PHP

SUCCINCTLY

BY **JOSÉ ROBERTO OLIVAS MENDOZA**

ᚷSyncfusion®

PHP Succinctly

By

José Roberto Olivas Mendoza

Foreword by Daniel Jebaraj

Syncfusion®
Deliver innovation with ease®

Technical Reviewer: James McCaffrey

Copy Editor: Courtney Wright

Acquisitions Coordinator: Hillary Bowling, online marketing manager, Syncfusion, Inc.

Proofreader: John Elderkin

Table of Contents

The Story behind the *Succinctly* Series of Books

Daniel Jebaraj, Vice President
Syncfusion, Inc.

Staying on the cutting edge

As many of you may know, Syncfusion is a provider of software components for the Microsoft platform. This puts us in the exciting but challenging position of always being on the cutting edge.

Whenever platforms or tools are shipping out of Microsoft, which seems to be about every other week these days, we have to educate ourselves, quickly.

Information is plentiful but harder to digest

In reality, this translates into a lot of book orders, blog searches, and Twitter scans.

While more information is becoming available on the Internet and more and more books are being published, even on topics that are relatively new, one aspect that continues to inhibit us is the inability to find concise technology overview books.

We are usually faced with two options: read several 500+ page books or scour the web for relevant blog posts and other articles. Just as everyone else who has a job to do and customers to serve, we find this quite frustrating.

The *Succinctly* series

This frustration translated into a deep desire to produce a series of concise technical books that would be targeted at developers working on the Microsoft platform.

We firmly believe, given the background knowledge such developers have, that most topics can be translated into books that are between 50 and 100 pages.

This is exactly what we resolved to accomplish with the *Succinctly* series. Isn't everything wonderful born out of a deep desire to change things for the better?

The best authors, the best content

Each author was carefully chosen from a pool of talented experts who shared our vision. The book you now hold in your hands, and the others available in this series, are a result of the authors' tireless work. You will find original content that is guaranteed to get you up and running in about the time it takes to drink a few cups of coffee.

Free forever

Syncfusion will be working to produce books on several topics. The books will always be free. Any updates we publish will also be free.

Free? What is the catch?

There is no catch here. Syncfusion has a vested interest in this effort.

As a component vendor, our unique claim has always been that we offer deeper and broader frameworks than anyone else on the market. Developer education greatly helps us market and sell against competing vendors who promise to "enable AJAX support with one click," or "turn the moon to cheese!"

Let us know what you think

If you have any topics of interest, thoughts, or feedback, please feel free to send them to us at succinctly-series@syncfusion.com.

We sincerely hope you enjoy reading this book and that it helps you better understand the topic of study. Thank you for reading.

Please follow us on Twitter and "Like" us on Facebook to help us spread the word about the *Succinctly* series!

About the Author

I'm an IT businesses entrepreneur, a software developer, and a huge technology fan. My company went to market in 1990 focused primarily on custom software development. We started with COBOL as our main programming language and over the years we've been evolving up to .NET and Office technologies. Throughout this time, some of my core activities have been research about cutting edge technologies, and searching for new tools which can help us to automate some or all process regarding our products development lifecycle.

In 2010, electronic invoicing was set as a requirement for business transactions in Mexico. This event forced us not only to upgrade all our business-related products, but also to change our development paradigm. At that time, we had been focusing only on desktop applications development. But, electronic invoicing involved a series of services which should be delivered over the Internet, such as an electronic documents downloading site. That situation led us to web applications development.

As a Research Manager, I was responsible for choosing which technologies should be employed for web applications development. Because we'd been mostly focused on Microsoft technology, the first thing that came up was ASP.NET. Unfortunately, the company's crew members were not well trained in such technology (at that time), and there weren't candidates in the market to be hired.

I found out that the most of potential candidates to be employed as a web developers were focused on PHP language, so I started a research about it. The results obtained from this investigation made me choose PHP as our primary web development programming language: it is a widely used language in the web development world, there are a pool of developers far larger than ASP.NET, there are a large series of online communities and forums to look for answers to problems, and finally, the fact that the language is open source and it is supported by many web hosting service providers.

As we've been going further into the web world, we realized that there's no unique technology to be employed in web development. At this time, we use things such as PHP, JavaScript, HTML5, CSS, and even ASP.NET. But definitely, PHP is the first option we take into account when we start a new project.

Who Is This Book For?

This book is written for computer programmers who want to get into PHP web development, which requires a basic understanding of databases (particularly MySQL), client/server applications, HTML, and how the Internet works.

The book is structured in chapters and sections, starting with an introduction. The second chapter explains how PHP should be deployed in a Windows environment, and how the deployment can be tested to make us sure the process has been performed correctly.

The third chapter covers some PHP basics, and each one of the subsequent chapters, up to fifth, cover key PHP programming themes such as variables, decision making, arrays, functions, and databases, in order to give the reader all necessary elements to build a simple, PHP-functional webpage. The sixth chapter is dedicated to creating this functional webpage, in the form of a contacts information website, which connects to a database to store information.

Every chapter ends with a summary, which emphasizes the knowledge acquired within it, and the book is summarized entirely in the seventh chapter.

PHP version 7.0.13 will be used for the exercises explained in this book. Also, IIS (Internet Information Services) version 10.0 running on a Windows 10 computer will be employed as a web server. The Microsoft Edge web browser will be used to run code samples.

All code examples discussed in this book can be downloaded here.

Chapter 1 An Introduction to PHP

What is PHP?

PHP is an open source, general-purpose scripting language oriented for web development. This language was originally created by Ramsus Lerdorf in 1994, and it was known as Personal Home Page / Forms Interpreter. The first version of PHP/FI was released in 1995. PHP/FI 2.0 appeared later, in 1997. PHP is now supported by The PHP Group, and the PHP acronym stands for PHP: Hypertext Preprocessor. The current stable release of PHP at the time of writing is 7.0.13.

PHP is normally processed by an interpreter, which is implemented as a web server module, so the web server combines the results of the PHP code and returns a webpage to the client (usually a web browser). The standard PHP interpreter is powered by the Zend Engine, a free software released under the GNU/GPL license that allows the use of the software for any purpose, including commercial projects. The interpreter can be obtained from the PHP Group website under the same license terms.

One of the best features of PHP is its simplicity for newcomers, which allows you to write simple scripts in a matter of hours. The language also offers a series of advanced features for professional programmers. The following table summarizes PHP's main features.

Table 1: PHP Main Features Summary

Feature	Description
Open source and free to use	Perhaps the most important feature for many developers. This feature has made the PHP developers community grow increasingly because there's no need to acquire any license to start with any kind of project.
Multi-platform	PHP can be used in many operating systems, such as Windows, Mac OS X, or Linux. PHP is supported by the most of the web servers today, including Apache and IIS.
Interpreted language	Unlike the C or C++ languages, in which code needs to be compiled to run on computers, PHP code is interpreted at the time it is used. This task is performed by the PHP interpreter implemented in the web server.
Procedural programming support	PHP allows to you employ procedure and function call programming paradigms.
Object-oriented programming support	PHP allows the use of object-oriented programming concepts like inheritance, polymorphism, and abstraction.

Feature	Description
C-like syntax	PHP programming syntax is C language-oriented.
Non-strongly typed	There's no need to specify the data type for variable declaration. The type is determined at run time.
Predefined super-global variables	A set of variables whose names start with **a** _ that can be accessed along the entire script execution, no matter where they are invoked. Some examples are **$_GET**, **$_SESSION**, and **$_SERVER**.
Database support extensions	PHP supports a wide range of databases, such as MySQL and PostgreSQL. ODBC databases are also supported.
Text processing	PHP has a set of useful text-processing features such as regular expressions, or XML documents accessing and parsing.
Non-HTML output capabilities	PHP can generate images, PDF files, or XHTML text on the fly, and save them in the file system. This can be useful to form a server-side cache in order to manage dynamic content.
Error Handling (PHP 7)	PHP 7 throws an exception when an error occurs during script execution. This exception can be caught to avoid script-crashing.

Besides the features detailed in the previous table, we can also say that PHP can be embedded into HTML, or HTML can be called from PHP. In this way, all HTML code for an entire website can be processed by PHP, so the webpages for this site can be created dynamically.

PHP also allows you to create variables dynamically. This is accomplished by using the value of a declared variable as the name for another variable (which will be explained in Chapter 3).

Chapter summary

This chapter gave a brief introduction to the PHP programming language. The most important aspects of PHP are summarized in the following list.

- PHP is an acronym for PHP: Hypertext Preprocessor
- PHP is an open source language, free to download and use
- PHP is executed on the server
- PHP can generate dynamic page content
- PHP can connect to a wide range of databases
- PHP can run on various platforms (Windows, Linux, UNIX, Mac OS X, etc.)
- PHP is compatible with almost all web servers used today (IIS, Apache, etc.)
- PHP is easy to learn for newcomers

Chapter 2 Deploying PHP

Installing PHP in a Windows environment

Prerequisites

In order to install PHP on Windows, the following requirements must be fulfilled:

- The computer should have a Windows operating system installed and running
- IIS (Internet Information Services) should be installed and configured.

Installation process

Setting up IIS

To install IIS on Windows 10, we will open the **Control Panel** and click on the **Programs** category link.
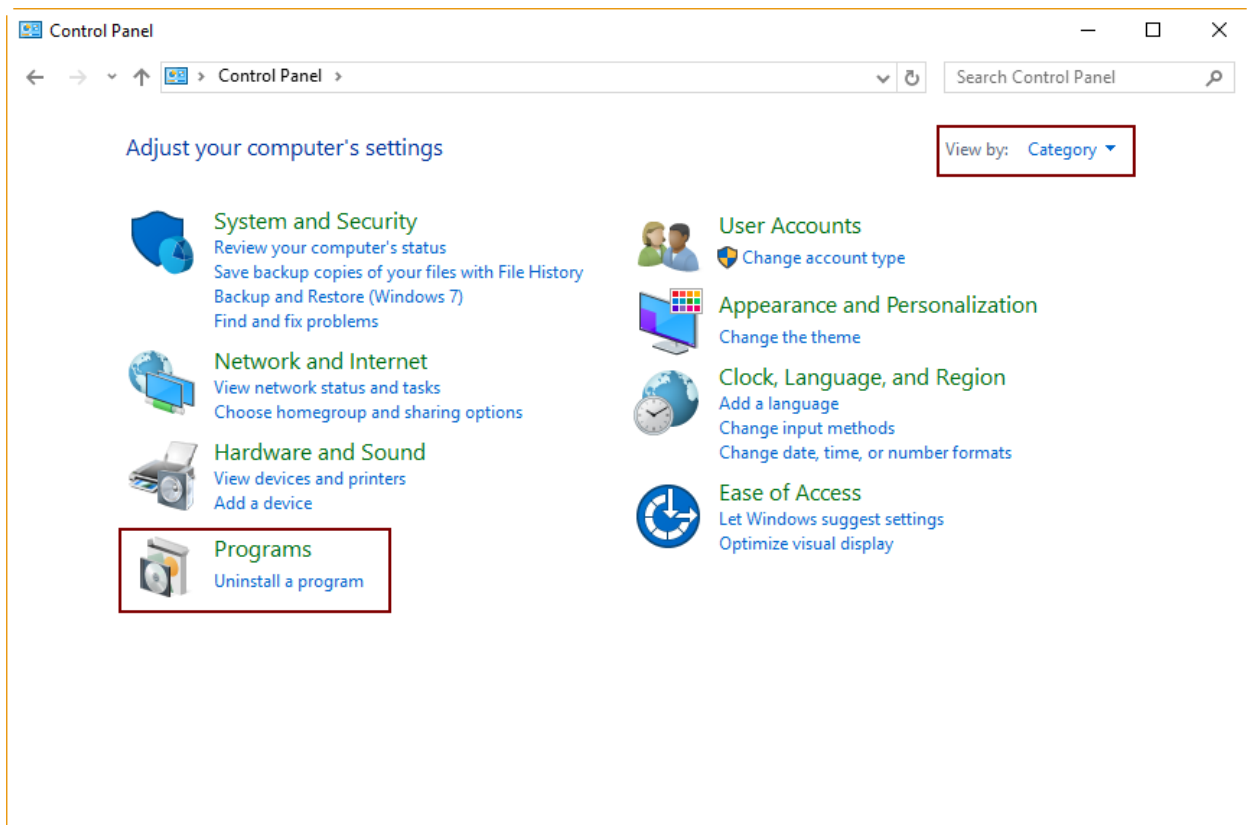
Figure 1: Programs Section in the Control Panel

After the **Programs** dialog is displayed, click **Turn Windows features on or off** to show the **Windows Features** dialog box.
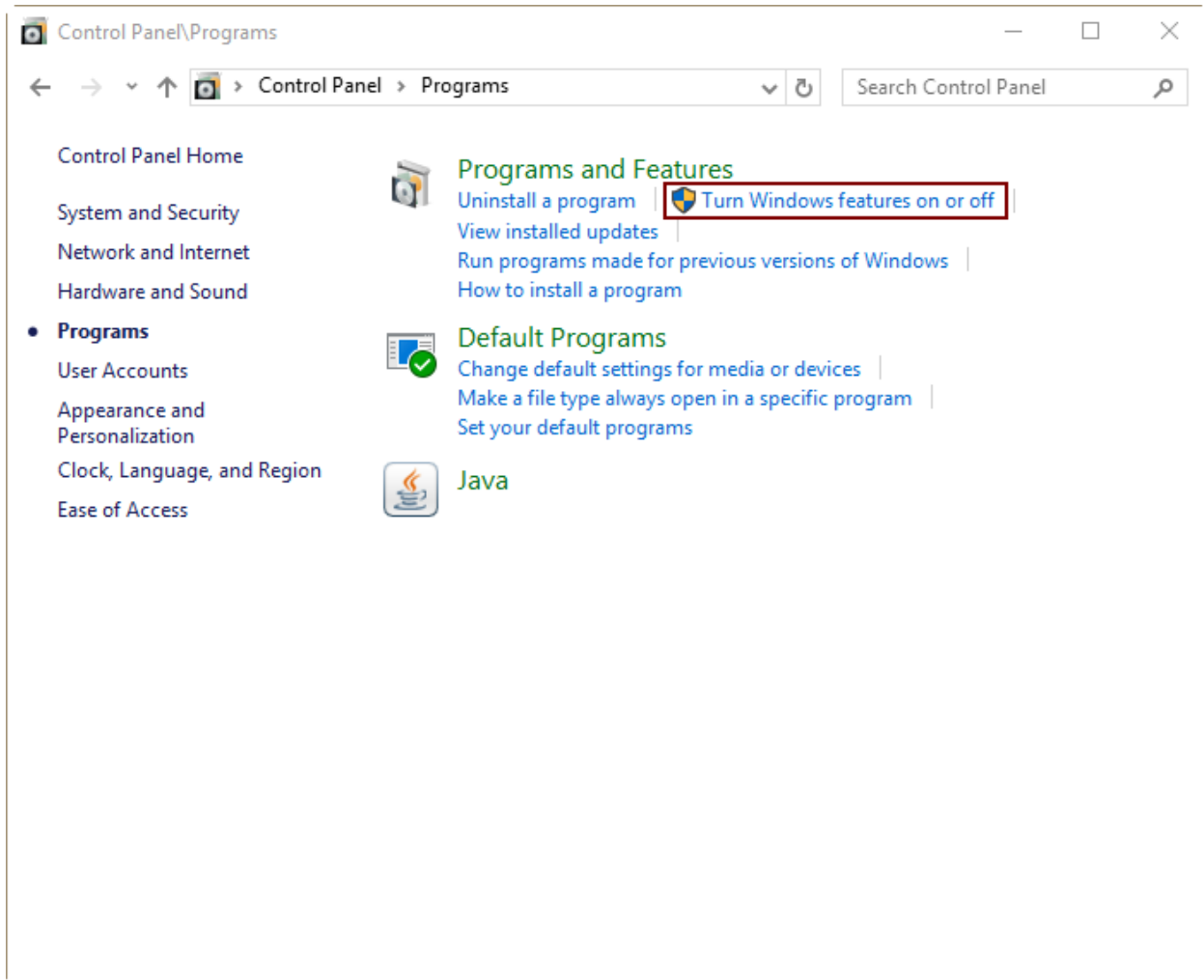
Figure 2: Turning Windows Features On or Off in the Programs Section

In the **Windows Features** dialog box, click on the **Internet Information Services** checkbox to choose all the features needed to host a website in the computer. You must also select the **CGI** entry under **World Wide Web Services | Application Development Features**, because PHP uses CGI, as you'll see shortly.
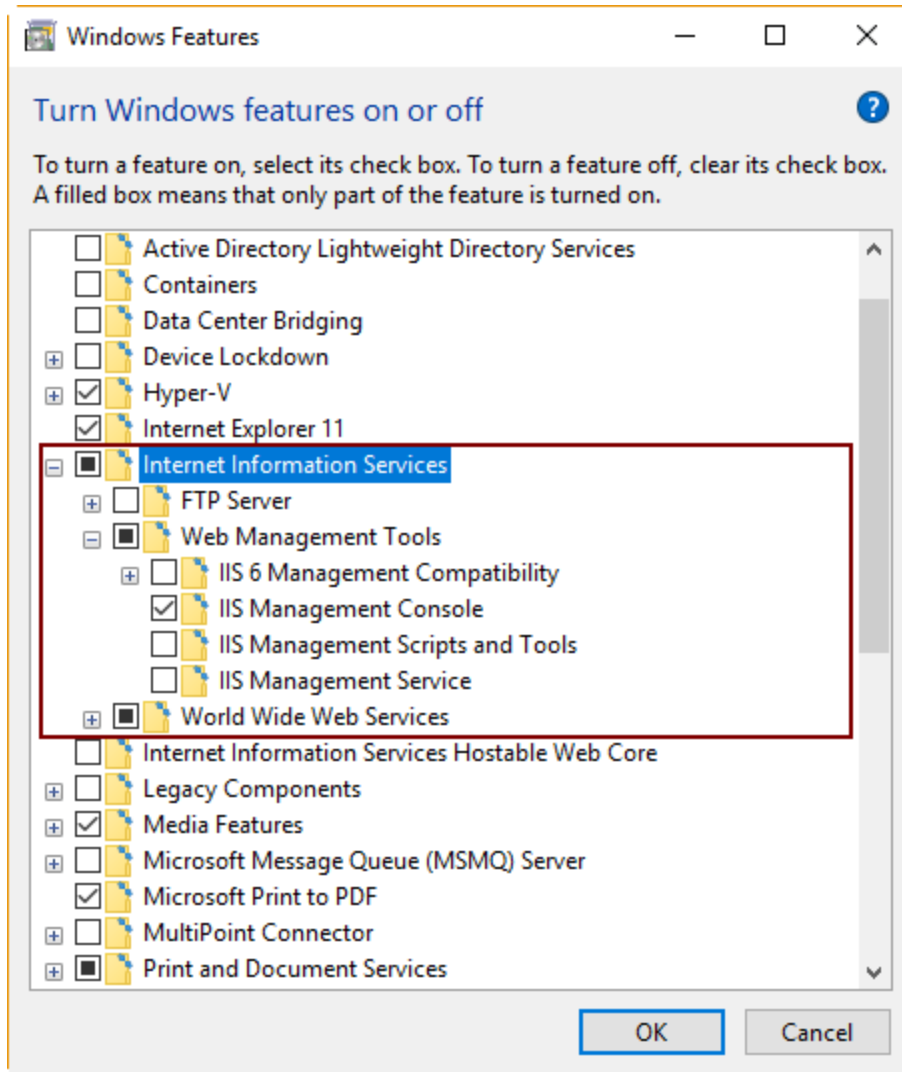
Figure 3: Internet Information Services Feature Selected

