = 'Error: no temp directory! Contact the administrator.';
            break;

        case UPLOAD_ERR_CANT_WRITE:
            $response = 'Error: it was not possible to write in the disk. Contact the ↩
                administrator.';
            break;

        case UPLOAD_ERR_EXTENSION:
            $response = 'Error: a PHP extension stopped the upload. Contact the ↩
                administrator.';
            break;

        default:
            $response = 'An unexpected error occurred; the file could not be uploaded.';
            break;
    }
} else {
    $response = 'Error: the form was not submitted correctly - did you try to access the ↩
        action url directly?';
}

echo $response;
```

Now, let's see the key instructions added to this script:

- We first check the `$_FILES` superglobal, in line 70. If it's empty, it would be because the access to this script has not been done through a form submission, so, we shouldn't perform any action.
- We get the `error` property of the superglobal, in line 73. Here is saved the state of the file submission, indicating if it has been done correctly, or if any error occured. The state is later evaluated in the `switch`, where we perform different actions depending on the state.
- Getting the basename of the file, in line 47. This PHP built-in gives us the file name. Even if it is defined in `$_FILES['file']['name']`, it is always recommended to get file names using this function.
- Checking if the file is actually an uploaded file, in line 50. This function checks that the given file is really a file uploaded through HTTP POST. This check is necessary to avoid malicious actions that may want to have access server files.
- Checking the size of the file, in line 51. Before proceding, we ensure that the uploading file doesn't exceed the maximum size set. Because the limit does not always have to be the one set in `php.ini`.
- Checking the file type, in line 53. This is probably one of the most interesting checks. Here, we ensure that the receiving file is actually a document, in two steps: getting the file extension (line 27), and the file MIME (line 28). It is not enough checking only one of these. Consider the following scenario: we are checking only the file extension, which is expected to be one of those defined in the array. If an evil user wants to upload an executable file, it only has to change its extension to one of those. But the content type is not modified with the extension. So, we have to check both to ensure that the files are real documents.
- Finally, we check that the file has been moved correctly to the target directory, with its original name (line 54). The difference here compared with the previous script, is that we retrieve the boolean value returned by the `move_uploaded_files` function, because it may fail, for any reason. As we said before, we can't take anything for granted.

2.3.2 Considerations

You may noticed that the temp files created by PHP (`$_FILES['file']['tmp_name']`) are not being deleted. This is because we don't have to care about it; when the script finishes it execution, PHP will delete them for us.

About the checking of allowed files, in some cases, when the upload is not oriented only to a very specific file type, it may be more interesting to have a blacklist where not allowed file types are defined, instead of having a whitelist, like in the script above. Generally, we won't expect to receive files that can be harmful, like executable files, or even scripts.

2.4 Summary

With this script, files will be safely uploaded to our server, ensuring that the received file is of the type/types we are expecting (remember that the **validation must reside always in the server-side**), and also that doesn't exceed the established size. And, if something goes wrong, we will be able to identify the exact error, that will allow us to take the most appropiate decision for each case.

2.5 Download the source code

This was an example of file uploading with PHP.

Download You can download the full source code of this example here: **PHPUploadScript**

Chapter 3

Mail Function Example

In this example, we will see how we can send emails with PHP. After seeing this example, you will be able to send emails programmatically, which is especially useful in scenarios like the following:

- Registration confirmation.
- Password recovery.
- Notification delivery.

And any other scenario where a service needs to contact a user.

PHP has a built-in function called `mail` that will do all the job but, first, we have to make some configurations in order to make this function work.

For this tutorial, we will use:

- Ubuntu (14.04) as Operating System.
- Apache HTTP server (2.4.7).
- PHP (5.5.9).
- sSMTP (2.64), a lightweight MTA (Mail Transfer Agent).

3.1 Preparing the environment

3.1.1 Installation

Below, the commands to install Apache and PHP are shown:

```
sudo apt-get update
sudo apt-get install apache2 php5 libapache2-mod-php5
sudo service apache2 restart
```

Now, we install sSMTP:

```
sudo apt-get install ssmtp
```

With this MTA, now, we will be able to send mails from our machine, to a SMTP server. In other words, we can make a SMTP server (like Gmail's, Outlook's, etc.) deliver the mails we want to send.

3.1.2 sSMTP configuration

The sSMTP configuration consists of two things:

- Defining the SMTP service we are going to use.
- Provide credentials for that service.

In this example, we will use Google's Gmail SMTP server., and supposing that we have an account named mygmailaccount@gmail.com, with mygmailpassword as password.

So, taking that into account, the `/etc/ssmtp/ssmtp.conf` file will remain as:

ssmtp.conf

```
UseSTARTTLS=YES
mailhub=smtp.gmail.com:587
AuthUser=mygmailaccount@gmail.com
AuthPass=mygmailpassword
```

This is what we are setting:

- UseSTARTTLS: necessary for using Gmail's SSL/TLS sever.
- mailhub: the smtp server itself, in port 587, for SSL/TLS.
- AuthUser and AuthPass: credentials for loging in Gmail.

Note: take care of who has access to this file - anyone having access to the file, can access the mail account set.

3.1.3 PHP configuration

The last step, is to make PHP now how to use sSMTP. For that, we have to specify the path to sSMTP executable, in PHP config file, `/etc/php5/apache2/php.ini` file:

php.ini

```
sendmail_path = /usr/sbin/sendmail
```

Don't forget to restart Apache after making any change in PHP configuration:

```
sudo service apache2 restart
```

3.2 PHP mail sending script

The PHP code is all about calling a function, named mail. Here an example that sends a mail to someone, with a subject and the message itself:

send_mail_example.php

```
<?php

$to = 'someone1@mail.com, someone2@mail.com';
$subject = 'Mail from PHP';
$message = "Hi,\r\nThis mail has been sent using PHP!";

$success = mail($to, $subject, $message);

if ($success) {
```

```
    $response = 'Message has been sent successfully.';
} else {
    $response = 'The message could not be sent.';
}

echo $response;
```

There are a few things that have to be taken into account:

- It is possible to set multiple mails to send the mail to, separated by commas, as in the example above.
- `mail` function does not perform any encoding. This can be a problem, that we will see below how to deal with.
- The lines must be separated with Windows line ending format, i.e., the CRLF: `'rn'`.
- Unfortunately, `mail` function only returns a boolean value so, if something goes wrong, we cannot have any details at runtime.

Mail function has two more optional parameters: for setting additional headers, and for setting additional parameters. The first one is to add additional headers, as we would do, for example, in a HTML page. The last one is, actually, a MTA command line parameter, used to pass additional flags and commands.

Let's see how can we use these additional options.

3.2.1 Setting additional headers

Consider the following scenario: we have a system that sends greetings to the users registered within, each one in the language set by the user. If we have, for example, a Chinese user, the message body would be:

send_mail_example.php

```
$message = 'ä½~å$\yen$½'; // 'Hello' in Chinese.
```

With the example we saw above, the result would be an ugly string of characters that don't match with the expected Chinese characters. For a scenario like this, is not enough to rely the encoding only in the language setting. The solution would be to encode the message properly, encoding it to UTF-8, and, for that, we have to set the header manually.

For that, we need to perform two additional steps:

send_mail_example.php

```
$headers = 'Content-Type: text/plain; charset=UTF-8';