After that, click **OK** to begin the installation process.
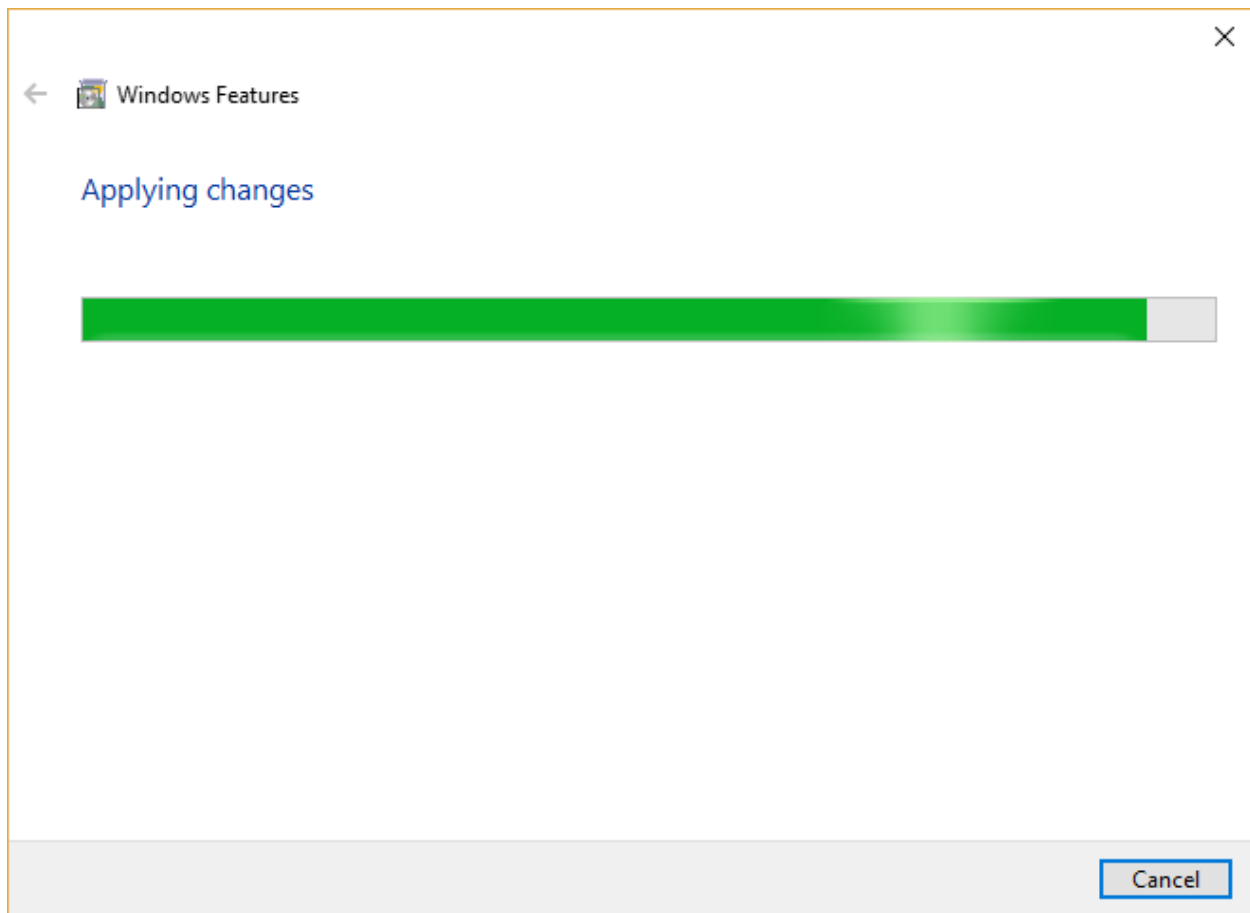
Figure 4: Features Installation Progress

When the process finishes, run your web browser and navigate to **http://127.0.0.1** in order to test the installation.

The browser should display the page shown in the following figure.
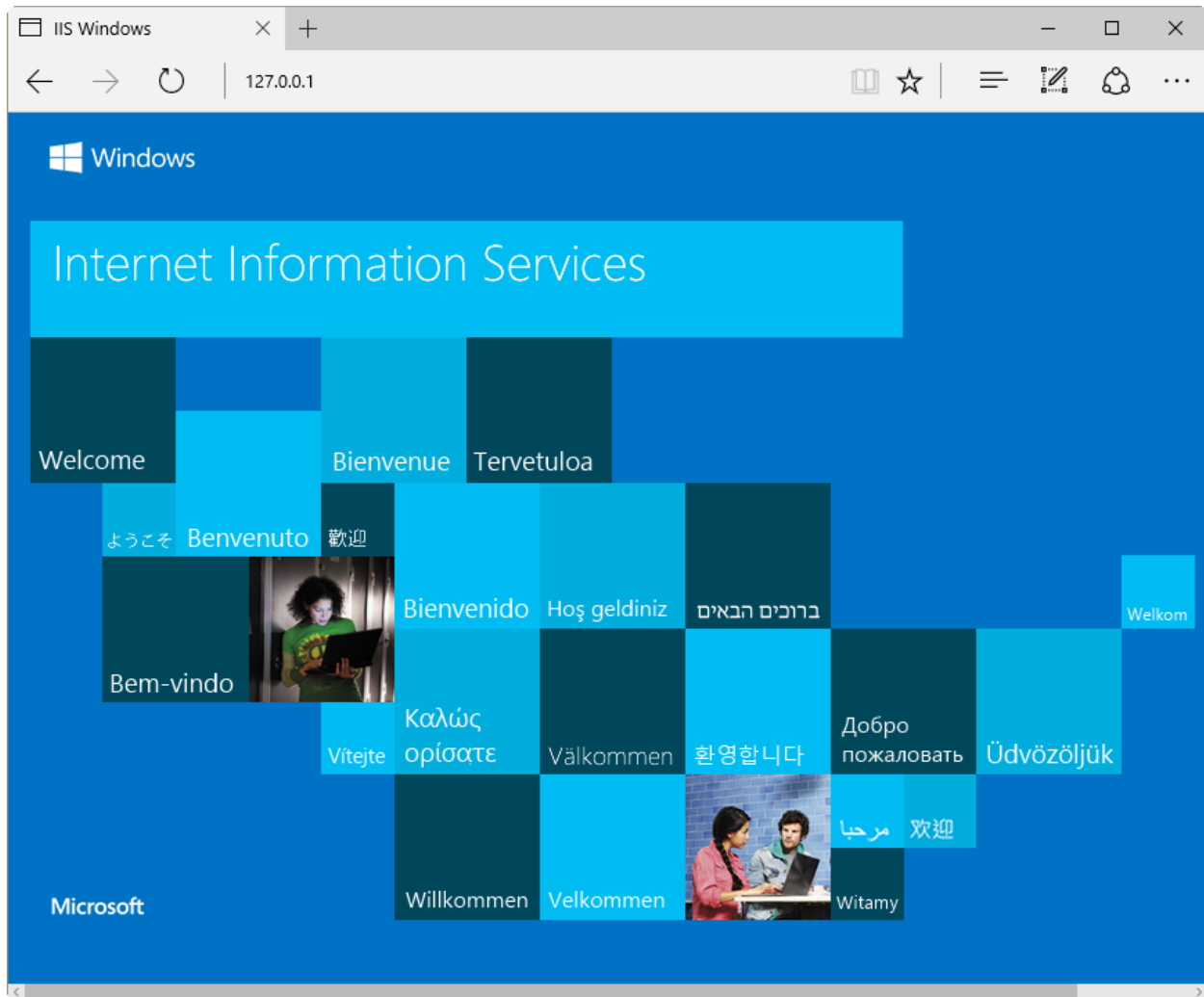
Figure 5: Internet Information Services Home Page

If the page shown in Figure 5 appears in the browser, IIS was installed successfully.

Installing PHP

The fastest and easiest way to install PHP on Windows is by using the Microsoft Web Platform Installer, which automates the process of installing and configuring PHP in the target system. A second way to install PHP is by using a compressed zip file installation. In this case, installing and configuring PHP should be done manually.

For the purposes of this book, we will explain the zip file installation process and manual PHP configuration.

Downloading PHP

Every PHP version has two builds available: a thread-safe version and a non-thread-safe version (NTS). The thread-safe version is intended for environments where a web server can keep the PHP engine in memory, being able to run multiple threads of execution for different web requests simultaneously. Since the architecture of IIS and its FastCGI extension provide an isolation model that keeps requests separate, there is no need for a PHP thread-safe version. The result is an important performance improvement on IIS, because the NTS version of PHP avoids unnecessary thread-safety checks.

According to the previous explanation, we will download the NTS version of PHP and install it in our computer. For a 32-bit system, the PHP NTS version can be obtained from this location. For a 64-bit system, the download URL is here.

Deploying PHP

Once the PHP package is downloaded, we should unpack the files from the zip PHP package into a directory of our discretion in the computer system where IIS is installed (C:\PHP is recommended).
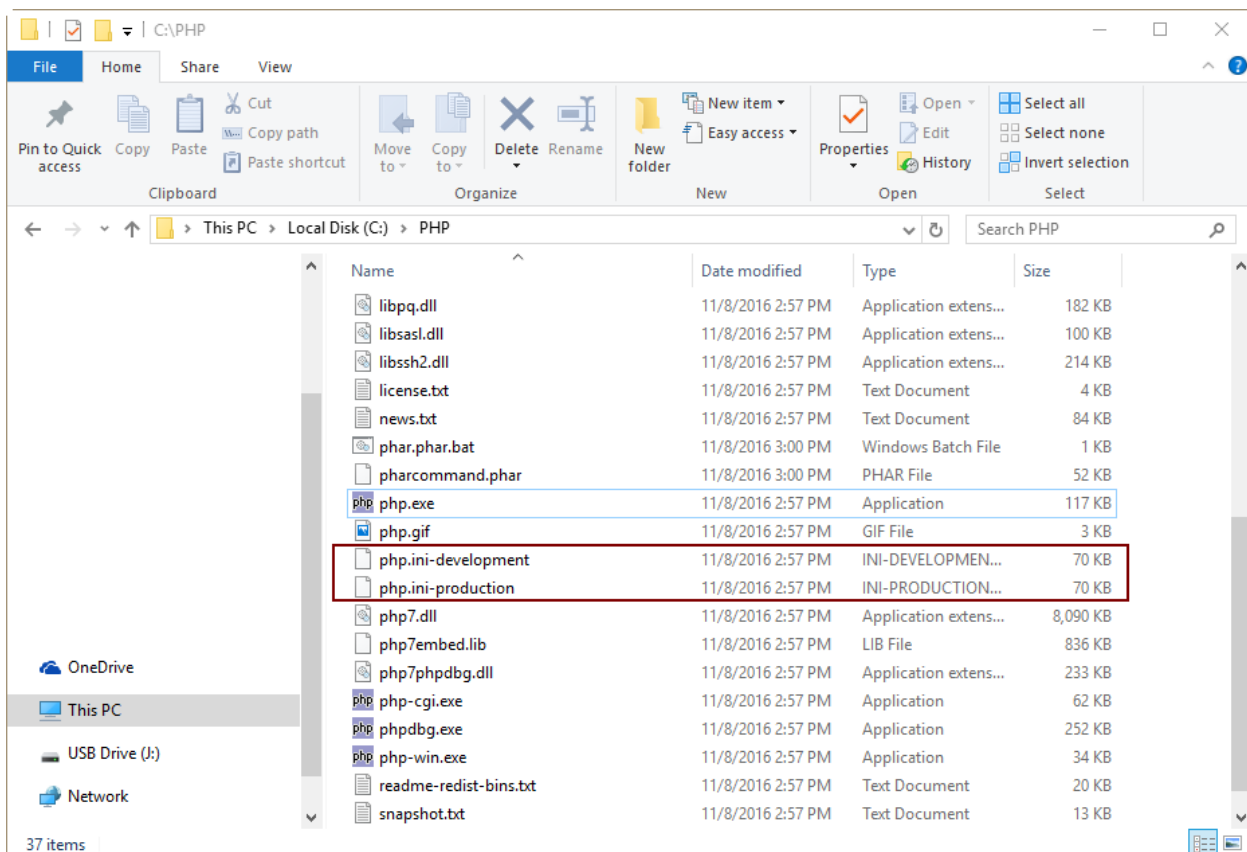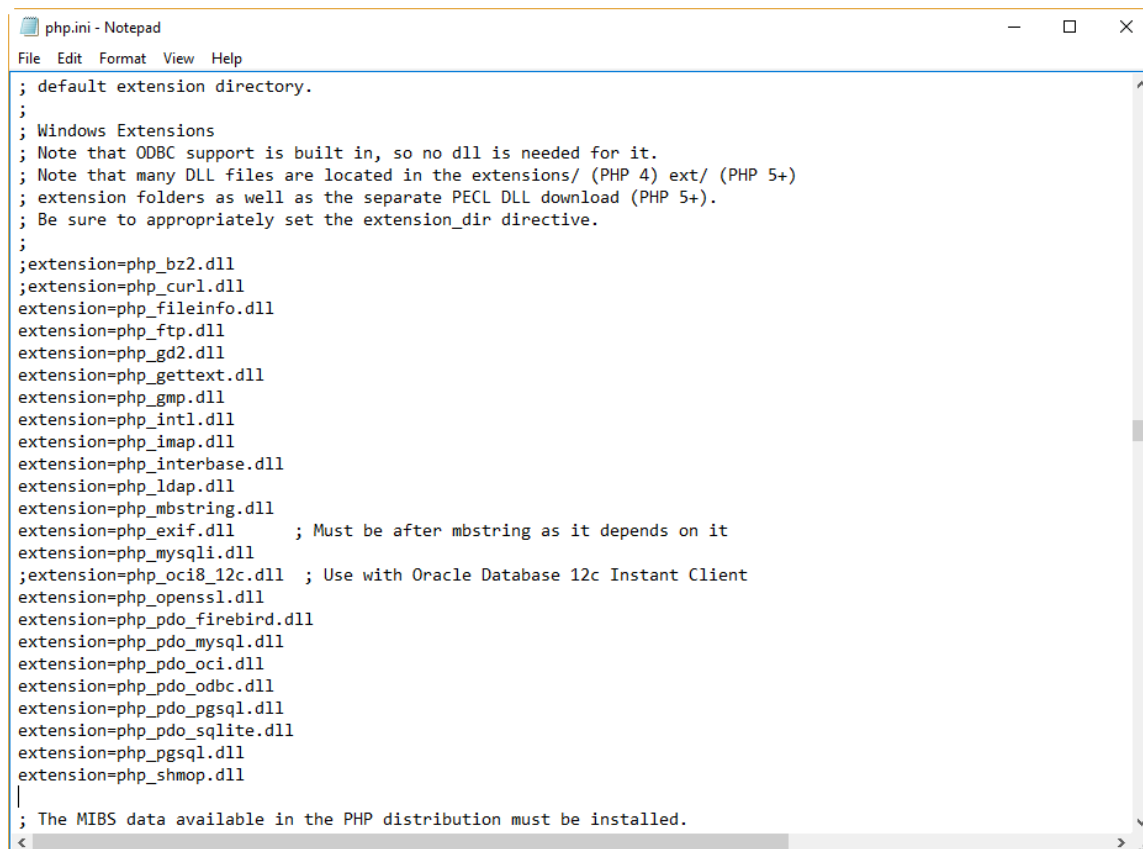
Figure 6: The C:\PHP folder with PHP Files Unpacked

Figure 6 shows the PHP files in the **C:\PHP** directory. We can see that the **php.ini-development** and **php.ini-production** files are highlighted. Those files contain the configuration settings that make PHP work. As suggested by their names, one version of the file is for development purposes, and the other is for deploying in a production environment. For the purposes of this book, we're going to make a copy of the **php.ini-development** file and save it as **php.ini** (so we have the original file as a backup).

Configuring PHP

We're going to open the **php.ini** file using a text editor (commonly Notepad.exe), and then we will uncomment and modify the following settings.

1. Uncomment and set **fastcgi.impersonate = 1**. The FasctCGI IIS extension supports the ability to impersonate security tokens coming from the calling client, allowing IIS to define the security context under which the request will run.
2. Uncomment and set **cgi.fix_pathinfo = 0**. This setting indicates to PHP that **PATH_TRANSLATED** will be set to **SCRIPT_FILENAME**).
3. Set **cgi.force_redirect = 0**.
4. Set **open_basedir** to point to the directory where the content of the website is located (typically **C:\inetpub\wwwroot**).
5. Uncomment and set **extension_dir** to point to the directory where PHP extensions reside (typically **extension_dir = "./ext"**).
6. Uncomment and set **error_log="php_errors.log"**. This is useful to deal with troubleshooting.
7. Uncomment every line that corresponds to a Windows extension DLL needed by PHP, as shown in Figure 7.

```
php.ini - Notepad                                                        —  □  ×
File  Edit  Format  View  Help
; default extension directory.
;
; Windows Extensions
; Note that ODBC support is built in, so no dll is needed for it.
; Note that many DLL files are located in the extensions/ (PHP 4) ext/ (PHP 5+)
; extension folders as well as the separate PECL DLL download (PHP 5+).
; Be sure to appropriately set the extension_dir directive.
;
;extension=php_bz2.dll
;extension=php_curl.dll
extension=php_fileinfo.dll
extension=php_ftp.dll
extension=php_gd2.dll
extension=php_gettext.dll
extension=php_gmp.dll
extension=php_intl.dll
extension=php_imap.dll
extension=php_interbase.dll
extension=php_ldap.dll
extension=php_mbstring.dll
extension=php_exif.dll       ; Must be after mbstring as it depends on it
extension=php_mysqli.dll
;extension=php_oci8_12c.dll  ; Use with Oracle Database 12c Instant Client
extension=php_openssl.dll
extension=php_pdo_firebird.dll
extension=php_pdo_mysql.dll
extension=php_pdo_oci.dll
extension=php_pdo_odbc.dll
extension=php_pdo_pgsql.dll
extension=php_pdo_sqlite.dll
extension=php_pgsql.dll
extension=php_shmop.dll

; The MIBS data available in the PHP distribution must be installed.
```

Figure 7: php.ini with Needed Extensions Uncommented

Now, we are going to save and close the **php.ini** file.

Adding PHP location to system path

We need to add the PHP location (commonly **C:\PHP**) to the system path variable in order to make the PHP engine available for execution. We should perform the following steps to accomplish this task.

First, right-click on the **This PC** icon located in the desktop. Then, click on the **Properties** option from the context menu displayed. The following dialog box will appear.