$success = mail($to, $subject, $message, $headers);
```

Quite simple: define the headers, and pass them to the function. If we try now the script, we will receive a mail with those nice Chinese characters.

It is also usual to use this additional headers, for example, to add carbon copies (cc) and blind carbon copies (bcc).

Note: Multiple headers must be separated as same as new lines in the message, with a CRLF.

3.2.2 Setting additional parameters

As told before, when we are setting additional parameters, we are actually passing instructions that will be executed by the server, te binary we set to `sendmail_path`, in `php.ini`.

For example, we could set the verbose mode, to see how is going the whole process, which can be useful for troubleshooting.

For this, we just declare another variable, for the parameters, and we pass it as last parameter to the function:

send_mail_example.php

```
$parameters = '-v';

$success = mail($to, $subject, $message, $headers, $parameters);
```

If we execute now the script, we will see the whole process: from the initial handshake with the server, to the connection close.

In this case, to pass several parameters, they must be separated by a blank space.

You can see the whole list of available options in sSMTP manual:

```
man ssmtp
```

3.3 Troubleshooting

If you have not been able to make this example work, in this section we are going to see which can be the possible causes, and finding solutions for them.

3.3.1 Set verbose mode

Before doing anything else, the first recommended step would be to set the verbose mode, to see in detail what's happening. You can see how to do it in section 2.2.

3.3.2 Login is rejected by SMTP server

It may occur that Gmail considered the login attempt through sSMTP as insecure or suspicious, because of trying to login using an external service or app.

To confirm that this is the problem, login into the account you set in sSMTP configuration file, and you should have received and email from Google with a subject similar to: `Suspicious sign in prevented.` To disable this security measure, in Gmail, go to My Account/Sign-in & security, and in Connected apps & sites section, enable the "Allow less secure apps" toggle to "on".

3.3.3 Firewall is filtering outgoing traffic

Maybe, a firewall is filtering the connection. Normally, by default, firewalls do not filter outgoing traffic, but it can be a possibility. Check that you don't have any firewall filtering outgoing traffic that would affect to this example (remember that we configured the MTA to use the port 587).

3.3.4 Check PHP error log

If it continues not working, check the PHP error log, to see which is the error thrown by PHP, to get more hints of which can be the error. By default, the error log is located in `/var/log/apache2/error.log`.

3.4 Alternatives to sSMTP

For this example, we have chosen sSMTP as it is the easiest and fastest way to deliver mails, only needing the simple configuration we saw in section 1.2, and also because it is very light; is not using daemons, and its impact in CPU and memory is negligible.

But, if you are looking for a more advanced, full-featured MTA, **Postfix** and **Exim** are probably the most used ones.

3.5 Summary

In this example, we have seen how to send mails with PHP, which is pretty simple - only a function call is needed. But we have also seen that, that alone is not enough, and that a MTA is needed, to connect with the SMTP server we are using. We have also analysed the most common errors that we can face in this task, and how to solve them, apart from the problem that can suppose the sending of not properly encoded mails.

3.6 Download the source code

This was an example of mail sending with PHP.

Download You can download the full source code of this example here: **PHPMailFunctionExample**

Chapter 4

Date Format Example

In this example, we shall show how to deal with dates in PHP. As dates can be represented in many different ways. It is important to find a way that helps to avoid possible unexpected errors that may arise.

For this example, we will use:

- Ubuntu (14.04) as Operating System.
- Apache HTTP server (2.4.7).
- PHP (5.5.9).

4.1 Preparing the environment

4.1.1 Installation

Below, commands to install Apache and PHP are shown:

```
sudo apt-get update
sudo apt-get install apache2 php5 libapache2-mod-php5
sudo service apache2 restart
```

Note: if you want to use Windows, installing XAMPP is the fastest and easiest way to install a complete web server that meets the prerequisites.

4.2 How should dates be stored?

As you will probably know, the date representation varies from region to region. For example, in UK, we would say that today is 13/05/2016; whereas in US, we would represent it as 05/13/2016.

This can lead to a problem. If we have a system that needs to do some calculations basing on the date, if a date is in an unexpected format, we would have nonsense results.

To solve this, a system known as **Unix time**, also known as **Epoch time** was proposed. What this system does, is describe time instants: it's defined as the number of seconds passed from 01/01/1970 at 00:00:00 (UTC). So, the date we mentioned above, in Unix time, would be `1463097600` (truncated to the day start, at 00:00:00).

We could define it as a "cross-region/language time representation".

The Unix time is universally used not only in Unix-like systems, but also in many other computational systems. So, with Unix time, we have widely used standard that does not depend on any region or system configuration. In the case a user has to deal with this data, the only thing we would have to do is to transform it to a human-readable format.

Note: Unix time is not a real representation of UTC, as it does not represent leap seconds that UTC does.

4.3 PHP examples

4.3.1 From time stamp to human-readable

Below, we are seeing how to get the current time stamp, and to get a formatted date:

timestamp_to_human.php

```
<?php

$timestamp = time();
$date = date('d-m-Y', $timestamp);
$datetime = date('d-m-Y h:i:s', $timestamp);

echo "Unix timestamp: $timestamp <br>";
echo "Human-readable date: $date <br>";
echo "Human-readable datetime: $datetime";
```

The `date()` function supports a wide list of formats, some of them are:

- `l` for day name.
- `N` for day number of the week, being Monday the first.
- `z` for day number of year.
- `W` for month number of year.

Updating the script to use them:

timestamp_to_human.php

```
echo 'Day name: ' . date('l', $timestamp) . '<br>';
echo 'Day number of week: ' . date('N', $timestamp) . '<br>';
echo 'Day number of year: ' . date('z', $timestamp) . '<br>';
echo 'Month number of year: ' . date('W', $timestamp) . '<br>';
```

Note: The second parameter, were we have specified the time stamp, is optional. If we don't specify any time stamp, current will be used.

4.3.2 From human-readable to time stamp

Now, the reverse process: from human-readable format, to Unix time stamp:

human_to_timestamp.php

```
<?php

$ukDate = '13-05-2016'; // For UK format, '-' must be the separator.
$usDate = '05/13/2016'; // For US format, '/' must be the separator.

$ukTimestamp = strtotime($ukDate);
$usTimestamp = strtotime($usDate);

echo "Timestamp created from UK date format: $ukTimestamp <br>";
echo "Timestamp created from US date format: $usTimestamp";
```

`strtotime()` function does the job. Additionally, the function supports expressions like:

- Adverbs, like `now`, `tomorrow`; and `next Friday`, `last Saturday`, etc.

- Mathematical expressions, e.g., `+2 weeks 5 days 12 hours`.

Let's see it:

human_to_timestamp.php

```
echo "Tomorrow's time stamp is: " . strtotime('tomorrow') . '<br>';
echo 'And in 2 weeks, 5 days and 12 hours time: ' . strtotime('+2 weeks 5 days 12 hours') . ↩
    '<br>';
```

4.3.3 DateTime class

Apart from those functions, it is also available a class named `DateTime`, with some useful methods.

We can, for example, calculate the difference between two dates:

difference_with_datetime.php

```
<?php

$currentDate = new DateTime();
$targetDate = new DateTime('19-01-2038'); // Date when Unix timestamp will overflow in 32  ↩
    bits systems!

$timeLeft = $currentDate->diff($targetDate);

$yearsLeft = $timeLeft->y;
$monthsLeft = $timeLeft->m;
$daysLeft = $timeLeft->d;
$hoursLeft = $timeLeft->h;
$minutesLeft = $timeLeft->i;
$secondsLeft = $timeLeft->s;

echo "Time left to Unix timestamp overflow in 32 bits systems:";
echo "| $yearsLeft years | $monthsLeft months | $daysLeft days | $hoursLeft hours |  ↩
    $minutesLeft minutes | $secondsLeft seconds |";
```

As we can see, we can specify a date when instantiating `DateTime` class, or we cannot specify it in order to use the current datetime.

4.3.4 Increasing precision

In some scenarios, it could occur that the time stamp precision is not enough. For these occasions, we have available the `microtime()` function, which gets the current Unix time, with microseconds. Let's see an example with this:

datetime_with_microseconds.php

```
<?php

$microtime = microtime();

list($microseconds, $seconds) = explode(' ', $microtime);

$microseconds = str_replace('0.', '', $microseconds);

$datetime = date('d-m-Y h:i:s');
$datetime .= ':' . $microseconds;

echo "Current datetime with microseconds: $datetime";
```

For some reason, `microtime()` function returns a string with the following format: `<microseconds> <seconds>`, separated by a blank space. So, we have to split it.

The microseconds are expressed as real number, so, we should remove the leading zero integer part. Then, we can manually add to the formatted datetime the microseconds part.

The funny fact is that, actually, the date() function does support the microseconds formatter, *u*. But it's not able to generate the microseconds. If you try to get a date using `date('d-m-Y h:i:s:u');` it will generate only zeros in the microseconds part. What were PHP developers thinking in?

4.3.5 Validating user introduced date

If we are having a web page where the user has to introduce same date, probably, we will use a JQuery plugin to allow the user select the date graphically. In any case, after performing any operations, we should ensure that the date is correct.

If you try to calculate the time stamp of a nonsense date, for example, `40/15/2016`, you will receive the following time stamp: `1555279200`, with no error. And, if you try to calculate the date of that time stamp, you will receive a correct date, `14/04/2019`.

Depending on the scenario, this can be something to check, or not. If you are expecting a date introduced by an user, normally, we want to ensure that it is actually a valid date expressed in human-readable format. Let's see how we would validate it in the following example:

validate_date.php

```
<?php

define('DEFAULT_DATE_SEPARATOR', '-');
define('DEFAULT_DATE_FORMAT', 'd' . DEFAULT_DATE_SEPARATOR . 'm' . DEFAULT_DATE_SEPARATOR . ↩
    'Y'); // 'd-m-Y'.

/**
 * Checks the validity of the date, using "checkdate()" function. Leap years are also taken ↩
    into consideration.
 *
 * @param $date The input date.
 * @return True if the date is valid, false if it is not.
 */
function validateDate($date) {
    list($day, $month, $year) = explode(DEFAULT_DATE_SEPARATOR, $date);

    $validDate = checkdate($month, $day, $year); // Not a typo; the month is the first ↩
        parameter, and the day the second.

    return $validDate;
}

// Flow starts here.

$inputDate = $_GET['input-date'];
$validDate = validateDate($inputDate);

if ($validDate) {
    $response = "The date $inputDate is valid.";
} else {
    $response = "Error: the date $inputDate is not correct.";
}

echo $response;
```

Fortunately, PHP has a built-in function for this cases, named `checkdate()`, that checks the validity of a Gregorian date, considering also leap years. So, `29-02-2016` would be considered valid, while `29-02-2015` wouldn't.

The problem here is that we have to parse manually the date, to extract the day, month and year, and also to the specified format. If we use any other function to change the format, or to retrieve the date numbers, it would automatically fix the date, doing the same as we saw before the example, so we could not validate the date.

4.4 Summary

We have seen that saving dates in any human-readable format is not a good idea, because the same date can have many different representations, which will depend on the region. We have seen that Unix time format solves this, using a portable format, that does not rely on region or languages, and how to deal with this format in PHP.

4.5 Download the source code

This was an example of date formatting with PHP.

Download You can download the full source code of this example here: **PHPDateFormatExample**

Chapter 5

SoapClient Example

In this example we will see an example of a SOAP client communicating with a server. SOAP is one of many web service protocol definition. A web service is a technology that uses a protocol (SOAP in this example) for exchanging data between applications over the network.

If we are developing a client for consuming a web service, we have to look at the protocol that the service is using. This example is for looking at the resources provided by PHP for developing a client for a web service implementing the SOAP protocol.

For this example, we will use:

- Ubuntu (14.04) as Operating System.
- Apache HTTP server (2.4.7).
- PHP (5.5.9).

5.1 Preparing the environment

5.1.1 Installation

Below, commands to install Apache and PHP are shown:

```
sudo apt-get update
sudo apt-get install apache2 php5 libapache2-mod-php5
sudo service apache2 restart
```

5.1.2 PHP configuration

Even if it's not necessary, is recommendable to disable the SOAP WSDL cache for development environments. In `/etc/php5/apache2/php.ini` file, set the `soap.wsdl_cache_enabled` directive to `0`:

php.ini

```
soap.wsdl_cache_enabled=0
```

Don't forget to restart Apache after doing any change.

Note: if you want to use Windows, installing XAMPP is the fastest and easiest way to install a complete web server that meets the prerequisites.

5.2 What is SOAP?

SOAP (Simple Object Access Protocol) is a standard protocol that defines how two different objects in different processes can communicate each other, through data exchange in XML format. Is one of the widest protocols used in web services.

In other words, this protocol allows to call method of objects that are defined in remote machines.

5.3 PHP example of SOAP clients

The server offering their SOAP services can define them in two different ways:

- With WSDL (Web Services Definition Language)
- Without WSDL

From the client point of view, there are very few differences, but let's see how to proceed for each one.

We will consider the following scenario, which is quite typical: a web service that inserts and reads from a database (simulated with a plain text file), for which we will develop a client.

Is responsibility of the server offering the service to let know the protocol used, as same as the available method definitions, in order to let the clients know how to deal with the service.

Even if this tutorial is about the client itself, we will see also the server side, in order to test that the client is actually working.

5.3.1 Working directory structure

For this example, we will use the following structure:

```
php_soapclient_example/
        simple_soap_client_class.php
        simple_soap_server_class.php
        handle_soap_request.php
        no_wsdl
                server_endpoint.php
        wsdl
                server_endpoint.php
                simple_service_definition.wsdl
```

Where the root directory, `php_soapclient_example`, will be in the web server's root directory, which is `/var/www/html/` by default.

Briefly explained each file:

- `simple_soap_client_class.php` is the class defined for the SOAP client methods.
- `simple_soap_server_class.php` is the class defined for the SOAP server methods.
- `handle_soap_request.php` is for instantiate and use the `SimpleSoapClient` class defined in `simple_soap_client_class.php`.
- In the directories `wsdl` and `no_wsdl`, the service defined in `simple_soap_server_class.php` is offered, in the way it accords to each of the modes. We won't go into details for these, since it is not the purpose of this example. The only thing we have to know is that in one of the directories there is a service offered in WSDL mode, and that in the other, in no WSDL mode; they are two independent web services that use the code defined in `simple_soap_server_class.php`.

Note: the WSDL mode needs a `.wsdl` file, where the web service is defined. We are not going to see it in this example, since the aim is not to see how to build a WSDL definition file, but it's also available to download in the last section of the example.

5.3.2 The server

As said, our server will read and write data into a text file. Let's see it:

simple_soap_server_class.php

```
<?php

/**
 * Methods of our simple SOAP service.
 */
class SimpleSoapServer {

    const FILENAME = 'data.txt';