21
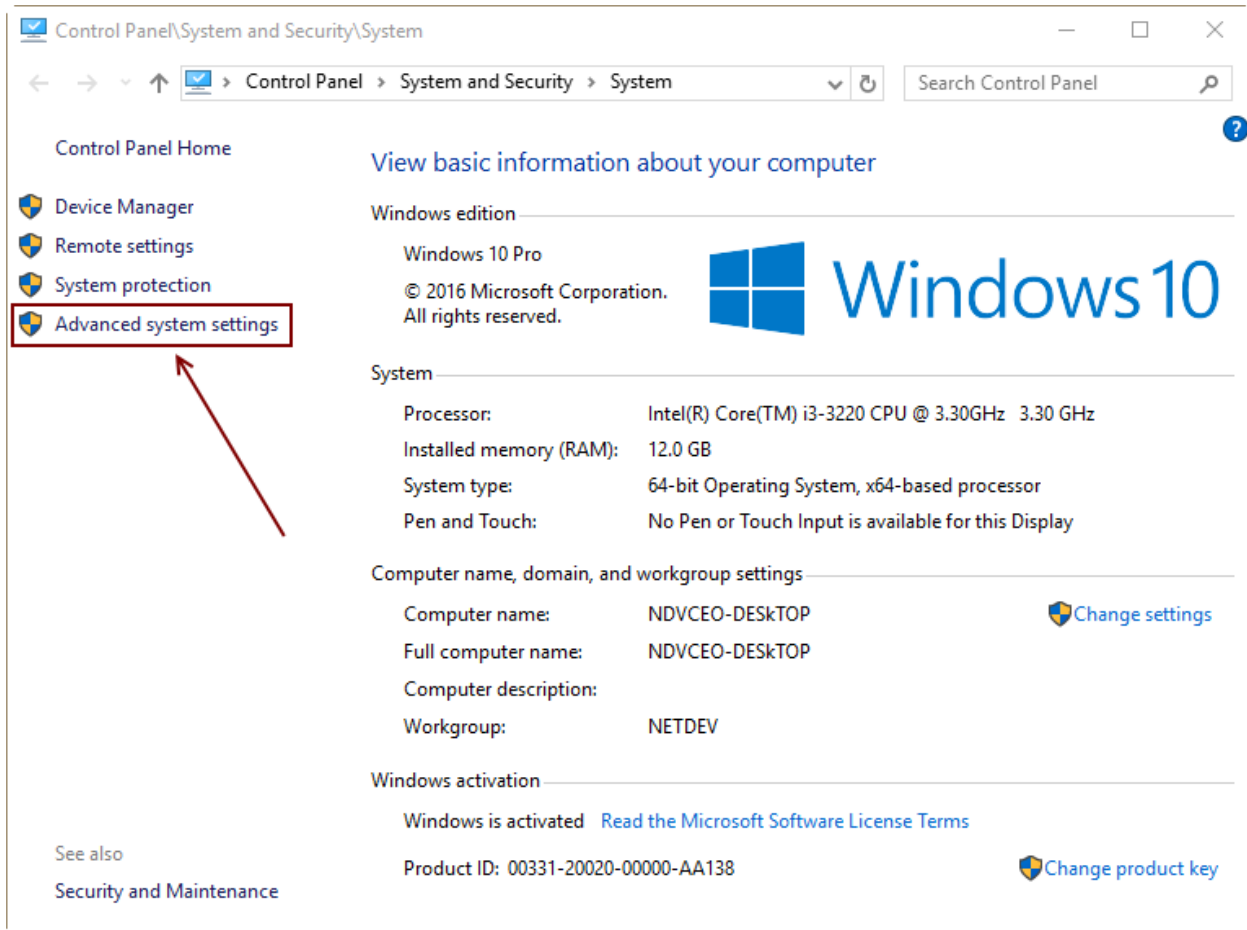
Figure 8: Advanced System Settings Link

Now, click on the **Advanced system settings** link, located in the left panel of the dialog box, and the **System Properties** dialog box will appear. Next, click the **Environment Variables** button.
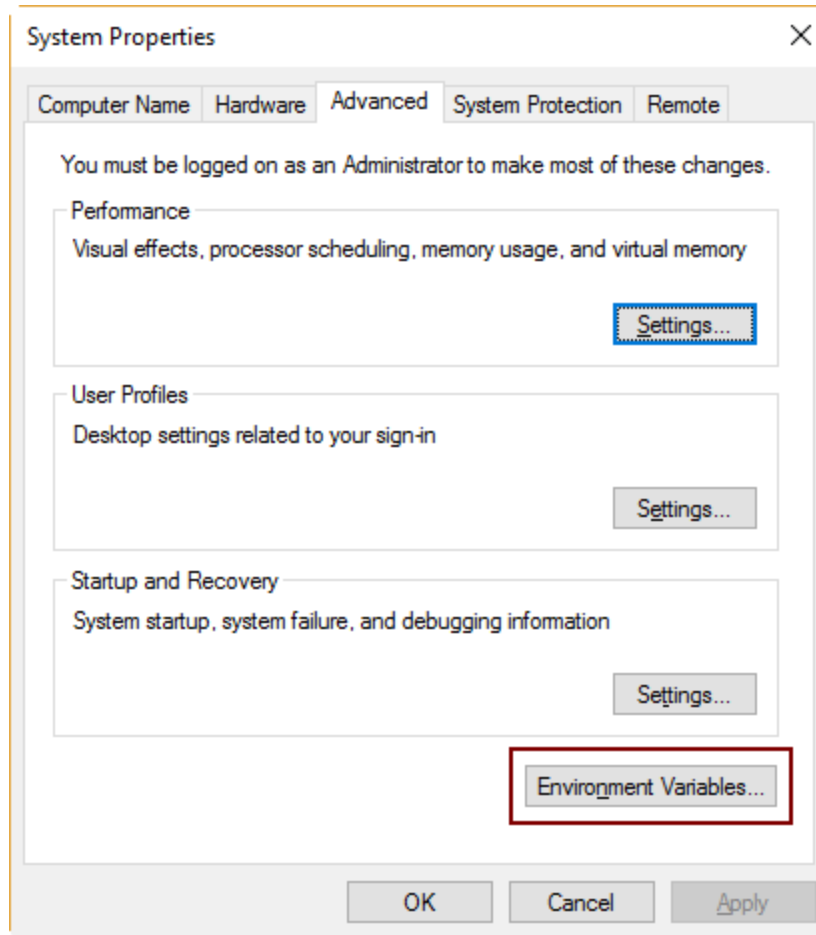
Figure 9: The Environment Variables Button

The **Environment Variables** dialog box will appear. Now, we need to select the **Path** variable within the **System Variables** section.
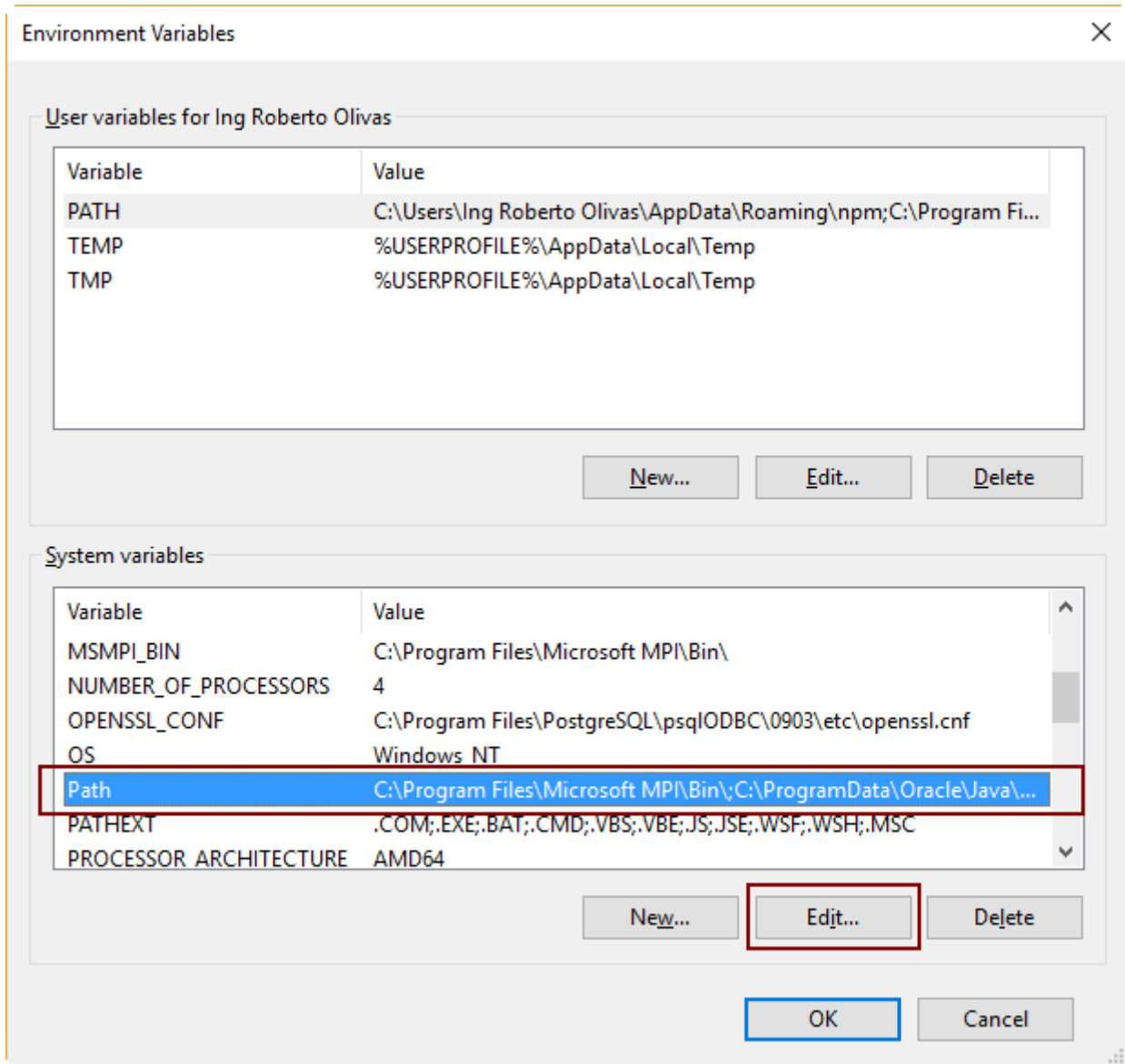
Figure 10: The Path System Variable in the Environment Variables Dialog Box

Next, click **Edit** to display the **Edit Environment Variable** dialog box.
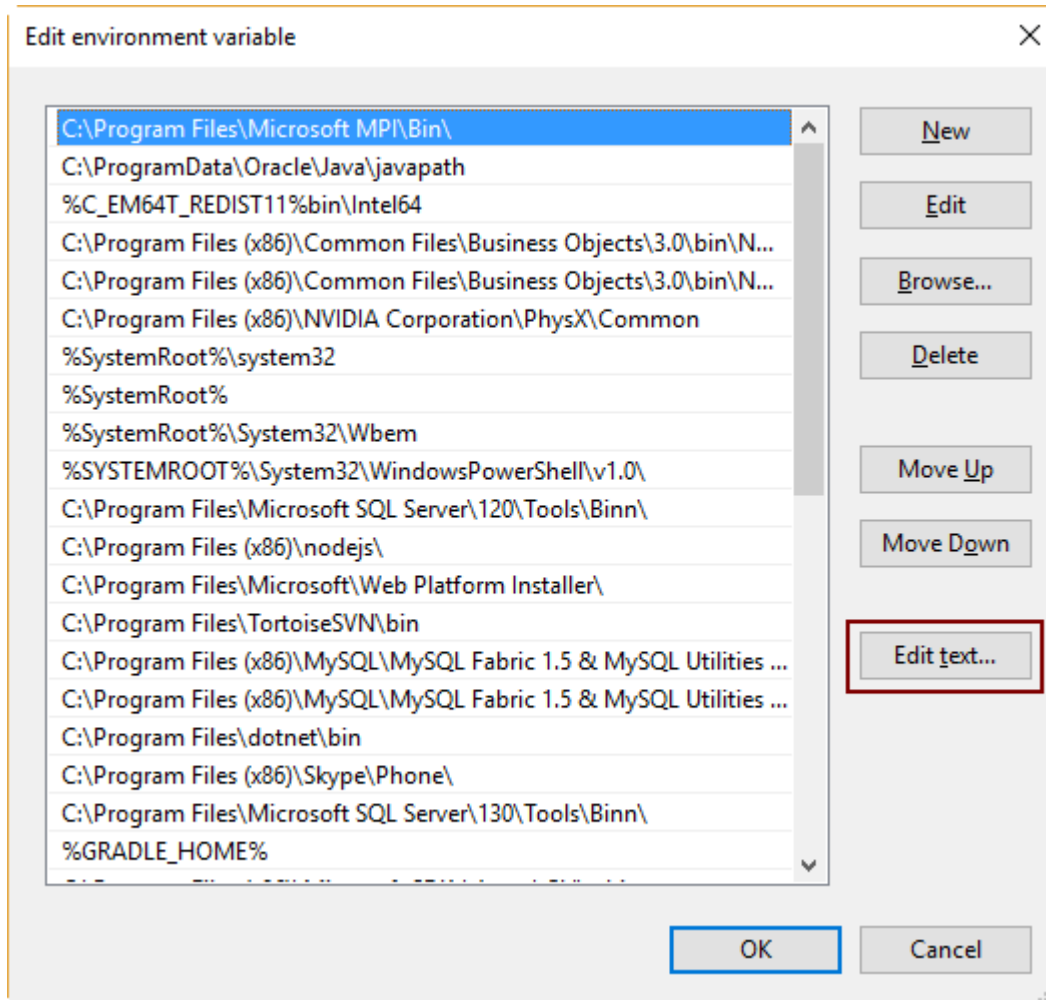
Figure 11: The Values of the Path System Variable in a List

As shown in Figure 11, all values for the Path system variable are displayed in a list that can be edited to add new variables or remove some of the values displayed. Click the **Edit text** button, which is available for modifying the values using a single line of text. Add the **C:\PHP** path leading with a semicolon at the end of the text line, as shown in the following figure.
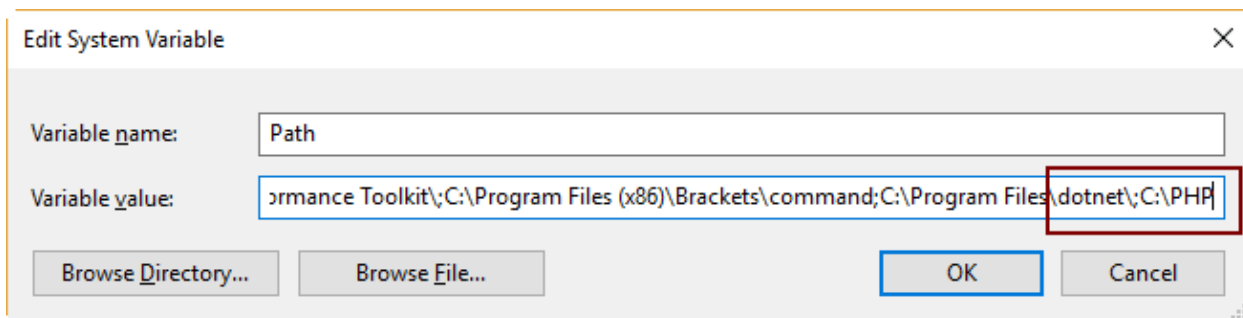
Figure 12: Adding C:\PHP Path to the Path System Variable

Now, we're going to click **OK** until we have exited the **System Properties** window.

Configuring PHP on IIS

Launch the **IIS Manager** from the **Windows Administrative Tools** section located in the **Start** menu.
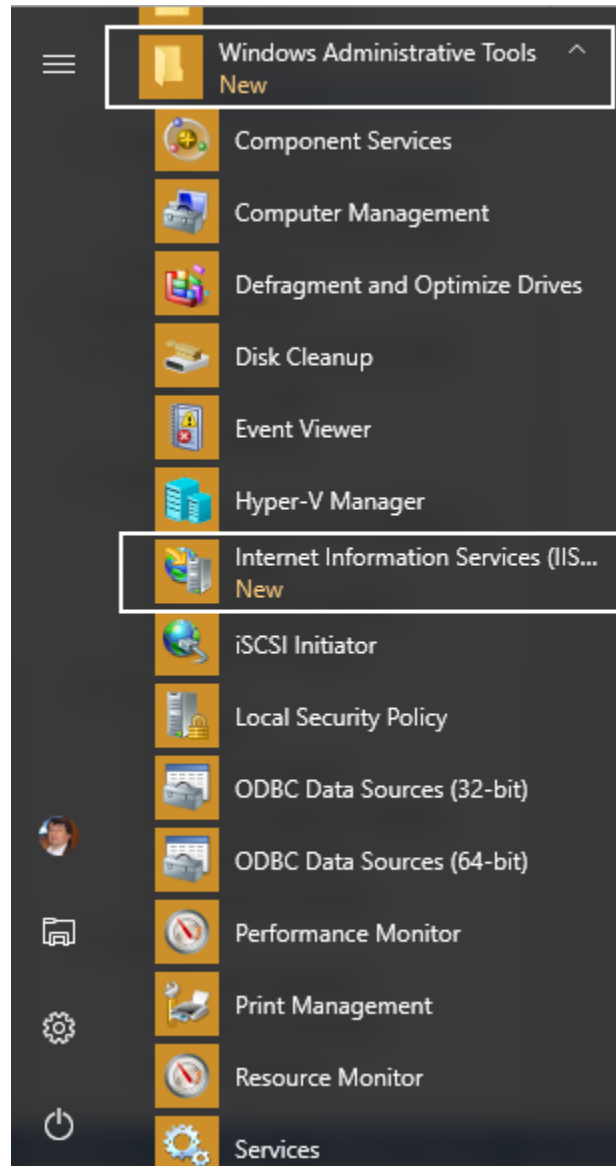
Figure 13: Windows Administrative Tools in the Start Menu

Now, when the **IIS Manager** window is displayed, click on the hostname that identifies the computer used as a server. This hostname is placed in the panel located at the left side of the window.