    /**
     * Inserts data. Invoked remotely from SOAP client.
     *
     * @param $data Data to insert.
     * @return Resume of operation.
     */
    public function insertData($data) {
        $writtenBytes = file_put_contents(self::FILENAME, $data . '<br>', FILE_APPEND);

        if ($writtenBytes) {
            $response = "$writtenBytes bytes have been inserted.";
        } else {
            $response = 'Error inserted data.';
        }

        return $response;
    }

    /**
     * Reads data. Invoked remotely from SOAP client.
     *
     * @return Data of file.
     */
    public function readData() {
        $contents = file_get_contents(self::FILENAME);

        return $contents;
    }
}
```

So simple, about reading and writing into a text file.

Note the functions this server implements, in lines 16 and 33. This functions will be those invoked by the client.

5.3.3 The client

As we said above, depending of the mode (WSDL or not WSDL), the client has to handle the connection in a different way but, once stablished, the procedure is the same. So, let's code a class that works for both modes, handling the SoapClient class instantiation accordingly to the mode:

simple_soap_client_class.php

```
<?php

/**
 * Methods for dealing with SOAP service.
 */
```

```
class SimpleSoapClient {

    const MODE_WSDL = 'wsdl';
    const MODE_NO_WSDL = 'no_wsdl';

    private $client;

    /**
     * SimpleSoapClient class constructor.
     *
     * @param $soapMode The SOAP mode, WSDL or non-WSDL.
     * @param $serverLocation The location of server.
     */
    public function __construct($soapMode, $serverLocation) {
        $this->initializeClient($soapMode, $serverLocation);
    }

    /**
     * Instantiates the SoapClient, depending on the specified mode.
     *
     * For WSDL, it just has to be instantiated with the location of the service, which ↩
        actually has to be the
     * .wsdl location.
     *
     * For non-WSDL, the first parameter of the constructor has to be null; and the second, ↩
        an array specifying
     * both location and URI (which can be the same, the important parameter is the ↩
        location).
     */
    protected function initializeClient($soapMode, $serverLocation) {
        switch ($soapMode) {
            case self::MODE_WSDL:
                $this->client = new SoapClient($serverLocation);

                break;

            case self::MODE_NO_WSDL:
                $options = array(
                    'location' => $serverLocation,
                    'uri' => $serverLocation
                );

                $this->client = new SoapClient(NULL, $options);

                break;

            default:
                throw new Exception('Error: invalid SOAP mode provided.');
                break;
        }
    }

    /**
     * Inserts data remotely into the SOAP service.
     *
     * @param $data Data to insert remotely.
     * @return Response from remote service.
     */
    public function insertData($data) {
        $response = $this->client->insertData($data);

        return $response;
```

```
    }

    /**
     * Reads data from SOAP service.
     *
     * @return Data received from remote service.
     */
    public function readData() {
        return $this->client->readData();
    }
}
```

The lines 35 and 45 instantiate the `SoapClient` class, depending on the `$mode` received. This is how it works for each mode:

- For WSDL mode, we have just to pass the server location to the constructor. For this mode, the **location must be the WSDL definition file (`.wsdl`), not a PHP file**.
- For non WSDL mode, the first parameter has to be null (because we are not accessing a WSDL definition). So, the location must be defined in a different way, providing an array with `'location'` and `'uri'` elements, with the server location as values. In this case, the **location must be the PHP file handling the web service**.

After the instantiation, the communication with the service is pretty simple. We have just to call the methods that we saw defined in `SimpleSoapServer` class, in `simple_soap_client_class.php`, through the `SoapClient` class instance. As we said before, we are calling methods that are defined in other place. What PHP `SoapClient` does, is to provide us those methods defined by the web service, and, when we call them, it will execute them in the server through the SOAP protocol that it has already implemented, with no needing to care about how it works. Seems magic, doesn't it?

If you don't believe it, let's see a script to use this client.

5.3.4 Using the client

The following script allows us to use the client to communicate with the service, through GET parameters. These are the parameters available:

- `'mode'`, to specify the mode (WSDL or non WSDL).
- `'action'`, to specify the action to perform. Available values are `'insert'` and `'read'`.
- `'value'`, to specify the value to insert, only necessary when the action is `'insert'`.

handle_soap_request.php

```
<?php

require_once('simple_soap_client_class.php');

// GET parameter definition.
define('ACTION_INSERT', 'insert');
define('ACTION_READ', 'read');
define('INSERT_VALUE', 'value');
define('MODE_WSDL', 'wsdl');
define('MODE_NO_WSDL', 'no_wsdl');

// Server location definition.
define('LOCATION_WSDL', 'https://127.0.0.1/php_soapclient_example/wsdl/ ↩
    simple_service_definition.wsdl');
define('LOCATION_NO_WSDL', 'https://127.0.0.1/php_soapclient_example/wsdl/server_endpoint. ↩
    php');

// Function definitions.
```

```
/**
 * Checks if the given parameters are set. If one of the specified parameters is not set,
 * die() is called.
 *
 * @param $parameters The parameters to check.
 */
function checkGETParametersOrDie($parameters) {
    foreach ($parameters as $parameter) {
        isset($_GET[$parameter]) || die("Please, provide '$parameter' parameter.");
    }
}

/**
 * Instantiates the SOAP client, setting the server location, depending on the mode.
 * If any error occurs, the page will die.
 *
 * @param $mode The SOAP mode, 'wsdl' or 'no_wsdl'.
 * @return SoapClient class instance.
 */
function instantiateSoapClient($mode) {
    if ($mode == MODE_WSDL) {
        $serverLocation = LOCATION_WSDL;
    } else {
        $serverLocation = LOCATION_NO_WSDL;
    }

    try {
        $soapClient = new SimpleSoapClient($mode, $serverLocation);
    } catch (Exception $exception) {
        die('Error initializing SOAP client: ' . $exception->getMessage());
    }

    return $soapClient;
}

// Flow starts here.

checkGETParametersOrDie(['mode', 'action']);

$mode = $_GET['mode'];
$action = $_GET['action'];

$soapClient = instantiateSoapClient($mode);

switch($action) {
    case ACTION_INSERT:
        checkGETParametersOrDie([INSERT_VALUE]);
        $value = $_GET[INSERT_VALUE];

        try {
            $response = $soapClient->insertData($value);
            echo "Response from SOAP service: $response<br>";
        } catch (Exception $exception) {
            die('Error inserting into SOAP service: ' . $exception->getMessage());
        }

        break;

    case ACTION_READ:
        try {
            $data = $soapClient->readData();
```

```
            echo "Received data from SOAP service:<br>";
            echo $data;
        } catch (Exception $exception) {
            die('Error reading from SOAP service: ' . $exception->getMessage());
        }

        break;

    default:
        die('Invalid "action" specified.');
        break;
}
```

Let's try it.

If we enter in the browser `https://127.0.0.1/php_soapclient_example/handle_soap_request.php?mode=no_wsdl&action=insert&value=testing_no_wsdl`, the service will create a `no_wsdl/data.txt` file (if not exists already), writing the provided value `'testing_no_wsdl'`, and the following will be printed: `Response from SOAP service:19 bytes have been inserted.` (The 4 bytes extra correspond to additional `<br>` characters inserted into the file).

The service offers a method to reading all the data, so, if we enter in the browser `https://127.0.0.1/php_soapclient_example/handle_soap_request.php?mode=no_wsdl&action=read`, the following will be printed: `Received data from SOAP service:testing_no_wsdl` We can check it also for WSDL mode, setting `mode=wsdl` in the parameters.

5.4 Considerations

As said in section 3, when we are developing a web service client (no matter if SOAP, REST, etc.), the service provider has to provide also documentation about the available methods; we need to know the definition of these: which parameters is expecting, af if it returns something. Because of that, the most usual way to develop a SOAP web service is using WSDL files, to provide documentation of the available methods. In any case, the `SoapClient` has a method to get the available methods, for debugging purposes, named `__getFunctions()`. So, we could see the offered methods by the service with the following:

```
var_dump($soapClient->__getFunctions())
```

Assuming that `$soapClient` has been instanced properly.

Also note that, if the service does not offer an encrypted connection, the communication between the client and the server will be made in plain text. So, if we have to develop a client that has to deal with sensible information, we should ensure that the communication with the server can be considered safe.

5.5 Summary

We have seen how to develop a client for a SOAP web service using PHP's SoapClient class. The server may offer the service using WSDL, or not, something to take into account when we are instantiating the `SoapClient` class. Then, to use the methods provided by the service, we just have to call those methods in `SoapClient` instance, as they were ours.

5.6 Download the source code

This was an example of SoapClient with PHP.

Download You can download the full source code of this example here: **PHPSoapClientExample**

Chapter 6

Login Form Example

In this example we will see how to make a login mechanism in PHP. The login systems is one of the most liked targets of attackers, so we have to be extremely careful with every aspect of the login.

For this tutorial, we will use:

- Ubuntu (14.04) as Operating System.
- Apache HTTP server (2.4.7).
- PHP (5.5.9).
- SQLite3, a lightweight and process-less DBMS. Credentials are almost always saved in databases, so we have chosen the lightest option for developing purposes. This not should be an option in production environments.

6.1 Preparing the environment

6.1.1 Installation

Below, commands to install Apache, PHP and SQLite are shown:

```
sudo apt-get update
sudo apt-get install apache2 php5 libapache2-mod-php5 php5-sqlite
sudo service apache2 restart
```

6.1.2 PHP configuration

We have to configure PHP to add the SQLite driver. Open `/etc/php5/apache2/php.ini`, and add the following directive, if it not exists:

```
extension=sqlite.so
```

Don't forget to restart Apache after doing any change.

Note: check the write permissions of your working directory, since the database will be placed in that directory.

6.2 How should passwords be stored?

After developing any login system, we first need to decide how the passwords are going to be stored. Let's see which are the possible ways to store password, from worse option, to best:

6.2.1 Worse: plain text

Even if can be an obviousness, is something that must be said (mostly, because it's still being done): **please, don't store passwords in plain text**. At the same time a password is stored as plain text, it's compromised. Anyone having access to the password storage (commonly, a database), will know the password, when is something that is thought to be known only by its owner. And if the system is compromised by the attacker, it would have all the passwords readable.

6.2.2 Not bad: password hashing

To avoid this, password hashing was designed. This consists on calculating the hash of the password, to store that hash. Hash functions are one-way functions, that is, if `hf(password) =hash`, where `hf` is the hash function, is computationally unfeasible to deduce `password` from `hash`. So, for a login, the stored hash has to be compared with the hash of the password provided but the user.

Password hashing is a much better password storing system that plain text, but keeps being vulnerable. Precomputed tables of hash values could be used (many of them are available already on Internet) to force the login, or to get the original password from its hash, also called digest. One of the features of hash functions is that they are extremely fast, and this, for login, is a disadvantage, since an attacker has more chances to attempt logins in less time.

6.2.3 Better: password hashing + salting

This solves, partially, the problem described in paragraph above. Salting consists on adding a random string of bytes (called salt) to the password before its hashing. So, same passwords, would have different hashes. To perform the login, the hash of the password provided by the user plus the stored salt for that user has to be calculated, and then compare with the stored one.

If an attacker uses a precomputed table of hash values, it still could attempt a brute-force attack, but it won't be that easy to revert the hash to get the plain password, because the random salt has been part of the hash calculation.

In this case, to calculate the hash, we would do the following: `hf(password + salt) =hash`.

Note: when calculating salts, strong random functions based on entropy should be used, not that weak and predictable functions like `rand()`.

6.2.4 Even better: Key Derivation Functions

As we said above, password hashing plus salting helps to keep the passwords secret against precomputed table-like attacks. But an attacker could gain access to a system with a brute-force attack, without needing to know the password.

To avoid this, Key Derivation Functions (KDF) where designed. This functions are **design to be computationally intense**, so, needs much time to derive the key. Here, computationally intense, means a second, maybe something more. Actually, the concept is the same that in hashing + salting, but here another variable is used: the computational cost, which defines the intensity we mentioned above.

So, the operation would be: `kdf(password, salt, cost) =dk`, where `dk` is the derived key generated.

Let's see a time comparison between a common hash function (SHA1, which is deprecated, but it's still widely used), and a KDF (bcrypt):

sha1_vs_bcrypt.php

```
<?php

$password = 'An insecure password';

$starttime = microtime(true);
sha1($password);
$sha1Time = microtime(true) - $starttime;

$bcryptOptions = array('cost' => 14);
$starttime = microtime(true);
```

```
password_hash($password, PASSWORD_BCRYPT, $bcryptOptions);
$bcryptTime = microtime(true) - $starttime;

echo "sha1 took: $sha1Time s <br>";
echo "bcrypt took: $bcryptTime s <br>";
```

The output will be something like this (depending on the hardware):

```
sha1 took: 1.5020370483398E-5 s
bcrypt took: 1.2421669960022 s
```

As you can see, we reduce significantly the attempts an attacker could perform, passing from $1.5*10^{-5}$ seconds, to 1,2 seconds. And, for a user, waiting a second for a login is almost imperceptible.

Is it possible to adjust the intensity the algorithm takes, as in line 9, to find the value that fits better with our hardware.

The derived password (the return value of `password_hash()` function) will be similar to the following string:

```
$2y$14$WH1yQiP1naJD8b8lWOK1bOxGQUjgCpFwuzSKohGL/ZV1NaYYr5Cge
```

Which follows the following format:

```
$<algorithm_id>$<cost>$<salt (22 chars)><hash (31 chars)>
```

So, in that example, would be:

- Algorithm id: `2y` (bcrypt).
- Cost: `14`
- Salt: `WH1yQiP1naJD8b8lWOK1bO`
- Hash, or derived key: `xGQUjgCpFwuzSKohGL/ZV1NaYYr5Cge`

Taking this into account, for authenticating the user, we would have to extract the cost and the salt to reproduce the operation, to then compare the hashes. We will see this in the login implementation.

Note: in the `password_hash()` function, in the `$options` array, we can also specify a salt. If any salt is specified, the function will generate one. For creating salts, the recommended way is using `mcrypt_create_iv()` function.

6.3 Creating users

Let's do a simple script that allows to create user, to later test that our login system works correctly.

create_user.php

```
<?php

require_once('db.php');

function checkGETParametersOrDie($parameters) {
    foreach ($parameters as $parameter) {
        isset($_GET[$parameter]) || die("'$parameter' parameter must be set by GET method." ↩
            );
    }
}

checkGETParametersOrDie(['username', 'password']);

$username = $_GET['username'];
$password = $_GET['password'];
```

```
$db = new DB();
$db->createUser($username, $password);

echo "User '$username' has been created successfully.";
```

To create the user in the database, we developed a class named DB. We will see it below. Now, enter the URL to the script location:

```
127.0.0.1/php_loginform_example/create_user.php?username=myuser&password=mypassword
```

With the username and password you prefer.

6.4 Login

6.4.1 Form

A simple login form that asks for an username and password.

login_form.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Login form</title>
</head>
<body>
    <form action="login.php" method="POST">

            <label for="username">Username:</label>
            <input type="text" name="username" id="username" required>


            <label for="password">Password:</label>
            <input type="password" name="password" id="password" required>


            <input type="submit" value="Login">

    </form>
</body>
</html>
```

6.4.2 Form handling

To handle the form, we will create the following script:

login.php

```
<?php

require_once('db.php');

/**
 * Checks if the given parameters are set. If one of the specified parameters is not set,
 * die() is called.
 *
```

```
 * @param $parameters The parameters to check.
 */
function checkPOSTParametersOrDie($parameters) {
    foreach ($parameters as $parameter) {
        isset($_POST[$parameter]) || die("'$parameter' parameter must be set by POST method ↩
            .");
    }
}

// Flow starts here.

checkPOSTParametersOrDie(['username', 'password']);

$username = $_POST['username'];
$password = $_POST['password'];

$db = new DB();

$authenticated = $db->authenticateUser($username, $password);

if ($authenticated) {
    $response = "Hello $username, you have been successfully authenticated.";
} else {
    $response = 'Incorrect credentials or user does not exist.';
}

echo $response;
```

Actually, only retrieves the parameters sent by the form, and calls DB class function authenticateUser() method. Let's see now that class.