When the hostname is highlighted, double-click on the **Handler Mappings** icon, placed in the **IIS section** within the panel situated at the middle of the window, as shown in the following figure.
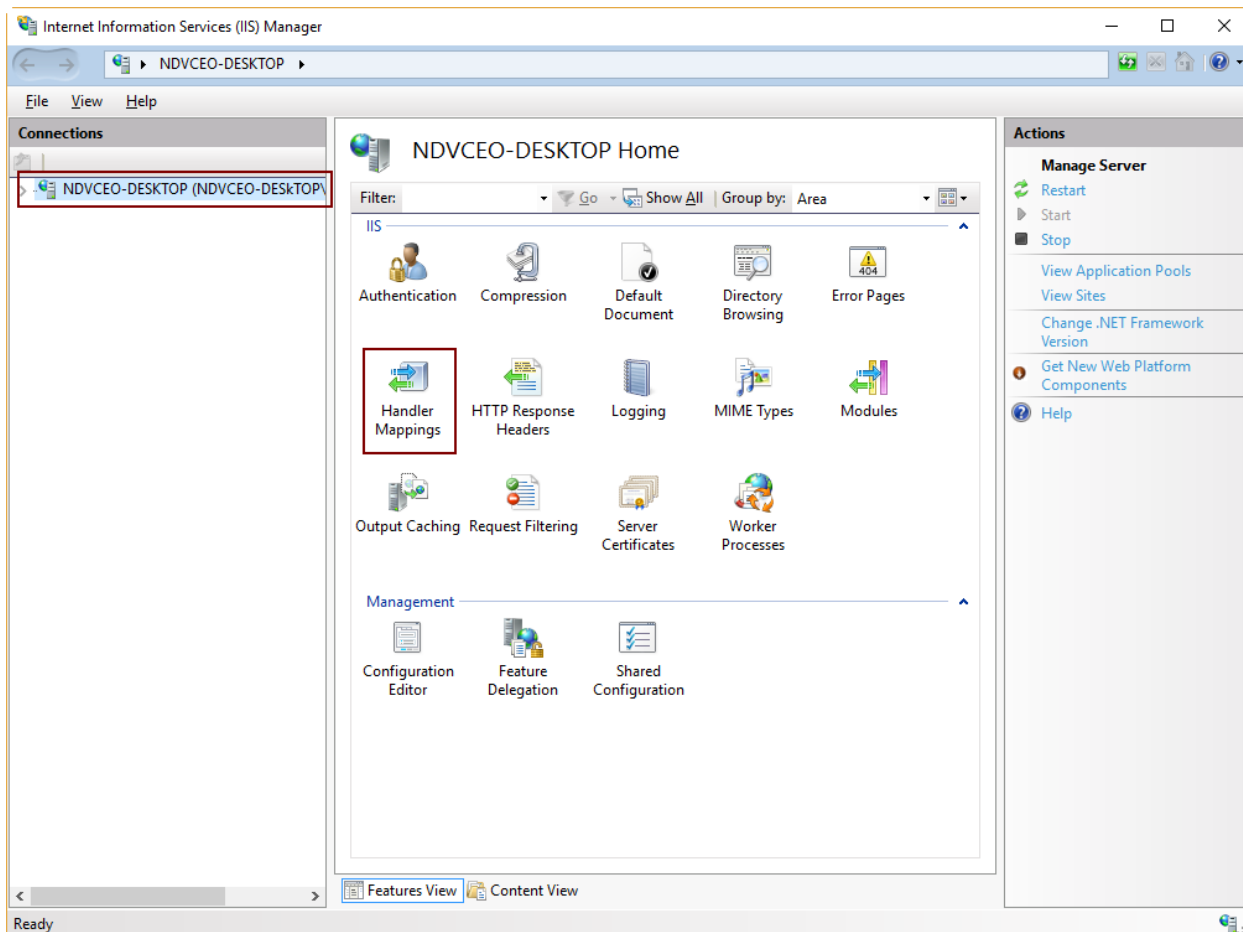
Figure 14: The Hostname and the Handler Mappings Button

The **Handler Mappings** action panel is displayed. Now, we should tell IIS which module is in charge of handling all PHP requests from the clients. In other words, we're going to establish which program or library will process all PHP code located in the server. To accomplish this task, click the **Add Module Mapping** link situated in the **Actions** panel, placed at the right of the window, as displayed in the following figure.
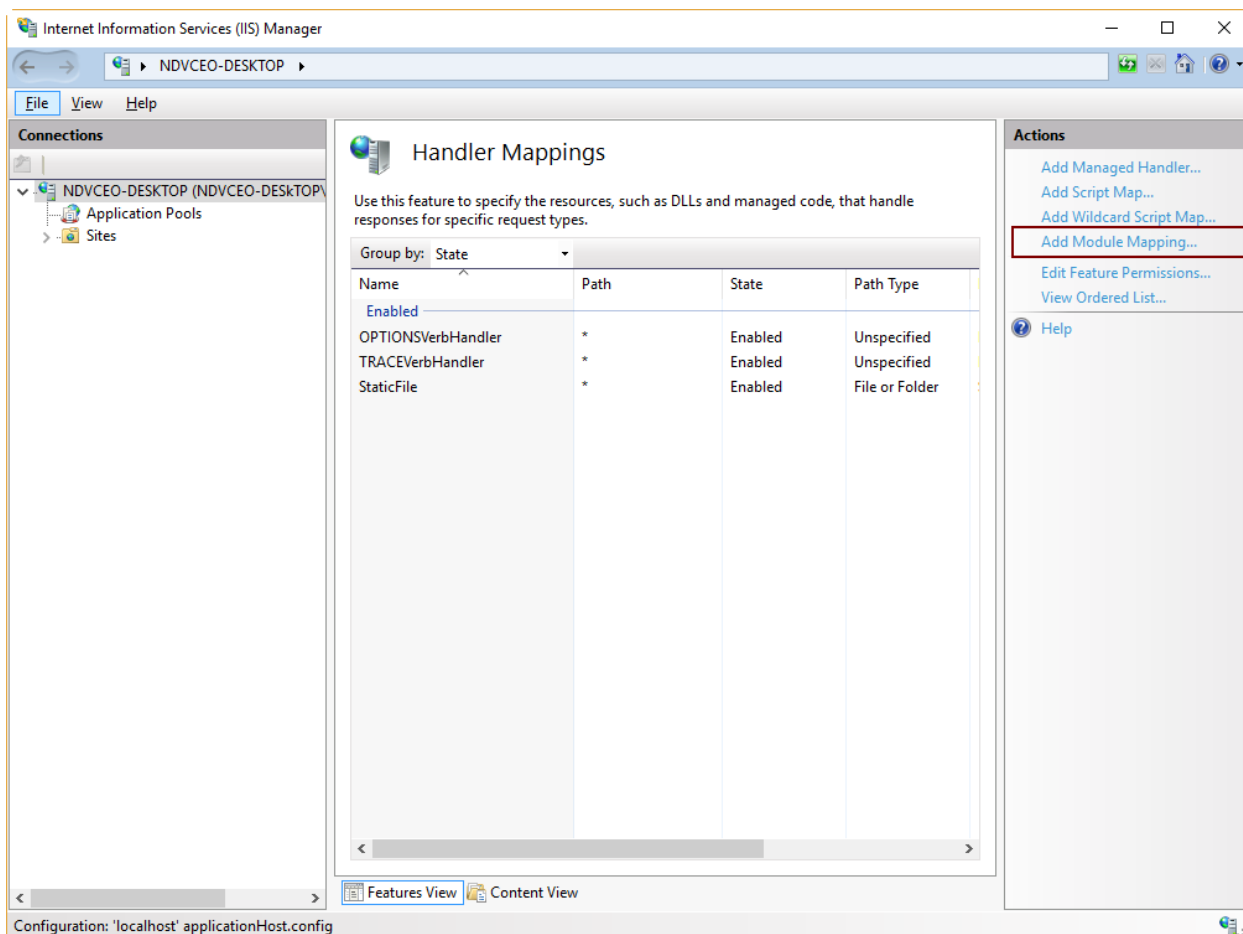
Figure 15: Handler Mappings Action Panel

Now, we're going to provide the following data:

- Request path: ***.php**
- Module: **FastCGImodule**
- Executable: **C:\PHP\php-cgi.exe**
- Name: **FastCGI**

With this data, we are telling to IIS that all requests ending with a **php** file extension will be handled by the **FastCGImodule** extension, using the **php-cgi.exe** program located in the PHP installation folder (**C:\PHP** in this case). The name FastCGI is used to identify the module mapping in the system. The following figure shows this data in the **Add Module Mapping** dialog box.

📝 ***Note: If the FastCGIModule entry is not in the Module dropdown control, that means you did not enable CGI in IIS Manager.***
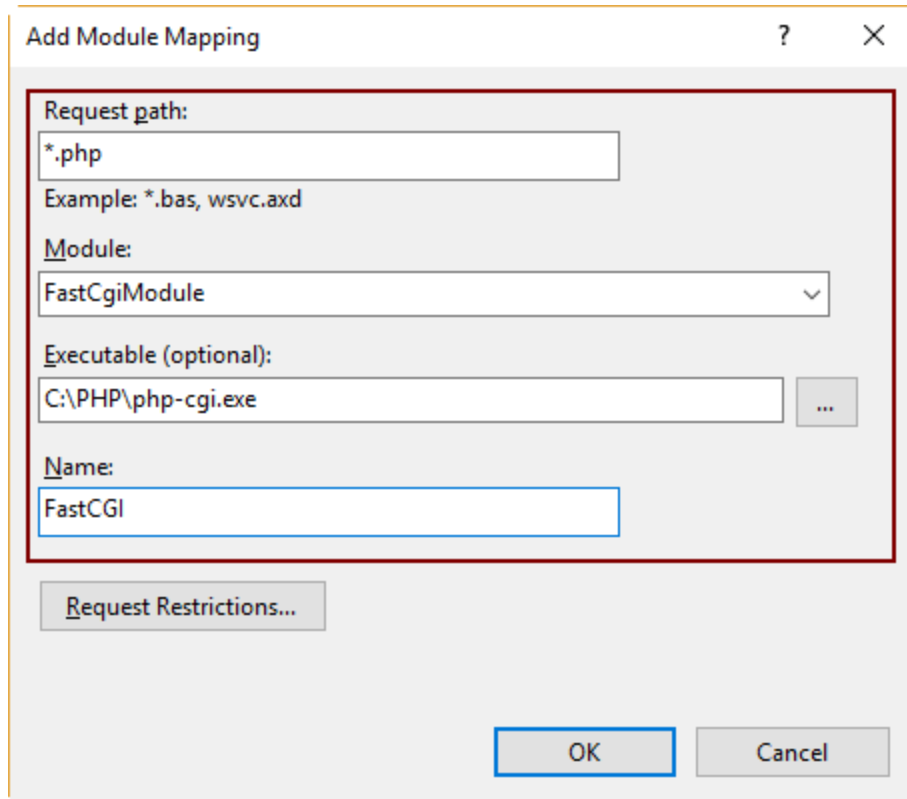
Figure 16: Adding PHP Module Mapping

Once the PHP module mapping is added to the system, we need to tell IIS which files will be treated as the default documents to process every time a web request is received. In other words, we're going to indicate which php files will be executed every time a user types **http://127.0.0.1** in the address bar of a web browser.

In order to do this, click on the computer's hostname located at the left side of the **IIS Manager** window. Then, double-click the **Default Document** icon, as displayed in the following figure.
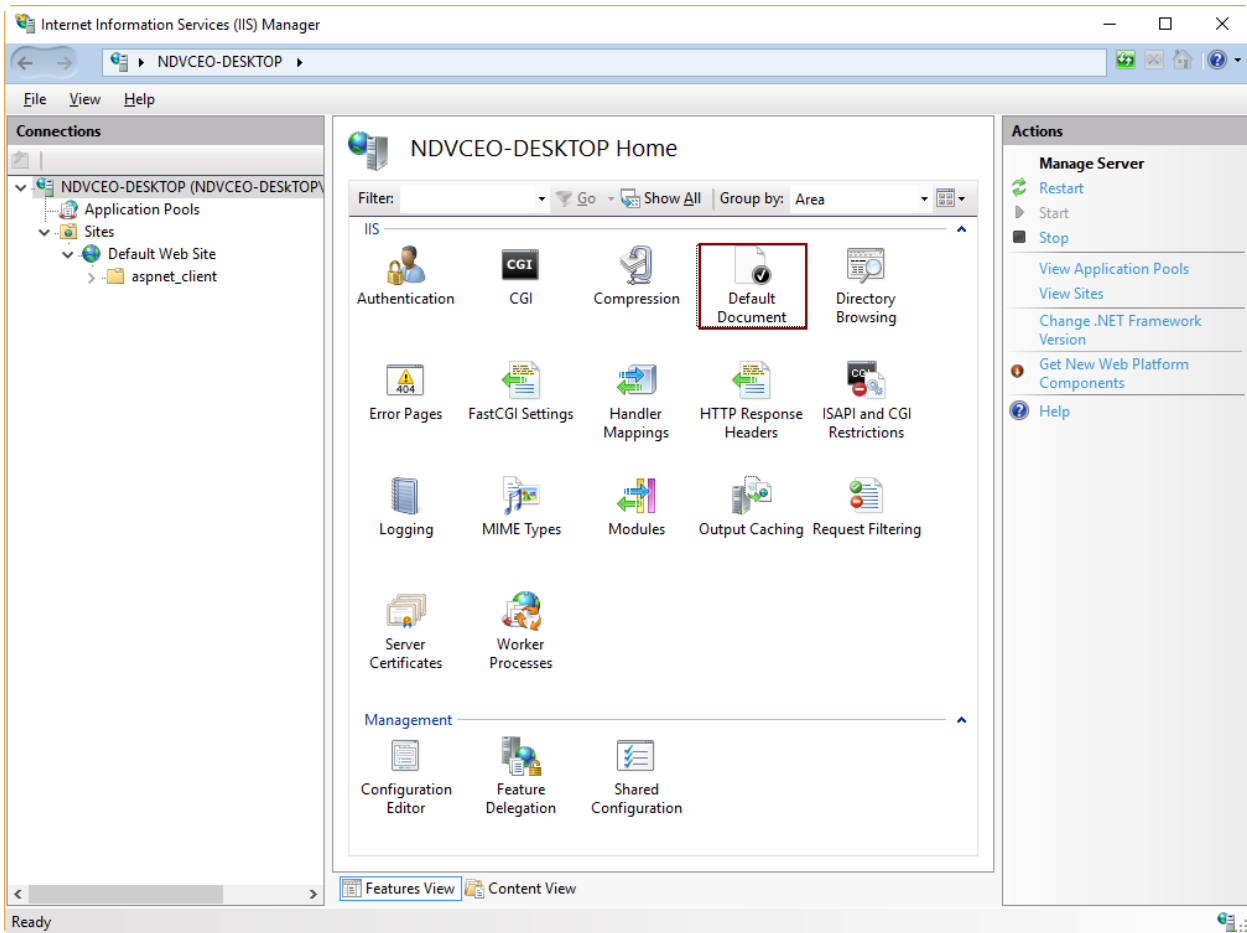
Figure 17: The Default Document Icon in the IIS Section

Now the **Default Document** actions panel will be displayed and all default document definitions will appear on the screen. To add a new default document, click on the **Add** link placed at the right of the window, as displayed in the following figure.
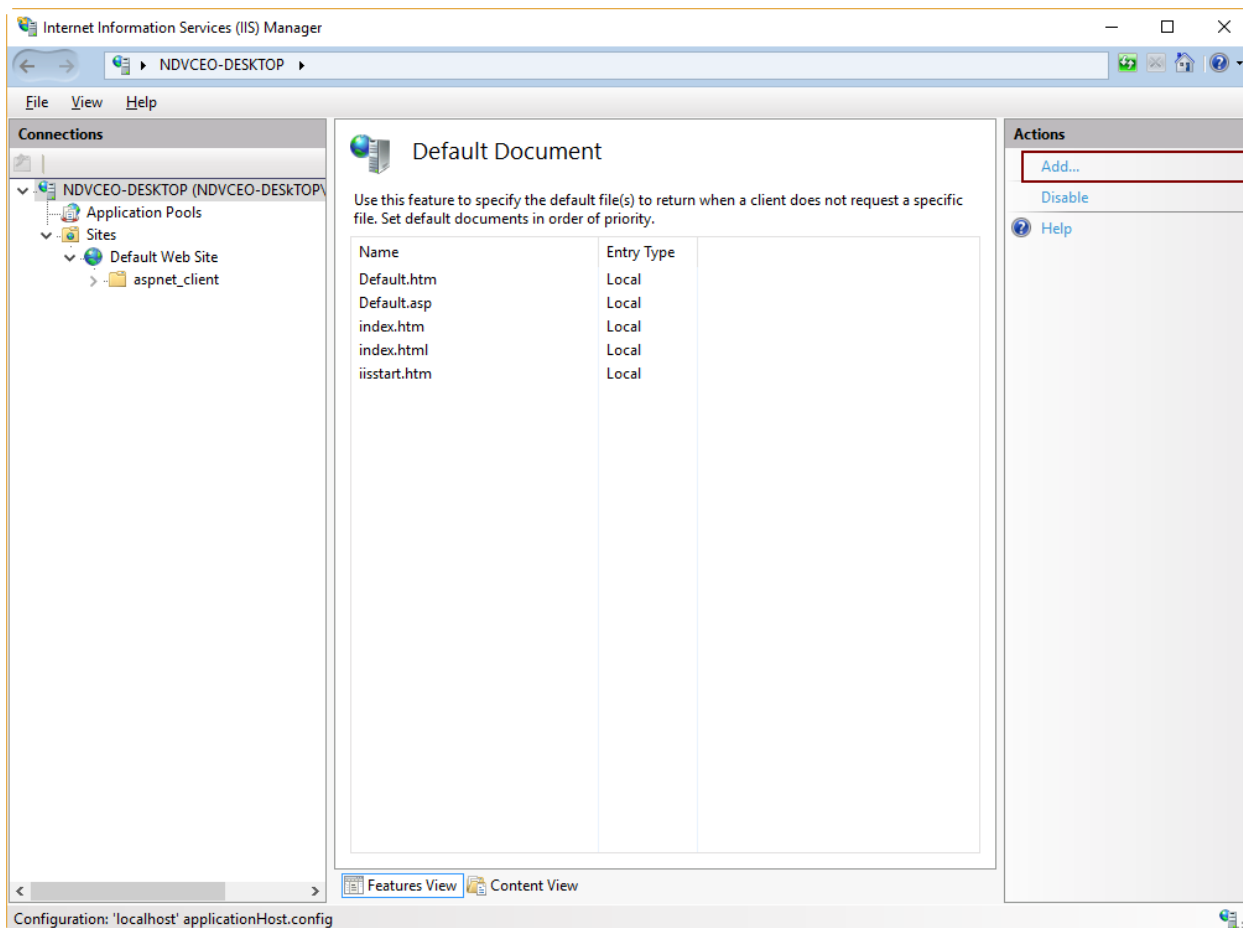
Figure 18: Default Document Actions Panel

Now, we're going to add **index.php** as a default document by entering the name in the **Add Default Document** dialog box, then clicking **OK**. The following figure shows this task.
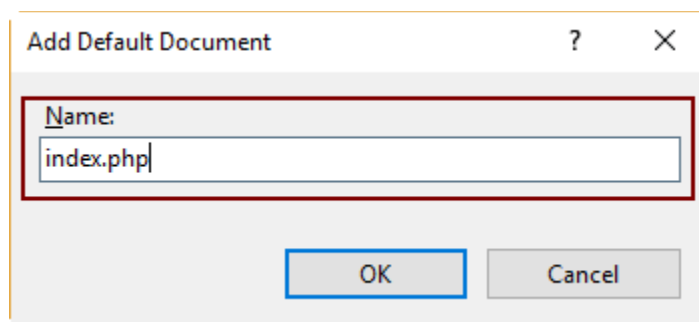
Figure 19: Adding a Default Document

We're going to repeat the process if we want to add **default.php** as a default document, too.