6.4.3 Login against database

The interesting part. This class is the one that interacts with the database, to perform the login, and also to create the users:

db.php

```
<?php

/**
 * Methods for database handling.
 */
class DB extends SQLite3 {

    const DATABASE_NAME = 'users.db';
    const BCRYPT_COST = 14;

    /**
     * DB class constructor. Initialize method is called, which will create users table if ↩
         it does
     * not exist already.
     */
    function __construct() {
        $this->open(self::DATABASE_NAME);
        $this->initialize();
    }

    /**
     * Creates the table if it does not exist already.
     */
    protected function initialize() {
```

```
        $sql = 'CREATE TABLE IF NOT EXISTS user (
                    username STRING UNIQUE NOT NULL,
                    password STRING NOT NULL
                )';

        $this->exec($sql);
    }

    /**
     * Authenticates the given user with the given password. If the user does not exist, ↩
        any action
     * is performed. If it exists, its stored password is retrieved, and then ↩
        password_verify
     * built-in function will check that the supplied password matches the derived one.
     *
     * @param $username The username to authenticate.
     * @param $password The password to authenticate the user.
     * @return True if the password matches for the username, false if not.
     */
    public function authenticateUser($username, $password) {
        if ($this->userExists($username)) {
            $storedPassword = $this->getUsersPassword($username);

            if (password_verify($password, $storedPassword)) {
                $authenticated = true;
            } else {
                $authenticated = false;
            }
        } else {
            $authenticated = false;
        }

        return $authenticated;
    }

    /**
     * Checks if the given users exists in the database.
     *
     * @param $username The username to check if exists.
     * @return True if the users exists, false if not.
     */
    protected function userExists($username) {
        $sql = 'SELECT COUNT(*) AS count
                FROM   user
                WHERE  username = :username';

        $statement = $this->prepare($sql);
        $statement->bindValue(':username', $username);

        $result = $statement->execute();
        $row = $result->fetchArray();

        $exists = ($row['count'] === 1) ? true : false;

        $statement->close();

        return $exists;
    }

    /**
     * Gets given users password.
     *
```

```
     * @param $username The username to get the password of.
     * @return The password of the given user.
     */
    protected function getUsersPassword($username) {
        $sql = 'SELECT password
                FROM   user
                WHERE  username = :username';

        $statement = $this->prepare($sql);
        $statement->bindValue(':username', $username);

        $result = $statement->execute();
        $row = $result->fetchArray();
        $password = $row['password'];

        $statement->close();

        return $password;
    }

    /**
     * Creates a new user.
     *
     * @param $username The username to create.
     * @param $password The password of the user.
     */
    public function createUser($username, $password) {
        $sql = 'INSERT INTO user
                VALUES (:username, :password)';

        $options = array('cost' => self::BCRYPT_COST);
        $derivedPassword = password_hash($password, PASSWORD_BCRYPT, $options);

        $statement = $this->prepare($sql);
        $statement->bindValue(':username', $username);
        $statement->bindValue(':password', $derivedPassword);

        $statement->execute();

        $statement->close();
    }
}
```

Remember when we said in 2.4 section that, for comparing the provided password with the string generated by the KDF algorithm, we would have to extract the cost and the salt from that string? Well, the `password_verify()` function, in line 45, does this for us: repeats the operation for the provided `$password`, extracting the algorithm, the salt and the cost from the existing derived key `$storedPassword`, and then compares it to the original password that is the "reference".

Note that, when querying the database, we use something called $statement. These are called Prepared Statements. We can execute directly a string SQL with a SQLite method called exec(), but this would be vulnerable against SQL Injection, and an attacker could gain access to the system injecting SQL commands. The Prepared Statement does not allow this, because it binds the parameters using a different protocol, and don't need to be escaped looking for characters like `'`.

If we try to fill the form with the credentials we created in section 3, we will receive the following message: `Hello <user>, you have been successfully authenticated.` Whereas, if we introduce incorrect credentials, we will see: `Incorrect credentials or user does not exist.`

6.5 Summary

We have seen that there are different mechanisms to design a login system, analysing the weakness of each one, concluding that the KDFs are the most secure alternative currently, and how to implement it in PHP. We have also seen the need of the use of the Prepared Statements, not to allow hypothetical attackers to introduce malicious commands to gain access or to extract information.

6.6 Download the source code

This was an example of a login form in PHP.

Download You can download the full source code of this example here: **PHPLoginFormExample**

Chapter 7

Curl Get/Post Example

If you have worked with Linux, you will have probably used cURL for downloading resources from the Internet. This powerful library can also be used in PHP scripts, and that is what we are going to see in this example.

For this example, we will use:

- Ubuntu (14.04) as Operating System.
- Apache HTTP server (2.4.7).
- PHP (5.5.9).

7.1 Preparing the environment

7.1.1 Installation

Below, commands to install Apache, PHP and PHP cURL library are shown:

```
sudo apt-get update
sudo apt-get install apache2 php5 libapache2-mod-php5 php5-curl
sudo service apache2 restart
```

7.1.2 PHP configuration

In `/etc/php/apache2/php.ini` file, we need to include the cURL extension, in order to use it:

```
extension=php_curl.so
```

Don't forget to restart Apache after doing any change.

7.2 GET requests

Let's see how we can use cURL to create GET requests:

curl_get.php

```php
<?php

/**
 * Checks if the given parameters are set. If one of the specified parameters
 * is not set, die() is called.
 *
 * @param $parameters The parameters to check.
 */
function checkGETParametersOrDie($parameters) {
    foreach ($parameters as $parameter) {
        isset($_GET[$parameter]) || die("Please, provide '$parameter' parameter.");
    }
}

/**
 * Gets the GET parameters.
 *
 * @return GET parameter string.
 */
function stringifyParameters() {
    $parameters = '?';

    foreach ($_GET as $key => $value) {
        $key = urlencode($key);
        $value = urlencode($value);
        $parameters .= "$key=$value&";
    }

    rtrim($parameters, '&');

    return $parameters;
}

/**
 * Creates the cURL request for the given URL.
 *
 * @param $url The URL to create the request to.
 * @return The cURL request to the url; false if an error occurs.
 */
function createCurlRequest($url) {
    $curl = curl_init();

    if (!$curl) {
        return false;
    }

    $configured = curl_setopt_array($curl, [
        CURLOPT_URL => $url . stringifyParameters(),
        CURLOPT_FOLLOWLOCATION => true,
        CURLOPT_RETURNTRANSFER => true
    ]);

    if (!$configured) {
        return false;
    }

    return $curl;
}

// Flow starts here.
```

```
checkGETParametersOrDie(['url']);

$url = $_GET['url'];

$curl = createCurlRequest($url);

if (!$curl) {
    die('An error occured: ' . curl_error($curl));
}

$result = curl_exec($curl);

if (!$result) {
    die('An error occured: ' . curl_error($curl));
}

echo 'The result of the cURL request:';
echo '<hr>';
echo $result;

curl_close($curl); // Don't forget to close!
```