To apply all these changes, we need to restart IIS. To do this, click on the computer hostname located at the left of **IIS Manager** window. Then, click **Restart** on the right-hand side of the window.
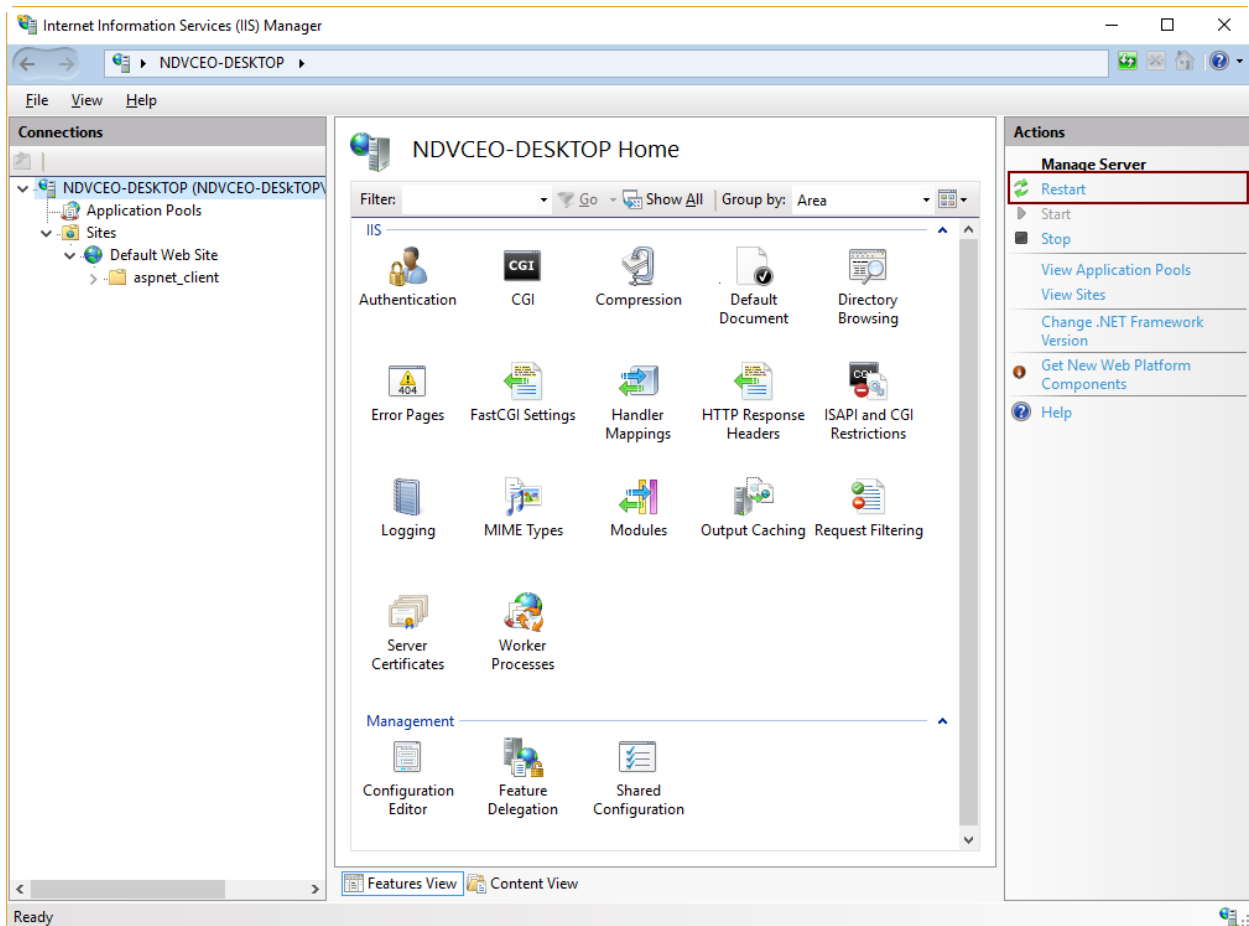
Figure 20: The Restart Link in the IIS Manager Window

Note: IIS searches default documents from top to bottom according to the list displayed in the Default Document actions panel. When one of these documents is found, this is executed and the search stops.

Testing the installation process

Now we should test the installation process. To accomplish this task, create a text file named **phpinfo.php** and save it into the website root folder (commonly **C:\inetpub\wwwroot**). The file should contain the following code.
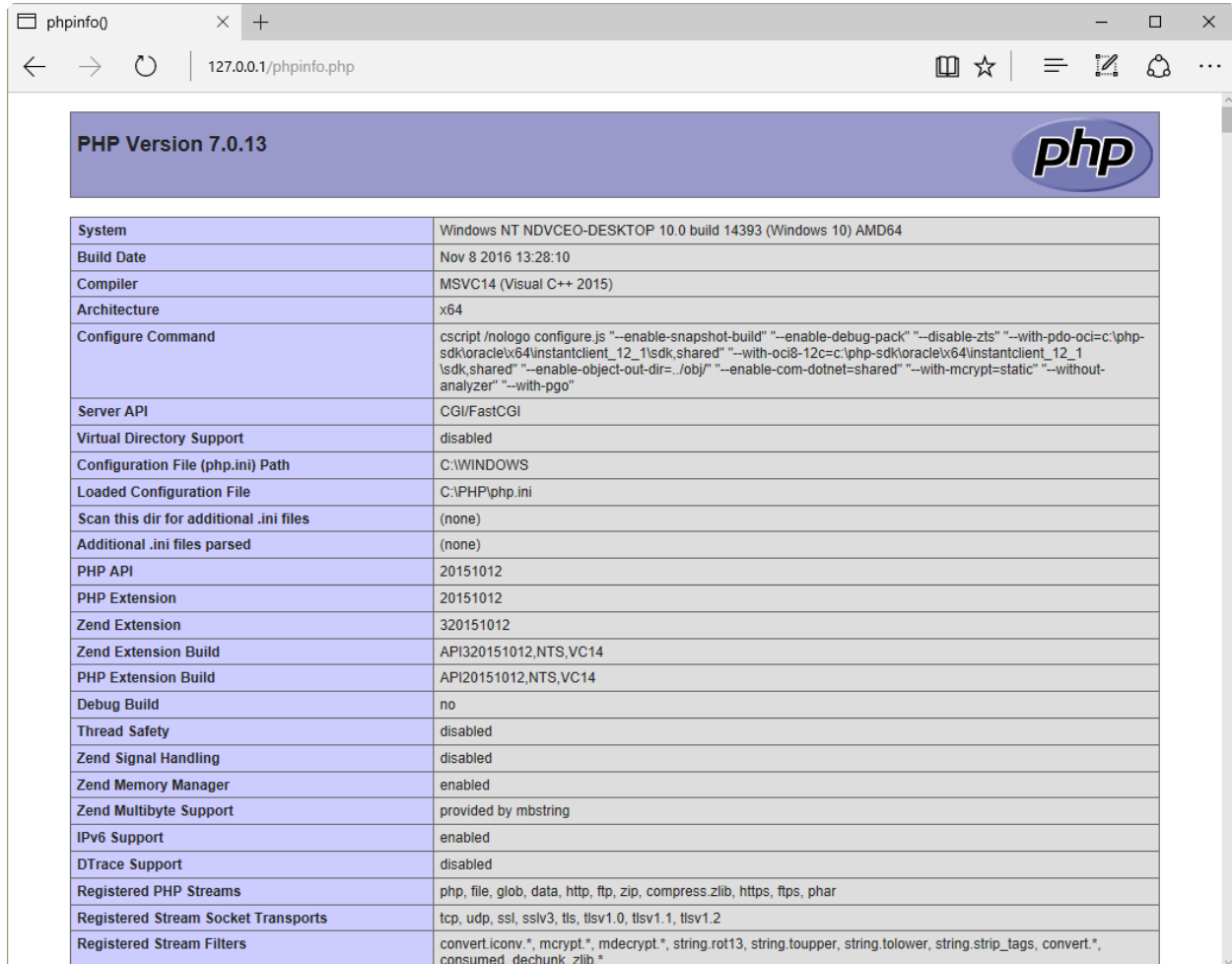
Code Listing 1: PHP Test Program

```php
<?php phpinfo(); ?>
```

Testing in the local computer

Launch an instance of Microsoft Edge, type **http://127.0.0.1/phpinfp.php** in the address bar, and press **Enter**. The result should be like the one displayed in the following figure.

Figure 21: Testing PHP Installation from the Local Computer

Testing from a remote computer

Launch an instance of a web browser in a computer connected to the same network where the IIS computer is plugged in. Then, assuming that the IIS computer IP address is 192.168.0.67, type **http://192.168.0.67/phpinfo.php** in the address bar and press **Enter**. In my case, I used a computer with Ubuntu Desktop as an operating system, and Mozilla Firefox as a web browser. The result looked like the one displayed in the following figure.
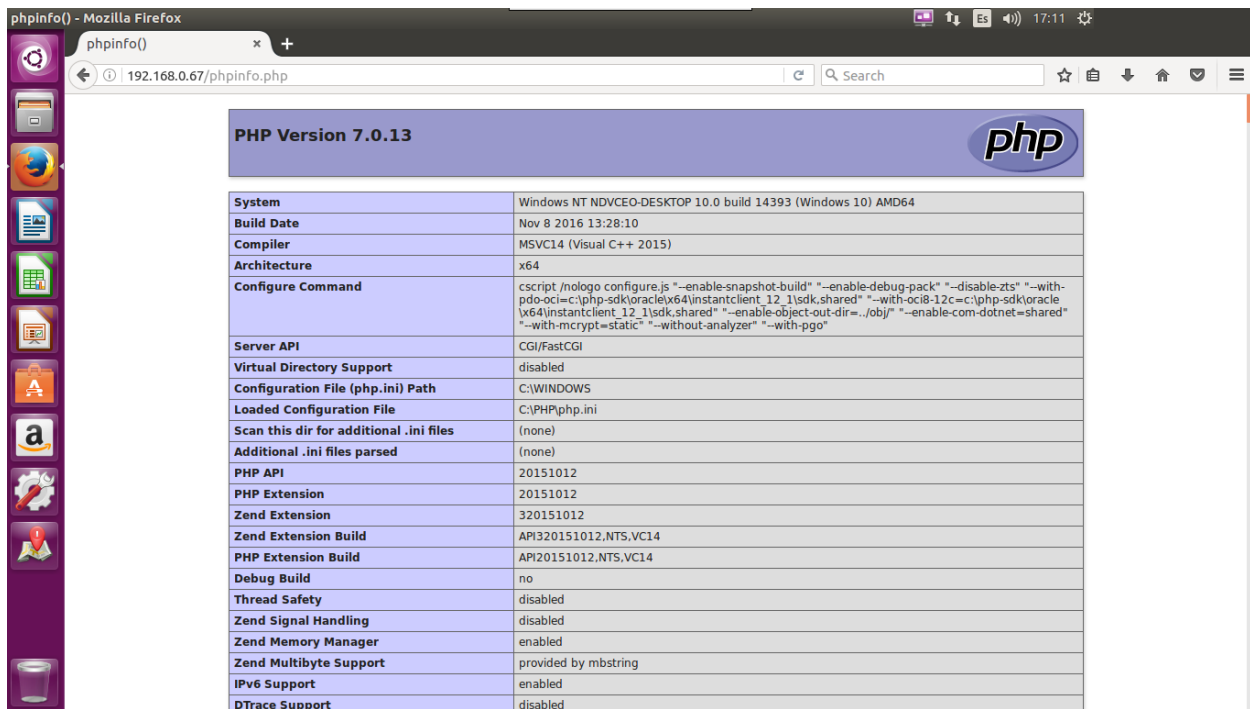
Figure 22: Testing PHP from a Remote Computer Running Ubuntu

Chapter summary

This chapter explained how to deploy PHP in a Windows environment using IIS (Internet Information Services) as a web server. To perform this deployment, the following requirements should be fulfilled:

- The computer should have a Windows operating system installed and running.
- IIS installed and configured.

To install IIS in a Windows 10 computer, you should go to the **Programs** section in the **Control Panel** and click **Turn Windows features on or off**. After that, you should click on the **Internet Information Services** checkbox in the **Windows Features** dialog box, in order to install IIS with the default features for a web server. The IIS installation process could be started by clicking the **OK** button.

Now, to install PHP on the computer, you should download a zip installation package from this location for a 32-bit system, or from this location for a 64-bit system. After the download is complete, you should unpack the zip file in a folder named **C:\PHP**.

To configure PHP, the file **php.ini-development** should be renamed to **php.ini**. Then, it should be edited in order to adjust some PHP working parameters to comply with IIS requirements.

As a final step, you should add **C:\PHP** to the Path system variable, and open the IIS Manager to set up PHP as the program that will handle all .php web requests coming from any client within the network.

Now, you should test the installation process by creating a text file named **phpinfo.php in** the website root folder (commonly **C:\inetpub\wwwroot**). The file should contain the following programming code: `<?php phpinfo(); ?>`. Then, you should type **http://127.0.0.1/phpinfo.php** from a web browser, and the PHP installation info should be displayed as a result.

Chapter 3 PHP Basics

Script: The basic concept of PHP

What is a script?

A script in PHP is a text file saved with the .php extension that contains pure PHP programming code, or PHP programming code embedded into HTML. The .php extension is necessary in the filename so that the file can be recognized by the PHP engine as a functional script.

Every PHP script has the following syntax.

Code Listing 2: PHP Scripts Syntax

```php
<?php
    /* PHP code goes here */
?>
```

We can embed PHP code into HTML. In this case, the syntax for the script is the same as the code displayed in the previous code listing, but it is inside HTML statements. The following code shows an example of PHP embedded into HTML.

Code Listing 3: PHP Embedded into HTML

```html
<html>
<head></head>
<body>
Hello, today is <?php echo date("l F jS \of Y"); ?>
</body>
</html>
```

Script samples

The ever-present Hello World

To say "Hello World" using PHP, we're going to create a file named **helloworld.php** and save it into our website root folder. The code for this file is displayed in the following sample.

Code Listing 4: Hello World Sample

```php
<?php
  echo 'Hello World from PHP';
?>
```

Now, if we enter **http://127.0.0.1/helloworld.php** in the address bar of our web browser, we should see something like the following figure.
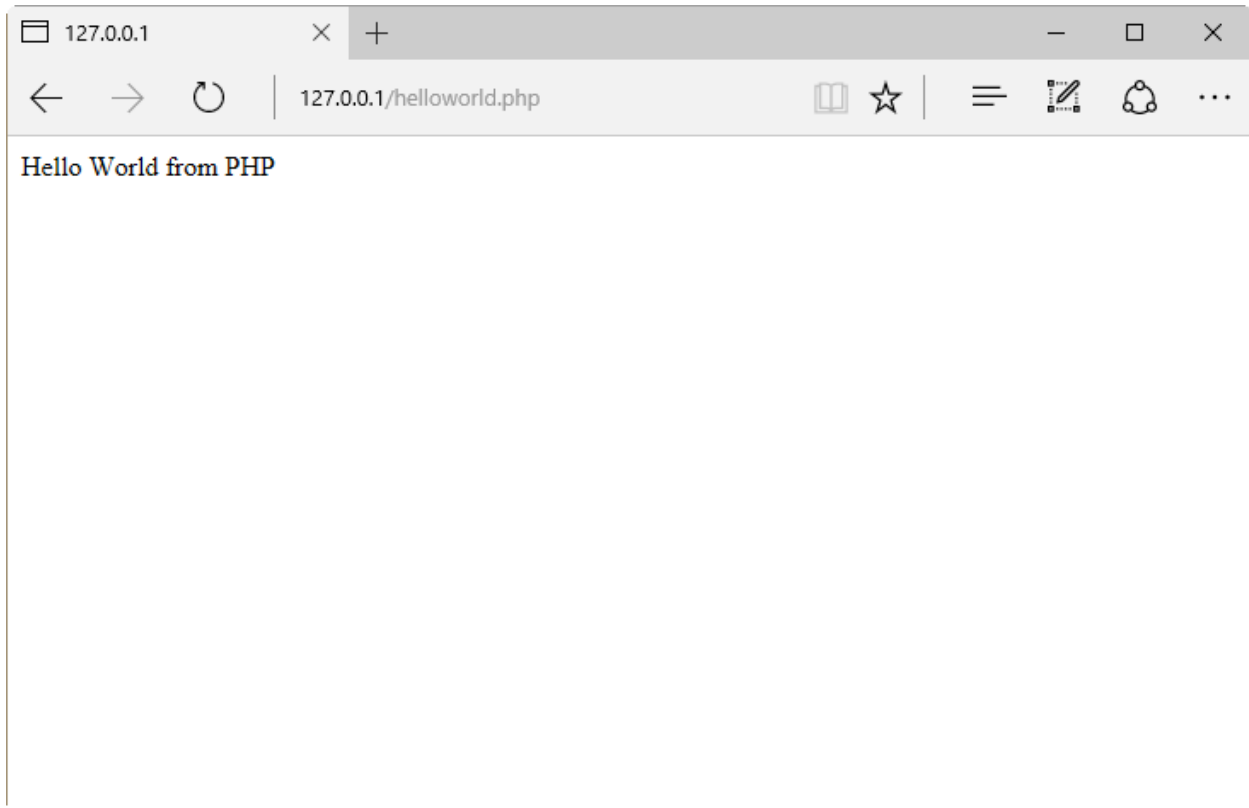
Figure 23: Hello World Sample Output

Displaying current date

The following sample displays the current date using pure PHP code. You should save the file as **currentdate.php**.

Code Listing 5: Displaying Current Date

```php
<?php
  echo 'Hello, today is ';
  echo date("l F jS \of Y");
?>
```

Again, typing **http://127.0.0.1/currentdate.php** in the address bar of the web browser, you should see something like the following figure.
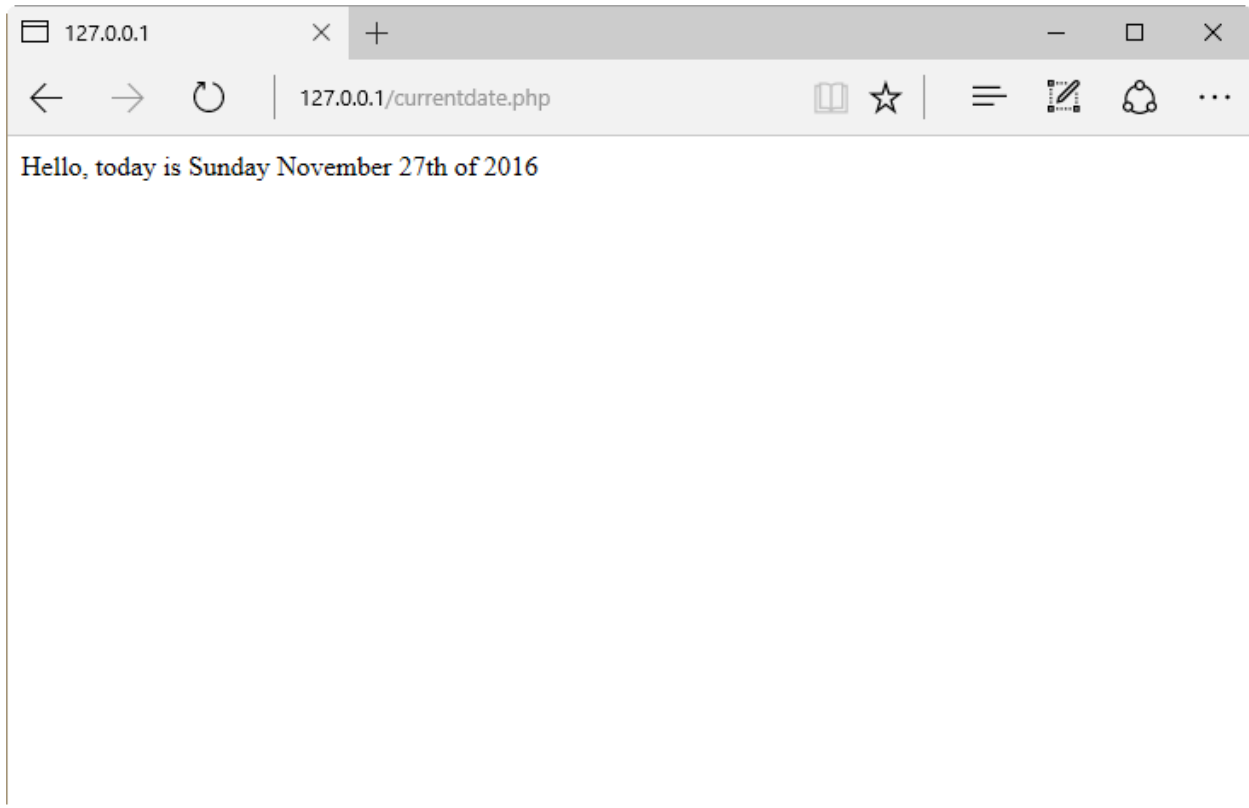
Figure 24: Output for Displaying Current Date Sample

Calling HTML from PHP sample

As explained in Chapter 1, PHP is able to call HTML code. The following code sample demonstrates this feature.

Code Listing 6: HTML Called from PHP

```php
<?php
  echo "<html>\n<head>\n<title>Calling HTML from
PHP</title>\n</head>\n<body>\n<h1>Hello, calling HTML from
PHP!</h1>\n</body>\n</html>";
?>
```

At this point, the **echo** statement has appeared in all the samples. This statement sends the content of the string placed beside it to the standard output device. For a web server, the standard output device sends the response for a request to the calling client (usually a web browser).

In the previous sample, the **echo** statement sends to the calling client the HTML code necessary to display **"Hello, calling HTML from PHP!"** in a web browser. To test this sample, we should save the code in a file named **callinghtml.php**. Then, from the address bar of the web browser, we will type **http://127.0.0.1/callinghtml.php**, and the output displayed will look like the following figure.
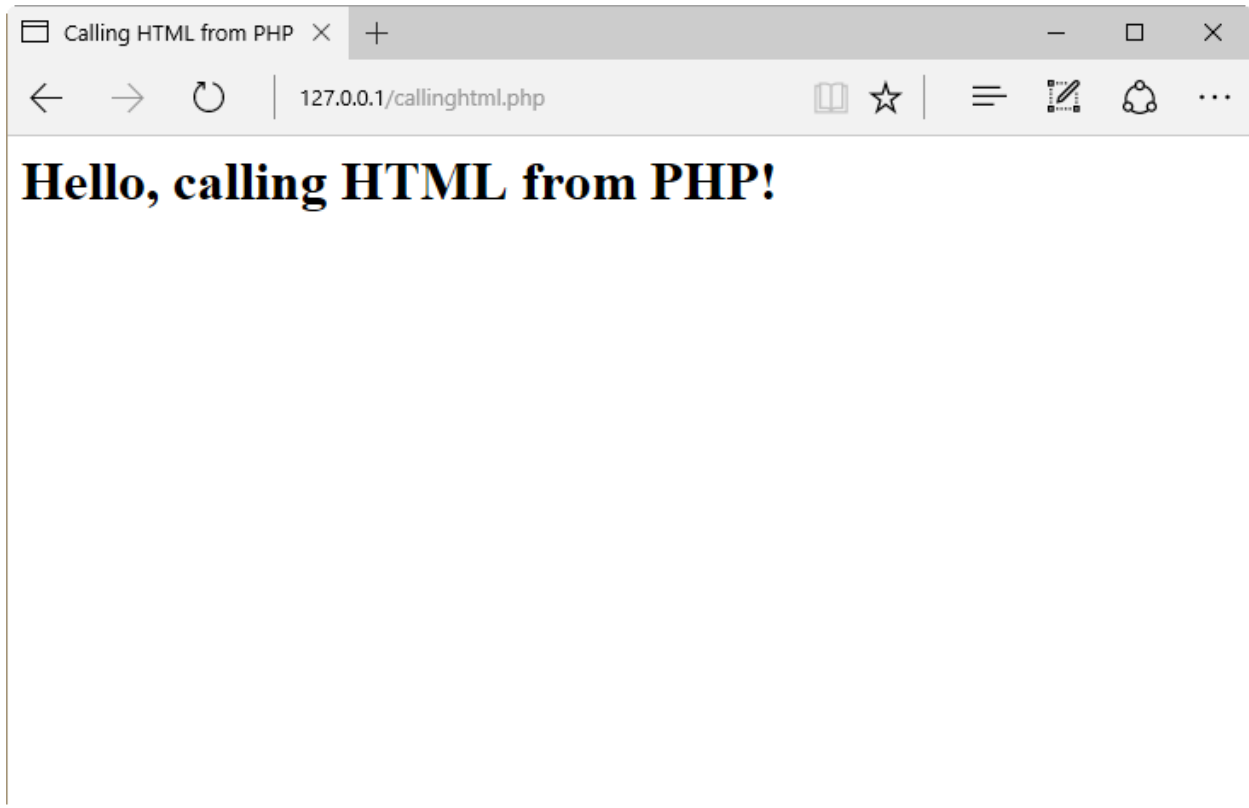
Figure 25: Calling HTML from a PHP Script

If we want to know which response was sent to the browser, we should press the **F12** key to get the following code in the DOM Explorer.

Code Listing 7: PHP Response Sent to the Browser

```html
<html>
<head>
<title>Calling HTML from PHP</title>
</head>
<body>
<h1>Hello, calling HTML from PHP!</h1>
</body>
</html>
```

The DOM Explorer is displayed in the following figure.
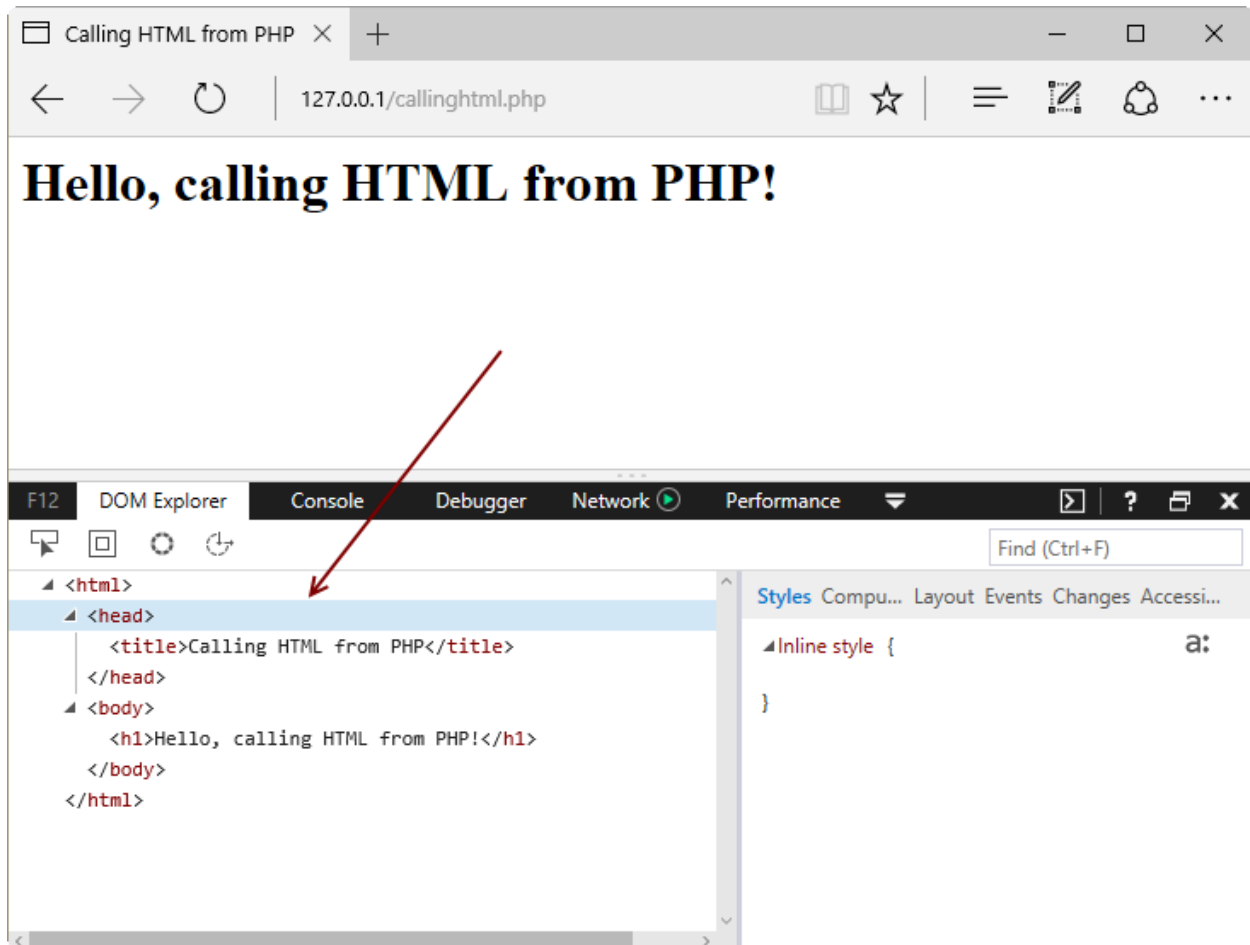
Figure 26: The DOM Explorer Displaying the PHP Response

As shown in Figure 26, the sample PHP script sent pure HTML code as a response for the request made by the web browser.

Variables

As in most programming languages, the main way to store data in a PHP program is by using variables. A variable is an identifier that can hold data dynamically, meaning that data stored in variables can change according to the program needs during the execution flow.

Declaring and using variables in PHP

Variable declaration, and the use of these variables in PHP, should comply with the following requisites.

- All variable names are denoted with a leading dollar sign (**$**).
- Variable names must begin with a letter or underscore character.
- Characters like +, -, %, (,), ., and & cannot be employed.

- Variables can be declared before assignment, but this is not absolutely necessary.
- Variables do not have intrinsic types; a variable cannot know in advance whether it will be used to store a number or a string.
- Converting variables from one type to another is performed automatically.
- Variable assignment is performed with the = operator, placing the variable on the left side and the expression to be evaluated on the right.
- The value of a variable is the value of its most recent assignment.

Variable types

The following table summarizes the data types available in PHP.

Table 2: PHP Data Types

Data Type	Description
Integer	Whole numbers with no decimal point
Double	Floating point numbers such as 1.31313 or 34.5
Boolean	A type with two possible values, either true or false
NULL	The NULL value
String	Sequences of characters like 'Hello World' or 'Last Name'
Array	A named and indexed collection of values
Object	Instances of programmer-defined classes that package both attributes (values) and methods (functions), specific to the class
Resource	Special data types that hold references to resources external to PHP, such as database connections

The following code sample shows a series of declared variables, each storing a value that corresponds to one the data types described in the previous table.

Code Listing 8: Declaring Variables in PHP

```php
$var_double = 3 + 0.14159;
$var_integer = 4;
$var_string = "This is a PHP variable";
$var_array = array("An array element","Other element");
$var_boolean = TRUE;
$var_null = NULL;
```

Variable scopes

The scope of a variable can be defined as the range of availability that variable has, starting from the program in which the variable is declared. So, in this context, we can have the following kind of scopes.

- Local variable – A variable that can be referenced in the declaring program only. Once this program finishes its execution, all local variables are destroyed.
- Global variable – A variable that can be accessed in any part of the executed thread, starting at the program in which the variable is declared. In order for PHP to recognize a variable as global, the prefix **GLOBAL** must be declared before the name of the variable. All global variables are destroyed when the executed thread finishes.
- Function parameter – A variable declared after a function name and inside parentheses. The scope of function parameters is the function itself.
- Static variable – A variable declared inside a function, with the word **STATIC** before its name. Unlike function parameters, a static variable keeps its value when the function exits, and that value will be held when the function is called again.

The following sample code shows how the different scopes work.

Code Listing 9: Variable Scopes

```php
<?php
//Local variables
$samplevar = 10;
function sumvars() {
      $a = 5;
      $b = 3;
       $samplevar = $a + $b;    //$samplevar is local variable inside this
function
      echo "\$samplevar inside this function is $samplevar. <br />";
}
sumvars();
echo "\$samplevar outside the previous function is $samplevar. <br />";
//Function parameters
function phpfunction($parameter1,$parameter2)
{
      return ($parameter1 * $parameter2);
}
$funcval = phpfunction(6,3);
echo "Return value from phpfunction() is $funcval <br />";

//Global variables
$globalvar = 55;

function dividevalue() {
      GLOBAL $globalvar;
      $globalvar/= 11;
      echo "Division result $globalvar <br />";
}
```

```php
dividevalue();

//Static variables
function countingsheeps()
{
    STATIC $sheepnumber = 0;
    $sheepnumber++;
    echo "Sheep number $sheepnumber <br />";
}

countingsheeps();
countingsheeps();
countingsheeps();
countingsheeps();
countingsheeps();

?>
```

If we copy the previous code in a file named **varscopesample.php** in the website root folder (**C:\Inetpub\wwwroot**), and type **http://127.0.0.1/varscopesamle.php** in the address bar of the web browser, the following output should be displayed.
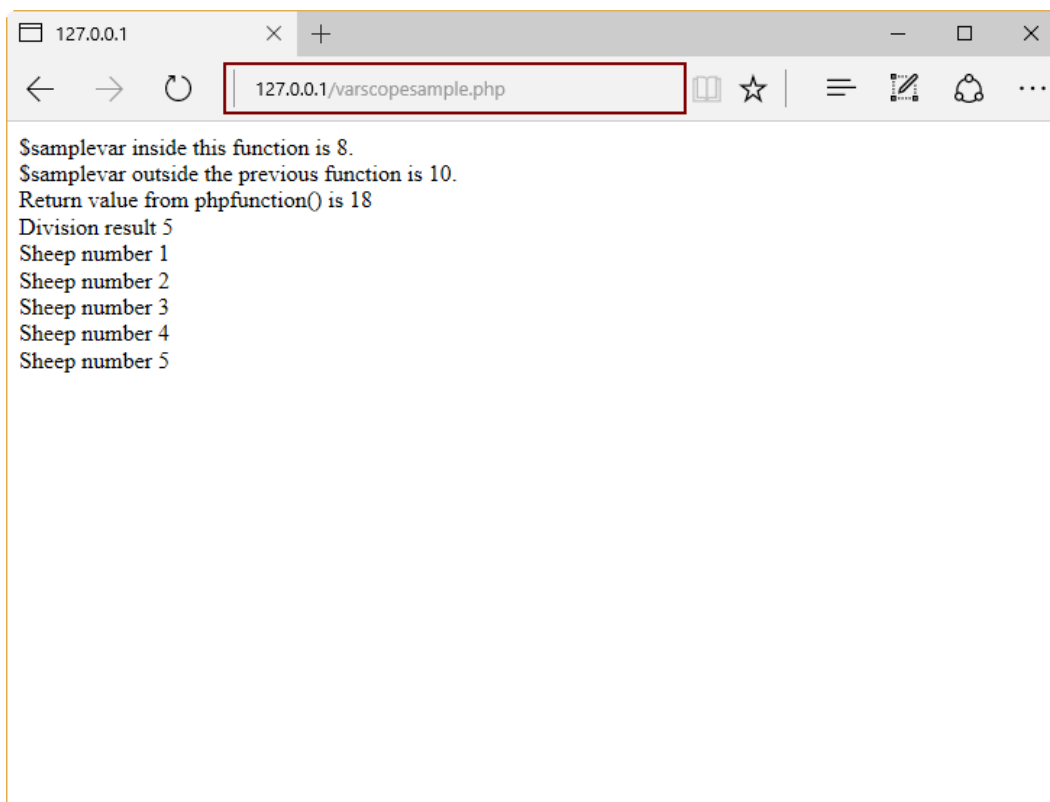
Figure 27: Variables Scope Sample Results

Predefined variables

There is a series of variables that are available to any script running. Also, PHP provides a set of predefined arrays containing variables from the environment, the web server, and user input. These arrays are called *superglobals*. The following table summarizes the superglobals.

Table 3: PHP Superglobals Summary

PHP Superglobals	
$GLOBALS	This array contains a reference to every global variable in the script. The name of every global variable is a key of this array.
$_SERVER	This array contains information about the web server environment, such as headers, paths, and script locations. Since these values are created by the web server being used, some of them could not exist in some environments.
$_GET	This array contains associations to all variables passed to the script via the HTTP **GET** method.
$_POST	This array contains associations to all variables passed to the script via the HTTP **POST** method.
$_FILES	This array contains associations to all items uploaded to the script via the HTTP **POST** method.
$_COOKIE	This array contains associations to all variables passed to the script via HTTP cookies.
$_REQUEST	This array contains associations to the contents of **$_GET**, **$_POST**, and **$_COOKIE**.
$_SESSION	This array contains associations to all session variables available to the script.
$_PHP_SELF	A string containing the script file name in which this variable is used.
$php_errormsg	A string variable containing the last error message generated by PHP. The **$php_errormsg** variable is not a true superglobal object, but it's closely related.

The following code shows some examples of predefined variables.

Code Listing 10: Predefined Variables Sample

```php
<?php
  /* $GLOBALS example*/
  $apptitle = "Application title"; //This is a global variable
  function locals()
```

```php
  {
        $apptitle = "Local application title";
        echo "\$apptitle value at global scope: " . $GLOBALS["apptitle"]
. "<br />";
        echo "\$apptitle value at local scope: " . $apptitle . "<br />";
  }

  locals();

  /* $_SERVER example */
  echo $_SERVER['PHP_SELF'] . "<br />"; //Script filename relative to the
website root
  echo $_SERVER['SERVER_NAME'] . "<br />"; //Server name or Server IP
Address
  echo $_SERVER['REMOTE_ADDR'] . "<br />"; //The IP address of the client
computer
  echo $_SERVER['REMOTE_HOST'] . "<br />"; //The host name or IP address
of the client computer

?>
```

Note: A detailed sample for all superglobals is beyond the scope of this book.