This is what we have done:

- First, we created a cURL session, with `curl_init()` function, as in line 41. If some error occurs, `false` will be returned, so we have to check it before continuing (line 43).
- Once we have created successfully the cURL session, we have to configure it, as we do with `curl_setopt_array()` function. In this case, we have configured it with the following options:
  - The URL itself. For GET requests, we just have to specify the URL with the parameter string, in `key=value` format, as you already know. Note how is the parameter string constructed in `stringifyParameters()` function: **the values should be encoded to URL encoding**, with `urlencode()` function (lines 24 and 25).
  - The `CURLOPT_FOLLOWLOCATION` option to `true`. This is for following the `3XX` HTTP redirections. If we set to `false`, and the specified URL makes a redirection, we won't reach the final URL.
  - The `CURLOPT_RETURNTRANSFER` option to `true`. This allows to save the HTTP response into a variable. If set to `false`, the response will be printed directly.
- We should always check that the cURL functions don't return any errors, checking the return values of the functions. When `false` is returned, we can get the information of the last error for the given cURL session with `curl_error()` function, as in lines 69 and 75.
- If any error has occurred initializing and configuring the cURL session, we can proceed to execute it, just calling `curl_exec()` function, specifying for which session we are executing the request. In this case, as we configured the `CURLOPT_RETURNTRANSFER` option, we can save the response value into a variable.
- Finally, when we have ended the cURL session handling, **we must close it**, `curl_close()` function.

We can test this script, entering, for example, `localhost/path/to/curl_get.php?url=webcodegeeks.com&s=php` in the browser. We will see that the output generated by the script, is the same of that which we would have received making a search in the above search box, in this page, entering `php`.

7.3 POST requests

For POST requests, we have to configure the cURL options in a slightly different way:

curl_post.php

```
// ...

function createCurlRequest($url) {
    $curl = curl_init();

    if (!$curl) {
        return false;
    }

    $configured = curl_setopt_array($curl, [
        CURLOPT_URL => $url,
        CURLOPT_POST => true,
        CURLOPT_POSTFIELDS => stringifyParameters(),
        CURLOPT_FOLLOWLOCATION => true,
        CURLOPT_RETURNTRANSFER => true
    ]);

    if (!$configured) {
        return false;
    }

    return $curl;
}

// ...
```

Note that, when setting the URL, we just set the URL itself without the parameters, that's for GET requests. Those parameters are specified in `CURLOPT_POSTFIELDS` option. And, apart from that, we are specifying that the request will be made for POST method, with `CURLOPT_POST` option.

The only difference between cURL POST and GET requests is the configuration of the session. The rest of the script, can be the same for both.

7.4 Encapsulating operations in a cURL class

When we are going to use cURL in a project, is recommendable to have encapsulated all the code instead of using every time the functions we have used above, because we would be repeating much code, and the possibilities of introducing errors would increase. So, we are going to create a class that encapsulates this logic.

curl_class.php

```
<?php

/**
 * Methods for cURL request handling.
 */
class CURL {

    /**
     * The cURL request object.
     */
    private $request;

    /**
     * CURL class constructor.
     *
     * @throws Exception if an error occurs initializating.
     */
    public function __construct() {
```

```
        $this->request = curl_init();

        $this->throwExceptionIfError($this->request);
    }

    /**
     * Configure the cURL request.
     *
     * @param $url The target url.
     * @param $urlParameters The array of parameters, with 'key' => 'value' format.
     * @param $method 'GET' or 'POST'; 'GET' by default.
     * @param $moreOptions Any other options to add to the cURL request. By default,
     *    'CURLOPT_FOLLOWLOCATION' (follow 3XX redirects) and 'CURLOPT_RETURNTRANSFER'
     *    (return HTTP response as a value, instead of outputting directly) are set.
     * @throws Exception if an error occurs configurating.
     */
    public function configure($url, $urlParameters = [], $method = 'GET',
        $moreOptions = [CURLOPT_FOLLOWLOCATION => true, CURLOPT_RETURNTRANSFER => true]) {

        curl_reset($this->request);

        switch ($method) {
            case 'GET':
                $options = [CURLOPT_URL => $url . $this->stringifyParameters($urlParameters ↩
                    )];
                break;

            case 'POST':
                $options = [
                    CURLOPT_URL => $url,
                    CURLOPT_POST => true,
                    CURLOPT_POSTFIELDS => $this->stringifyParameters($urlParameters)
                ];

                break;

            default:
                throw new Exception('Method must be "GET" or "POST".');
                break;
        }

        $options = $options + $moreOptions;

        foreach ($options as $option => $value) {
            $configured = curl_setopt($this->request, $option, $value);

            $this->throwExceptionIfError($configured);
        }
    }

    /**
     * Executes the cURL request for the options configured.
     *
     * @return The return value of curl_exec(). If CURLOPT_RETURNTRANSFER was configured,
     *     the return value will be the HTTP response. In other case, a true value (or  ↩
        false
     *     if some error has occured).
     * @throws Exception if an error occurs executing.
     */
    public function execute() {
        $result = curl_exec($this->request);
```

```
        $this->throwExceptionIfError($result);

        return $result;
    }

    /**
     * Closes cURL session.
     */
    public function close() {
        curl_close($this->request);
    }

    /**
     * Checks if a curl_* function returns success or failuer, throwing an exception
     * with the cURL error message if it has failed.
     *
     * @param $success IF the curl function has succeeded or not.
     * @throws Exception if the curl function has failed.
     */
    protected function throwExceptionIfError($success) {
        if (!$success) {
            throw new Exception(curl_error($this->request));
        }
    }

    /**
     * Creates a string of GET parameters.
     *
     * @param $parameters Parameter array.
     * @return Parameters in string format: '?key1=value1&key2=value2'
     */
    protected function stringifyParameters($parameters) {
        $parameterString = '?';

        foreach ($parameters as $key => $value) {
            $key = urlencode($key);
            $value = urlencode($value);

            $parameterString .= "$key=$value&";
        }

        rtrim($parameterString, '&');

        return $parameterString;
    }
}
```

This class doesn't do anything we haven't seen before, except the way the options are configured (in line 60), with `curl_setopt()` function, to set the options with no needing to know which options are.

In order to use this class, we would only have to do something similar to this:

```
<?php

require_once('curl_class.php');

try {
    $curl = new CURL();

    $curl->configure('webcodegeeks.com');
    $response = $curl->execute();
```

```
    $curl->close();
} catch (Exception $exception) {
    die('An exception has been thrown: ' . $exception->getMessage());
}
```

Which is much more clean, readable and easy to maintain. And also more flexible, since it allows to configure any cURL option (and there is a large list of available options).

Note: cURL requests can be reused, increasing the performance. For example:

```
<?php

require_once('curl_class.php');

try {
    $curl = new CURL();

    $curl->configure('webcodegeeks.com');
    $response1 = $curl->execute();

    $curl->configure('webcodegeeks.com', ['s' => 'php'], 'GET');
    $response2 = $curl->execute();

    $curl->close();
} catch (Exception $exception) {
    die('An exception has been thrown: ' . $exception->getMessage());
}
```

The above script will work with any problem, and being more optimal than closing and opening a new cURL session for each request. In this case, the difference would probably be negligible, but for several request can suppose a significant change.

To reuse the cURL sessions, we should clean and remove the options of this before adding new ones, as done in `curl_class.php` in line 38, with curl_reset `curl_reset()` function.

7.5 Summary

This example has shown how we can use the cURL library in PHP scripts, for making both GET and POST requests, which have to be handled in a slightly different way. Apart from that, we have developed a class for handling cURL in a much more ordered, clean and flexible way, allowing to reuse already opened sessions, in order to improve performance.

7.6 Download the source code

This was a cURL example for PHP.

Download You can download the full source code of this example here: **PHPCurlGetPostExample**

Chapter 8

HTML Table Example

In this example we shall show you how to create html tables with dynamic php. PHP is a server-side scripting language used by most web developers for creating dynamic and responsive web pages(A dynamic webpage is one whose content change automatically each time the page is viewed).

Html is simply a language used to describe webpages. Html tables allow web developers arrange data into rows and columns also html tables also allow web developers display structured data in a web page.

For this example we would use

- A computer with PHP 5.5 installed
- notepad++

8.1 Getting Started

To explain dynamic PHP and html tables we are going to develop a trivial web app which allows a teacher record the scores of students on a remote server. The server counts the number of records it has received, after the third count it outputs the received record in an html table and resets its counter. Before we continue let's start by explaining some concepts.

8.1.1 Forms

Html forms are used to collect user input. We are going to use forms to collect each student details(firstname, lastname, subject and score) and submit it

index.html

```
<form action=action.php  method=post onsubmit="return validateForm()">
<input type=text placeholder=FirstName id=firstname name=first required><br>
<input type=text placeholder=LastName id=lastname name=last required><br>
<input type=text placeholder=Subject id=subject name=subject required><br>
<input type=text placeholder="Student Score" id=score name=score required><br>

<input type=submit value=submit id=sub>


</form>
```

This code displays a form to the user, the `required` attribute in each textbox tells the browser that this field is required and must be filled while the `placeholder` is just a short hint that describes the expected input. The `validateForm()` function that is called when the form is submitted checks if the user has entered all the required fields. If the user entered all the required

fields it returns true and the form is submitted else the function returns false and halts the submission process alerting the user to the error encountered.

index.html

```
function validateForm(){
var firstname=document.getElementById("firstname").value.trim();
var lastname=document.getElementById("lastname").value.trim();
var subject=document.getElementById("subject").value.trim();
var score =document.getElementById("score").value.trim();
if(firstname==""||lastname==""||subject==""||score==""){
alert("All Fields Must be Filled")
return false;
}

return true;
}
```

8.1.2 Session in php

We need store each student record processed by our php script, to do this we will make use of php session(in an advanced and scalable web app we would store all records in a database but for the sake of simplicity we would use php session) Simply put a session is a way to store information, which can be accessed across mutiple webpages.When the form is submitted our script access each data submitted and saves it in session variables

action.php

```
<?php
session_start();
if($_SESSION['num']==null)//check if the num variable is empty
$_SESSION['num']=1;// if it is empty set it to 1
?>
```

To use session in php we call the `session_start()` function before anything else is output to the browser. All session variable values are stored in the global $_SESSION associative array. Assigning a session variable is as simple as writing `$_SESSION['num']=1` and we can read from it the same way we would read from a normal variable. We use the `$_SESSION['num']` variable as a counter to store the number of times the script has received a record.

action.php

```
if($_SESSION['num']==1){
$_SESSION['num']++;
 $_SESSION['first']=$_POST['first'];
 $_SESSION['last']= $_POST['last'];
 $_SESSION['subject']=$_POST['subject'];
 $_SESSION['score']=$_POST['score'];
include("index.html");
}
else if($_SESSION['num']==2){
$_SESSION['num']++;
 $_SESSION['first1']=$_POST['first'];
 $_SESSION['last1']= $_POST['last'];
 $_SESSION['subject1']=$_POST['subject'];
 $_SESSION['score1']=$_POST['score'];
include("index.html");
}
else if($_SESSION['num']>=3){
$_SESSION['num']=1;
 $_SESSION['first2']=$_POST['first'];
 $_SESSION['last2']= $_POST['last'];
 $_SESSION['subject2']=$_POST['subject'];
```

```
 $_SESSION['score2']=$_POST['score'];
?>
<!DOCTYPE html>
<html lang=eng>
<head>
<title>
Dynamic php and html table testing
</title>
<style>
html, body{
        width:100%;
        height:100%;
        margin:0%;
        font-family:"helvetica","verdana","calibri", "san serif";
        overflow:hidden;
        padding:0%;
        border:0%;
        }
        table{
        width:50%;
        border:2px solid #343434;
        margin-left:auto;
        margin-right:auto;
        }
        td{
        text-align:center;
    width:25%;
        border:2px solid #343434;
        }
        #he{
height:10%;
background-color:#343434;
margin-bottom:2%;
}
        </style>
        </head>
        <body bgcolor="#e5a010">

        <table>
        <thead>
        Student Result Score
        </thead>
        <tbody>
        <th>Firstname</th><th>Lastname</th><th>Subject</th><th>Score</th>

        <?php
 echo "|" . $_SESSION['first']. "";
 echo "|" . $_SESSION['last']. "";
 echo "|" . $_SESSION['subject']. "";
 echo "|" . $_SESSION['score']. "";
 echo "";

 echo "";
 echo "|" . $_SESSION['first1']. "";
 echo "|" . $_SESSION['last1']. "";
 echo "|" . $_SESSION['subject1']. "";
 echo "|" . $_SESSION['score1']. "";
 echo "";

 echo "";
 echo "|" . $_SESSION['first2']. "";
 echo "|" . $_SESSION['last2']. "";
```

```
 echo "|" . $_SESSION['subject2']. "";
 echo "|" . $_SESSION['score2']. "";
 echo "";
//this removes all the session variable
 session_unset();
 //this functions destroys the session
 session_destroy();
}
?>
<tbody>

</body>
</html>
```

We check how many times our script has received a record, if it's not up to three we increment our counter variable, store each record received in our session array and display the form to the teacher(we do this by calling the `include("index.html")` function). After three records has been submitted we reset the counter variable and output an html table containing each student record to the browser. It is considered good practice to clean up after using session. we call php `session_unset()` to delete all session data and `session_destroy()` to the destroy the session.

8.1.3 Html Tables

Let's start by explaining each tag used in our example

- The `<table>` tag: it is used to define/create a table in html
- The `<thead>` tag: it is used to define an header for a table.
- The `<tbody>` tag: it is used in conjunction with the `<thead>` and `<tfoot>` tag to describe a table. It always comes after the `<thead>` tag and before the `<tfoot>` tag
- The `<tr>` tag: it is used to define a row in an html table
- The `<th>` tag: This tag defines a table header.
- The `<td>` tag: This tag defines a cell in an html table

8.1.4 Other table tags worthy of mention

- `<caption>` tag : This is an optional tag, it gives the title of the table. It is always the first descendant of the `<table>` tag.
- `<tfoot>` : This tag defines a set of rows summarizing the columns of the table.

8.2 Summary

In this example we have learnt about dynamic php and html tables(what they mean and how to use them). we also discussed about html forms and php sessions(how to create and destroy them).

8.3 Download the source code

Download You can download the full source code of this example here: **Phphtmltableexample**