Assuming the previous sample is saved in a file named **predefinedvariables.php**, if we type **http://127.0.0.1/predefinedvariables.php** into the address bar of the web browser, we should get the following output.
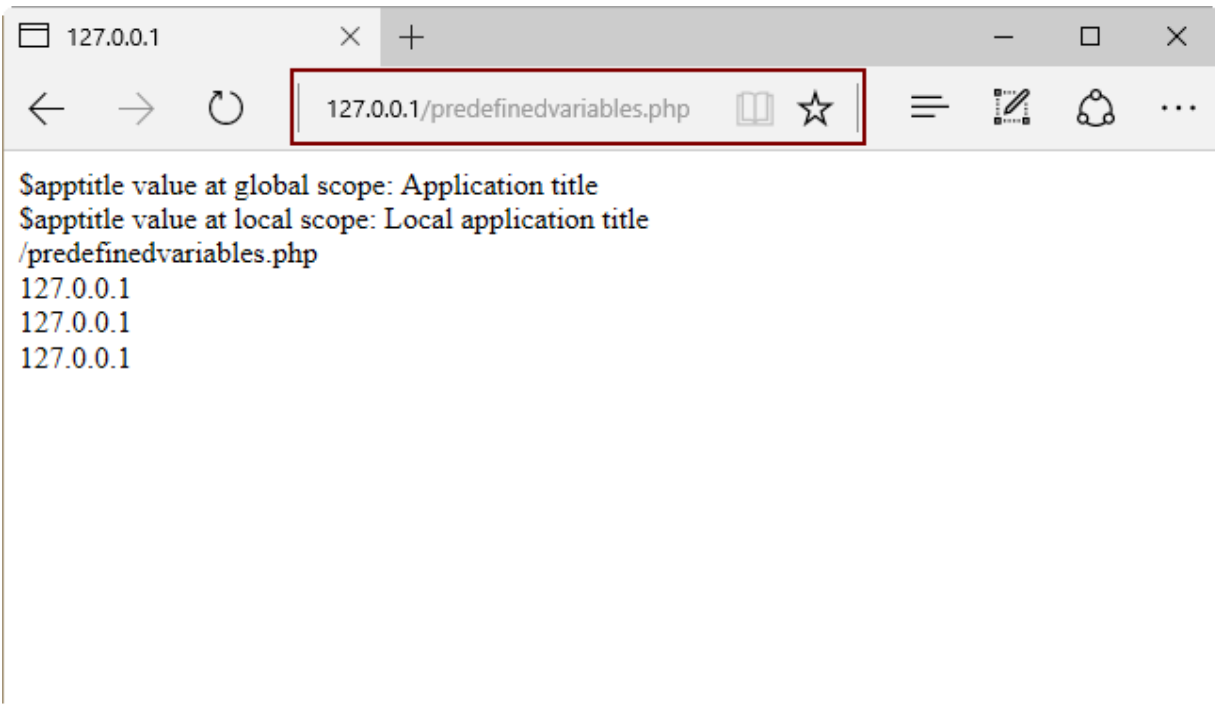
Figure 28: Superglobals Sample Output

Constants

A constant is an identifier that holds a simple value. This value cannot change during the execution of the script.

Naming constants

Constants are case-sensitive by default. A good practice in PHP dictates that constant names should always be uppercase. Constant names can start with a letter or underscores.

Defining constants

In PHP, all constants are defined by the `define()` function. To retrieve the value of a constant, we have to specify its name. We can also use the `constant()` function to read a constant value. Unlike with variables, we don't need to use the dollar (`$`) sign at the beginning of the constant name.

The following code sample shows how to define and read constant values.

Code Listing 11: Constants Sample

```php
<?php
```

```
   define("WEBPAGEWIDTH", 100);

   echo WEBPAGEWIDTH; //Retrieving constant value using its name directly
   echo constant("WEBPAGEWIDTH"); //Retrieving constant value with
constant() function
?>
```

Operators

An operator is a symbol that is used to perform a process into an expression. This process is also known as an operation.

Code Listing 12: Operators Sample

```
$total = $subtotal + $tax;
```

In this code sample, **$subtotal + $tax** is an expression. The process to be performed in this expression is to add the value of the **$subtotal** variable to the value of the **$tax** variable. This process is represented by the plus (**+**) sign, and this sign is called an operator. The $**subtotal** and $**tax** variables are called operands.

PHP supports some types of operators, which are explained in the following sections.

Arithmetic operators

The following table summarizes the use of arithmetic operators, which are used to perform arithmetic operations.

Table 4: PHP Arithmetic Operators Summary

Operator	Description
+	Adds the values of two operands
-	Subtracts the value of the second operant from the first one
*****	Multiplies two operands
/	Divides a numerator operand (placed at the left) by a de-numerator operand (placed at the right)
%	Gets the remainder of an integer division between two operands
++	Increases the value of an integer operator by 1
--	Decreases the value of an integer operator by 1

Note: Technically, ++ and -- can also be considered assignment operators.

Comparison operators

As suggested by their type name, comparison operators are used to check whether or not a criteria between two operands is met. If the operands adhere to the checked criteria, the result returned by the comparison operation will be the **true** Boolean value. Otherwise, a **false** Boolean value will be returned. These operators are summarized in the following table.

Table 5: PHP Comparison Operators Summary

Operator	Description
`==`	Checks if the value of two operands are equal
`!=`	Checks if the value of the operand placed at the left is not equal to the value of the operand placed at the right
`>`	Checks if the value of the operand placed at the left is greater than the value of the operand placed at the right
`<`	Checks if the value of the operand placed at the left is less than the value of the operand placed at the right
`>=`	Checks if the value of the operand placed at the left is greater than or equal to the value of the operand placed at the right
`<=`	Checks if the value of the operand placed at the left is less than or equal to the value of the operand placed at the right

Logical operators

These operators are used to perform logical operations between two operands. A logical operation is a process that returns either **true** or **false**, depending on the logical state (true or false) of two operands. These operators are summarized in the following table.

Table 6: Logical Operators Summary

Operator	Description
`and`	Logical **AND**. Returns **true** if both operands are true.
`or`	Logical **OR**. Returns **true** if any of the two operands is true.
`xor`	Returns **true** if any of the two operands are true, but not both.

Operator	Description
&&	Logical **AND**. Returns **true** if both operands are true.
||	Logical **OR**. Returns **true** if any of the two operands is true.
!	Logical **NOT**. Reverses the logical state of its operand; if the operand is true, it becomes **false**, and vice versa.

Assignment operators

These operators are used to store a value into an operand. In this case, the value to store is placed at the right side, and the operand that will receive the value is placed at the left. The following table summarizes these operators.

Table 7: Assignment Operators Summary

Operator	Description
=	The simple assignment operator. Stores values from the right side to the left-side operand.
+=	Add and assign operator. Adds the value of the operand placed at the right side to the value of the operand placed at the left, then assigns the result to the left operand.
-=	Subtract and assign operator. Subtracts the value of the operand placed at the right side from the value of the operand placed at the left, then assigns the result to the left operand.
*=	Multiply and assign operator. Multiplies the value of the operand placed at the right side by the value of the operand placed at the left, then assigns the result to the left operand.
/=	Divide and assign operator. Divides the value of the operand placed at the left side by the value of the operand placed at the right, then assigns the result to the left operand.
%/	Modulus and assign operator. Divides the value of the operand placed at the left side by the value of the operand placed at the right, then assigns the remainder to the left operand.

Conditional operator

The conditional operator (expressed as **?** **:**) performs an inline decision-making process. It evaluates the logical state of an expression placed at the left side of the **?** sign, and then executes one of two given expressions, both separated by the **:** sign. If the logical state of the expression is true, the expression placed at the left side of the **:** sign is executed; otherwise, the right-side expression is performed. The following code sample illustrates this.

Code Listing 13: Conditional Operator Code Sample

```
$total = $subtotal + $tax;
$discount = $total < 150 ? $total*0.15 : $total*0.20;
/*
$discount receives a value depending of $total value. If $total is less
than 150, then $discount receives the result of multiplying $total by
0.15. Otherwise, the result of multiplying $total by 0.20 is assigned to
$discount
*/
```

Precedence of operators in PHP

Operator precedence is the order in which a certain kind of operation (defined by the operator itself) is performed within an expression. Let's consider the following code sample.

Code Listing 14: An Expression Sample to Explain Operator Precedence

```
$tax = $subtotal - $discount * $taxrate;
```

This expression will perform three kind of operations: subtraction, multiplication, and assignment. The important thing to find out here is the order in which the computer will execute the operations. This depends on the operators' precedence.

To get a better understanding of operator precedence, we should classify the operators into the following categories.

- Unary operators: Operators that precede a single operand
- Binary operators: Operators that take two operands
- Ternary operators: Operators that take three operands and evaluate either the second or the third operand, depending on the value of the first one
- Assignment operators: Operators that assign a value to an operand

Taking these categories into account, the following table dictates the order in which operators are executed within an expression, from top to bottom.

Table 8: Operator Precedence Table

Associativity	Operator	Additional Information
non-associative	clone new	clone and new

Associativity	Operator	Additional Information
left	[	array()
right	**	arithmetic
right	++ -- ~ (int) (float) (string) (array) (object) (bool) @	types and increment/decrement
non-associative	instanceof	types
right	!	logical
left	* / %	arithmetic
left	+ - .	arithmetic and string
left	<< >>	bitwise
non-associative	< <= > >=	comparison
non-associative	== != === !== <> <=>	comparison
left	&	bitwise and references
left	^	bitwise
left	|	bitwise
left	&&	logical
left	||	logical
right	??	comparison
left	? :	ternary
right	= += -= *= **= /= .= %= &= |= ^= <<= >>=	assignment
left	and	logical
left	xor	logical
left	or	logical

Now, if we review Code Listing 14, the computer will first multiply $discount by $taxrate (multiplicative operators are placed first in precedence), and then will subtract the result of the operation from $subtotal.

Strings

A string is a sequence of characters, like "Welcome to PHP Succinctly e-book samples", that can be assigned to a variable or processed directly by a PHP statement.

The following code sample illustrates the use of strings.

Code Listing 15: Using Strings

```php
<?php

  $salutation = "Good morning";
  echo $salutation . ", today is " . date("l F jS \of Y") . "<br />";

?>
```

Note that there are two different uses of strings: a string assigned to a variable, and a string processed directly by a PHP statement (**echo** in this case). The dot employed in the expression that follows the **echo** statement is known as a concatenate operator, and is used to join the contents of two strings, evaluating them from left to right.

A string may be delimited either by single (') or double (") quotes, but there's a big difference in how the strings are treated, depending on the delimiter employed. Strings delimited with single quotes are treated literally, while double-quoted strings re"place variables with their values in case a string contains variable names within it. Also, double-quoted strings interpret certain character sequences that begin with the backslash (****), also known as escape-sequence replacements. These sequences are summarized in the following table.

Table 9: Escape Sequence Replacements Summary

Escape Sequence	Replacement
\n	Replaced with the newline character
\r	Replaced with the carriage-return character
\t	Replaced with the tab character
\$	Replaced with the dollar sign itself. This avoids the interpretation of the dollar sign as the variable name starting character.
\"	Replaced by a single double-quote
\'	Replaced by a single quote
****	Replaced by a backslash

The following code shows a bit about string treatment.

```php
<?php
   $somevariable = "Hello";
   $literalstring = 'The $somevariable will not print its contents';

   print($literalstring);
   print("<br />");

   $literalstring = "$somevariable will print its contents";
   echo $literalstring;
?>
```

As shown in the previous sample, when the variable name appears in the single-quoted string, it is treated literally as a part of the string itself. But, when the same variable name is placed into the double-quoted string, it is replaced with its own contents when the string is printed.

Arrays

We can define an array as a data structure intended to hold one or more values of similar type. That is, if we need to store 50 different strings, we can use one array to place them instead of declaring 50 string variables. We can create the following kinds of arrays in PHP:

- Numeric arrays – Arrays with a numeric index, the values of which are accessed and stored in linear order
- Associative arrays – Arrays with strings as indexes. The values stored in them are associated with key values instead of linear index order
- Multidimensional arrays – Arrays that contain one or more arrays and their values are accessed using multiple indices

Arrays can be created using the **array()** function, or by declaring the name of the variable that will store the array, followed by its index enclosed in brackets. The following code sample shows how to create an array.

Code Listing 17: Arrays in PHP

```php
<?php
   /* First method to create array. */
   $intnumbers = array( 1, 2, 3, 4, 5);

   /* We iterate the array */
   foreach( $intnumbers as $value ) {
    echo "Array member value is $value <br />";
   }

   /* Second method to create array. */
   $letternumbers[0] = "one";
   $letternumbers[1] = "two";
```

```php
   $letternumbers[2] = "three";
   $letternumbers[3] = "four";
   $letternumbers[4] = "five";

   foreach( $letternumbers as $value ) {
   echo "Array member value is $value <br />";
 }
?>
```

We just saw how to create a numeric array. Now, let's see how to create an associative array.

Code Listing 18: Associative Arrays

```php
<?php
   $intnumbers = array("one" => 1,"two" => 2,"three" => 3,"four" =>
4,"five" => 5);

   /* We iterate the array */
   foreach( $intnumbers as $k => $value ) {
    echo "$k => $value <br />";
   }
?>
```

Decision making

PHP provides a set of keywords to take a course of action based on a condition. If the condition is met, some statements are executed; otherwise, the script could do nothing or execute another group of different statements.

If elseif … else

The **if** statement executes some code if a certain condition is met. If it doesn't, execution jumps to the **elseif** clause and evaluates the expression placed after it. When the expression after the **elseif** clause evaluates to **true**, the code placed within that clause is executed. If the expression evaluates to **false**, the code within the **else** clause is performed.

Let's take a look at the following code sample.

Code Listing 19: Use of if elseif ... else Statement

```php
<?php
  date_default_timezone_set("Etc/GMT+7");
  $hour = date('H');
  if ($hour >=0 && $hour < 12)
      echo "Good Morning!";
```

```
  elseif ($hour >= 12 and $hour < 19)
    echo "Good afternoon!";
  else
      echo "Good Evening!";
?>
```

The previous code displays a greeting message depending on the hour of the day, which is stored in the $**hour** variable. The **if elseif … else** statement evaluates the **$hour** variable starting with the condition placed after the **if** statement. If the condition is met, the greeting message within the statement is printed. Otherwise, the condition after the **elseif** statement is now evaluated. Again, if this condition is fulfilled, the greeting message within **elseif** is printed. If not, the greeting message within the **else** statement is printed. Then, the execution ends.

Note: We can omit the `elseif` statement in order to get two-way decision-making.

Switch statement

The **switch** statement allows us to execute a block of code depending on a comparison of an expression, which is placed within parentheses after the statement declaration, and a series of values placed in each one of them in a separate **case** clause. Every **case** clause has an associated a block of code. This code will be executed when the expression linked to the **switch** statement equals to the value placed in that **case** clause. The following sample illustrates the use of this statement.

Code Listing 20: Using Switch Statement

```
<?php
  date_default_timezone_set("Etc/GMT+7");
  $hour = date('H');
  $dow  = date("D");

  if ($hour >=0 && $hour < 12)
      $greeting = "Good Morning!";
  elseif ($hour >= 12 and $hour < 19)
    $greeting = "Good afternoon!";
  else
      $greeting = "Good Evening!";

  switch ($dow){
          case "Mon":
              echo $greeting . ", today is Monday";
              break;
          case "Tue":
              echo $greeting . ", today is Tuesday";
              break;
          case "Wed":
```

```
                echo $greeting . ", today is Wednesday";
                break;
            case "Thu":
                echo $greeting . ", today is Thursday";
                break;
            case "Fri":
                echo $greeting . ", today is Friday";
                break;
            case "Sat":
                echo $greeting . ", today is Saturday";
                break;
            case "Sun":
                echo $greeting . ", today is Sunday";
                break;
            default:
                echo $greeting . "What day is this?";
        }
    ?>
```

In this code sample, the **switch** statement is employed to display the name of the day-of-week along with the greeting message. First, we call the **date('D')** function, which stores an abbreviated version of day-of-week name in the **$dow** variable. Then, we use **switch** to execute the **echo** statement with the corresponding full day-of-week name, depending on the value stored in the $**dow** variable.

Loops

A loop statement allows you to execute the same code block repeatedly, either while a certain condition is met, a specific number of times, or until a series of elements from a data structure have been all iterated. PHP supports the following loop statements:

- **for** – Loops through a code block a specified number of times
- **while** – Loops through a code block while a certain condition is met
- **do … while** – Loops through a code block once, and repeats the execution as long as the condition stablished is true
- **foreach** – Loops through a code block as many times as elements exist in an array

The following code snippets explain the syntax for each of the previous loop statements.

Code Listing 21: for Statement Syntax

```
for(initializer=initial value; condition; increment)
{
    //code to be executed
}
/*
```

```
The initializer is used as a counter of the number of times the code
block will be executed.

The condition is an expression which can be evaluated either true or
false, and in this case this condition establishes the final value the
initializer can take, before the loop ends.

The increment is a value which will be added to or subtracted from the
initializer, in order to keep the initializer from going beyond the final
value established in the condition, making the loop end.
*/

//Example.
for( $iteration = 0;$iteration <= 10;$iteration++)
{
     echo "$iteration";
}
```

Code Listing 22: while Statement Syntax

```
while(condition)
{
        //code to be executed
}
/*
Condition is an expression which evaluates either true or false. When it
evaluates to false, the loop ends.
*/

//Example.
$sheepnumber = 0;
while ($sheepnumber < 11)
{
     echo "Sheep number $sheepnumber";
     $sheepnumber++;
}
```

Code Listing 23: do ... while Statement Syntax

```
do
{
        //code to be executed
}
while(condition)
/*
```

```
The code block within curly brackets is executed once. After that,
condition is evaluated.
Condition is an expression which evaluates either true or false. When it
evaluates to false, the loop ends.
*/

//Example.
$sheepnumber = 1;
do
{
    echo "Sheep number $sheepnumber";
    $sheepnumber++;
}
while ($sheepnumber < 11);
$totalsheeps = $sheepnumber – 1;
echo "We count only $sheepnumber sheeps";
```

Code Listing 24: foreach Statement Syntax

```
foreach (arrayname as value)
{
  //Code to be executed
}
/*
The code block within curly brackets is executed as many times as there
are elements in the array that is evaluated.
*/

//Example.
$sheepsarray = array(1,2,3,4,5,6,7,8);
foreach($sheepsarray as $value)
{
    echo "Sheep number $value <br/>";
}
```

Continue and break special keywords

There are two special keywords that can be used within a loop: **break** and **continue**.

- The **break** keyword terminates the execution of a loop prematurely.
- The **continue** keyword halts the execution of a loop and starts a new iteration.

Let's look at the following code sample.

```php
<?php
  $subtotal = 3.5;
  while (true)
  {
    $taxrate = rand(0,10);
    if ($taxrate == 10) break;

    if ($taxrate == 0) continue;
    $taxvalue = $subtotal*($taxrate/100);
    echo "Tax to be payed $taxvalue <br />";
  }
?>
```

This code shows the execution of a **while** loop indefinitely (the condition established is always **true**). The mechanism used to end this loop is the **break** keyword. This keyword is executed only if the variable $taxrate takes a value of **10**. If not, the execution of code continues, and if the $taxrate variable evaluates to **0**, and the loop halts and starts a new iteration, so the **echo** statement is not executed.

Chapter summary

This chapter covered the basics of PHP, starting with the concept of a script. A script in PHP is a text file that contains pure PHP programming code, or PHP programming code embedded into HTML, and is executed in the web server.

As in most programming languages, the main way to store data in a PHP program is by using variables, which are identifiers intended to hold data dynamically, meaning the data stored in variables can change during the execution flow.

Variables in PHP are declared by denoting their names with a leading dollar sign (**$**), and then starting with a letter or underscore. Variables can be converted from one data type to another automatically.

A variable can be known only in certain regions of a PHP script. This is known as variable scope. PHP has the following variable scopes: *local*, for variables that are available only in the program where they are declared; *global*, for variables that can be accessed in any part of the executed program; *function parameters*, which are variables available within the function where they're employed; and *static*, which are variables declared inside functions that keep its values between every function call.

PHP also allows you to use constants. A constant is an identifier which holds a simple value and cannot be changed during the execution of the script. Constant identifier names are case-sensitive. The best practices in PHP dictate that constant names should be uppercase. Constants are defined by using the **define()** function.

PHP uses expressions to perform calculations. A set of symbols are used in order to perform these calculations. These symbols are called *operators,* and the identifiers declared between the operators are called *operands*. PHP has the following types of operators: arithmetic operators, which are used to perform operations with numbers; comparison operators, which are used to check if certain criteria between two operands are met; logical operators, which are employed to get a true or a false value depending on the logical state of two operands; assignment operators, which are employed to store the value of an expression into an operand; and conditional operators, which are employed to perform inline decision-making.

When an expression contains several operators, calculations are performed following a strict order, known as operator precedence. To explain this precedence, we can classify operators in the following categories: unary operators, which are operators preceding a single operand; binary operators, which take two operands; ternary operators, which take three operands, evaluating either the second or the third depending on the value of the first one; and assignment operators, which store a value into an operand.

Operator precedence is rather complicated. Common operators in an expression are executed in the following order: increment and decrement, unary, multiplicative and division, addition and subtraction, relational, equality, bitwise, ternary, assignment, logical **AND**, logical **XOR**, logical **OR**.

In PHP, we can use sequences of characters stored in variables or directly placed at the right of a statement. These sequences are called strings. Strings can be delimited either by single or double quotes. PHP treats strings in a different way depending on how they're delimited. Every PHP statement is considered a single-quoted string literal. When variable names are present in a double-quoted string, PHP replaces the name of the variable with its contents.

When we need to store several values of a similar type, PHP provides us with a data structure known as an array. We can use this structure instead of declaring many variables. In PHP we have the following kind of arrays: numeric, which store values that can be accessed using a numeric index; associative, which use strings as indexes and associate them to the values stored; and multidimensional, which contain one or more arrays accessing its values using multiple indexes. An array can be created using the **array()** function, or by declaring a variable followed by an index enclosed in brackets.

PHP provides a set of statements to take a course of action based on a condition. These statements are known as decision-making statements, and they are: **if … elseif … else**, which executes a code block when the condition after the **if** statement is **true**, or the code block within the **elseif** statement if the condition of the **if** statement is **false** and the condition of the **elseif** statement is **true**, or it executes the code within the **else** statement if both conditions are **false**; and the **switch** statement, which executes a block of code depending on a comparison of equality for an expression with a series of values, each one placed after a **case** clause, which also contains the code to be executed if the expression value is equal to the value associated to this particular **case** clause.

At the end, we learned about loop statements, which allow you to execute a particular code block repeatedly, either while a certain condition is met, a specific number of times, or until a series of elements from a data structure have been all iterated. These statements are: **for**, which loops through a code block a specified number of times; **while**, which loops through a code block while a certain condition is met; **do … while**, which loops through a code block once, and repeats the execution as long as the condition is **true**; and **foreach**, which loops through a code block many times as elements exist in an array.

PHP provides two special keywords to be used within a loop: **break**, which terminates the execution of a loop prematurely; and **continue**, which halts the execution of statements within a loop and starts a new iteration.

Chapter 4 Functions and File Inclusion

User-defined functions

Function definition

A function is a piece of code that receives data by using a set of variables named parameters, and then processes this data and returns a value. Sometimes functions don't receive any data at all, and only return a value, or just perform an action and don't return a value.

Creating functions

To create a function in PHP, we should use the reserved keyword **function**, followed by the name we want to give to that function. Then, we need to write the code the function will execute within curly braces. The following code shows an example of a function.

Code Listing 26: A Simple PHP User-Defined Function

```php
<?php

  function showmessage()
  {
      echo "This message was displayed from within a function <br />";
  }

  showmessage();
?>
```

This function code prints a message and returns no value; a return value is not strictly necessary when defining functions. There's another issue regarding this code: every time we call the function, the same message is printed. This is not practical, because if we want to print a different message, another function should be created. To deal with cases like this, we need to provide data to the function by using a series of identifiers called parameters.

Employing parameters

Parameters in functions are a series of identifiers declared after the function name, enclosed in parentheses. We can declare as many parameters as we need. These parameters are considered variables within the function.

Now, let's make a better version of the **showmessage()** function displayed in Code Listing 26.

Code Listing 27: A User-Defined Function with Parameters

```php
<?php

  function showmessage($message)
  {
      echo "$message <br />";
  }

  showmessage('This message is displayed by using parameters');
?>
```

Returning values from a function

A function can return a value to the calling program by employing the **return** statement. When the **return** statement is issued, the function execution stops. The following code returns a greeting message from a function.

Code Listing 28: A Function That Returns a Value

```php
<?php

  function greetingmessage()
  {
    date_default_timezone_set("Etc/GMT+7");
    $hour = date('H');
    $dow  = date('N')-1;
      $namesofdays =
array("Monday","Tuesday","Wednesday","Thursday","Friday","Saturday","Sunday
");

    if ($hour >=0 && $hour < 12)
        $result = "Good Morning!";
    elseif ($hour >= 12 and $hour < 19)
      $result = "Good afternoon!";
    else
        $result = "Good Evening!";

    $result = $result . ", today is $namesofdays[$dow]";
      return $result;
  }

  echo greetingmessage();

  ?>
```

Defining default values for parameters in a function

We can set function parameters to have a default value, in case the calling program doesn't pass any value to them. To define default values for parameters, we just place the desired value beside the parameter name, leading by an equal assignment operator (=).

Now, we're going to modify the **greetingmessage()** function of the previous sample, in order to display or omit the day-of-week name. By default, if a value is not passed to the corresponding parameter, the day-of-week name will be omitted.

Code Listing 29: Setting Up Default Values for Parameters

```php
<?php

  function greetingmessage($showdayofweek = false)
  {
    date_default_timezone_set("Etc/GMT+7");
    $hour = date('H');
    $dow  = date('N')-1;
      $namesofdays =
array("Monday","Tuesday","Wednesday","Thursday","Friday","Saturday","Sunday
");

    if ($hour >=0 && $hour < 12)
        $result = "Good Morning!";
    elseif ($hour >= 12 and $hour < 19)
      $result = "Good afternoon!";
    else
        $result = "Good Evening!";

      if ($showdayofweek) $result = $result . ", today is
$namesofdays[$dow] <br />";
      return $result;
  }

  echo greetingmessage(true); //Shows day-of-week name
  echo greetingmessage(); //Shows greeting message only

?>
```

Calling functions dynamically

A function can be called dynamically by storing its name into a string variable. Then, we can use this variable as we would the function name itself. The following sample uses the **greetingmessage()** function discussed in previous code examples and modifies it to be called dynamically, depending on the current hour.

Code Listing 30: Calling Functions Dynamically

```php
<?php

  function getgreetingfunction()
  {
       date_default_timezone_set("Etc/GMT+7");
       $funcnames =
array('goodmorning','goodmorning','goodafternoon','goodevening');
       $hour = date('H');
     return $funcnames[(int)$hour/6];
  }

  function dayofweekname($daynumber)
  {
       $namesofdays =
array("Monday","Tuesday","Wednesday","Thursday","Friday","Saturday","Sunday
");
     return $namesofdays[$daynumber - 1];
  }

  function goodmorning($showdayofweek = false)
  {
       $result = "Good Morning!";
       if ($showdayofweek) $result = $result . ", today is " .
dayofweekname(date('N'));
     $result = $result . " <br />";
       return $result;
  }

  function goodafternoon($showdayofweek = false)
  {
       $result = "Good Afternoon!";
       if ($showdayofweek) $result = $result . ", today is " .
dayofweekname(date('N'));
     $result = $result . " <br />";
       return $result;
  }

  function goodevening($showdayofweek = false)
  {
       $result = "Good Evening!";
       if ($showdayofweek) $result = $result . ", today is " .
dayofweekname(date('N'));
     $result = $result . " <br />";
       return $result;
  }

  $functionname = getgreetingfunction();
```

```php
  echo $functionname();
  echo $functionname(true);

?>
```

As noted in the previous code, we broke down the original function into five smaller ones. The first (**greeetingfunction)** will return the name of one of the program functions that will submit the greeting message. For doing this, we split the hours of a day into four pieces of six hours each (4 x 6 = 24). Each one of these pieces has a matching item into the **$funcnames** array.

This array stores the names of the functions that should be executed according to the hour of the day. We assume that the first twelve hours are considered part of the morning, which is from 0 to 11 hours. So, the first two items in the array store the same name. The third and the fourth items store the name for the afternoon and the evening greeting messages, in that order.

To get the proper array index, we divide the current hour returned by the **date('H')** function by 6. The **dayofweekname** function returns the day-of-week name, according to the current date. Finally, we store the corresponding greeting message function name into the $**functioname** variable, and then we execute it by placing the variable name after the **echo** statement.

Built-in functions

PHP has a large set of built-in functions. We can classify these functions into several categories. The most important of them, according to the scope of this book, are displayed in the following list.

- Array functions – Allow us to interact with and manipulate arrays
- Date & Time functions – Help us to get the date and time from the server in which scripts are running
- String functions – Allow us to manipulate strings
- Character functions – Help us to check whether a string or character falls into a certain class
- File system functions – Used to access and manipulate the file system of the server in which scripts are running
- Directory functions – Used to manipulate directories located at the server in which scripts are running

The following sections summarize each category of functions in a series of tables, where each table contains the most relevant functions (according to the scope of this book) within that category.

Array functions

Table 10: Array Functions Listing

Function	Description
`array()`	Creates an array
`array_change_key_case()`	Returns an array with all keys either in lowercase or uppercase
`array_chunk()`	Splits an array into chunks of arrays
`array_combine()`	Creates an array by using one array for keys and another array for its values
`array_count_values()`	Returns an array with the number of occurrences for each value
`array_diff()`	Compares array values and returns the differences
`array_fill()`	Fills an array with values
`array_keys()`	Returns all keys of any array
`asort()`	Sorts an array and maintains index association
`arsort()`	Sorts an array in reverse order and maintains index association

Date and time functions

Table 11: Date and Time Functions Listing

Function	Description
`date_date_set()`	Sets the date to a given date-time value
`date_format()`	Returns a formatted date according to a given format
`date_parse()`	Returns an associative array with detailed info about a given date
`date_time_set()`	Sets the time to a given value
`time_zone_indentifiers_list()`	Returns a numeric index array with all time zones identifiers
`date()`	Formats the local time/date

String functions

Table 12: String Functions Listing

Function	Description
`strlen()`	Returns the length of a string
`strpos()`	Finds and returns the position of the first occurrence of a string within another string
`substr()`	Returns a part of a string
`ltrim()`	Removes white spaces or other characters from the beginning of a string
`rtrim()`	Removes white spaces or other characters from the end of a string
`str_repeat()`	Returns a repeated string

Character functions

Table 13: Character Functions Listing

Function	Description
`ctype_alnum()`	Checks for alphanumeric characters in a string
`ctype_alpha()`	Checks for alphabetic characters in a string
`ctype_digit()`	Checks for numeric characters in a string
`ctype_lower()`	Checks for lowercase characters in a string
`crtype_upper()`	Checks for uppercase characters in a string
`ctype_cntrl()`	Check for control characters (such as Tab) in a string

File system functions

Table 14: File System Functions Listing

Function	Description
`copy()`	Creates a copy of a file
`delete()`	Deletes a file

Function	Description
dirname()	Returns the directory portion for a given path
file()	Reads an entire file into an array
file_exists()	Checks whether a file or directory exists
basename()	Returns the filename portion of a given path

Directory functions

Table 15: Directory Functions Listing

Function	Description
chdir()	Changes the current directory
dir()	Opens a directory handle and returns an object
closedir()	Closes a directory previously opened with **dir()**
getcwd()	Returns the current working directory
readdir()	Reads an entry from a directory handle
scandir()	Returns a list of all files and directories inside a specified path

File inclusion

Being able to reuse code is an important aspect of application development. Reusing code gives the developer an easy mechanism to maintain complex applications with reduced effort. In PHP, we can reuse code by using file inclusion.

File inclusion allows us to include the content of a PHP file into another file, before the server executes it. There are two functions that help us to perform file inclusion:

- **include()** – Copies all the text in the specified file into the file where the function is used. In case of any problem when loading the file, the function generates a warning message and the script continues its execution.
- **require()** – Similar to **include()**, except that if there is any problem when loading the file, the function generates a fatal error and the script execution is halted.

Tip: Use the `require()` **function instead of** `include()` **to prevent the script's execution when there is a problem with file inclusion.**

To illustrate file inclusion, we're going to split the code sample detailed in the "Calling functions dynamically section" into two files. The first one will contain all function declarations, and it will be named **commonfunctions.php**.

Code Listing 31: commonfunctions.php

```php
<?php
  function getgreetingfunction()
  {
      date_default_timezone_set("Etc/GMT+7");
      $funcnames =
array('goodmorning','goodmorning','goodafternoon','goodevening');
      $hour = date('H');
    return $funcnames[(int)$hour/6];
  }

  function dayofweekname($daynumber)
  {
      $namesofdays =
array("Monday","Tuesday","Wednesday","Thursday","Friday","Saturday","Sunday
");
    return $namesofdays[$daynumber - 1];
  }

  function goodmorning($showdayofweek = false)
  {
      $result = "Good Morning!";
      if ($showdayofweek) $result = $result . ", today is " .
dayofweekname(date('N'));
     $result = $result . " <br />";
      return $result;
  }

  function goodafternoon($showdayofweek = false)
  {
      $result = "Good Afternoon!";
      if ($showdayofweek) $result = $result . ", today is " .
dayofweekname(date('N'));
     $result = $result . " <br />";
      return $result;
  }

  function goodevening($showdayofweek = false)
  {
```

```php
        $result = "Good Evening!";
        if ($showdayofweek) $result = $result . ", today is " .
dayofweekname(date('N'));
      $result = $result . " <br />";
        return $result;
  }
?>
```

The second file will contain the script to be executed. It will be named **usingfileinclusion.php**.

Code Listing 32: usingfileinclusion.php

```php
<?php

  require("commonfunctions.php");
  $functionname = getgreetingfunction();

  echo $functionname();
  echo $functionname(true);

?>
```

This code sample shows the use of the **require()** function. This function takes the content of **commonfunctions.php** and inserts it into the script. Now when we call **getgreetingfunction()**, it is available to the script even when it is not explicitly defined in **usingfileinclusion.php**.

Chapter summary

This chapter explored topics related to functions. A function is a piece of code that receives data by using a set of variables named parameters, and then processes this data and returns a value. In PHP, we have user-defined and built-in functions. A user-defined function is created by the developer by using the reserved keyword **function**, followed by the name of the function.

When you need to pass data to a function, you should use a series of identifiers called parameters. Parameters are declared after the function name and enclosed in parentheses. You can declare as many parameters as you need. All these parameters will be considered variables within the function. A function can return a value to the calling program employing the **return** statement.

We can set functions' parameters to have a default value, in case the calling program doesn't pass any value to any of them. Default values are defined by placing the desired value at the right side of the parameter name, leading with an equal assignment operator (=).

A function can be called dynamically by storing its name into a string variable. We can use this variable as we would the function name itself.

PHP has a large set of built-in functions that can be classified in categories. The categories discussed in this chapter are: array functions, which allow the developer to interact with and manipulate arrays; date and time functions, which get the date and time from the server in which scripts are running; string functions, which allow the developer to manipulate strings; character functions, which check whether a string or character falls into certain class; file system functions, which access and manipulate the file system; and directory functions, which are used to manipulate directories.

Reusing code is important for maintaining complex applications with minimal effort. PHP allows the reuse of code by means of file inclusion. File inclusion is the mechanism used to insert the content of a PHP file into another one, and it is performed by two functions: **`include()`**, which copies all the text in the specified file into the script, generating a warning message when a problem occurs; and **`require()`**, which is similar to **`include()`**, except that it halts script execution when a problem occurs.

Chapter 5 Files and Databases

Managing Files with PHP

PHP provides a series of functions that help us manipulate files, performing operations such as opening, reading, writing, and closing a file.

Reading a file

If we want to read the contents of a file, the functions **fopen()**, **filesize()**, **fread()**, and **fclose()** should be used together. The following code sample demonstrates how to read a file and display its contents using a webpage.

Code Listing 33: Reading a File

```php
<?php
  $filename = "license.txt";
  $file = fopen( $filename, "r" );

  if( $file == false )
   {
    echo ( "Error when opening file" );
    exit();
   }

  $filesize = filesize( $filename );
  $filecontents = fread( $file, $filesize );
  fclose( $file );

  echo "<html>\n";
  echo "<head>\n<title>Reading a file using PHP</title>\n</head>\n";
  echo "<body>\n<pre>$filecontents</pre></body>\n";
  echo "</html>\n";

?>
```

This code attempts to open the file named **license.txt**, by using the **fopen()** function. The **"r"** parameter value passed to the function indicates that the file should be opened as read-only. If the operation fails, an error message is displayed, and the script execution is halted with the **exit()** function.

A file pointer is saved into the **$file** variable. This pointer is going to be used to perform the rest of file operations in the script.

After the file is opened, we need to calculate the size of it in order to tell PHP how many bytes we need to read. This task is executed using the **filesize()** function, and the actual size is stored in the **$filesize** variable.

Now, we store the contents of the **license.txt** file, which is assumed to be in the same directory as the PHP script, using the **fread()** function. To perform this operation, **fread()** needs to receive the file pointer stored previously in the **$file** variable, and the file size stored in the **$filesize** variable. The file contents are placed into the **$filecontents** variable. After that, we close the file using the **fclose()** function and the file pointer is saved in the **$file** variable.

The last statements send the HTML code needed to create the webpage, including the contents of the text file that were read previously, enclosed in a HTML **<pre>** tag.

Writing text to a file

We can also write text to a file using PHP. This is similar to reading a file, but we need to use the **fwrite()** function instead. The following code writes a webpage into a text file, and then uses the file contents to display the page in the web browser.

Code Listing 34: Writing Text to a File

```php
<?php
  require("commonfunctions.php");

  function createwebpage()
  {
      $greetingfunction = getgreetingfunction();
      $greeting = $greetingfunction(true);

      $webpagecontent = "<html>\n<head>\n<title>Writing a file using
PHP</title>\n</head>\n";
      $webpagecontent = $webpagecontent .
"<body>\n<pre>$greeting</pre>\n<pre>This web page was created writing a
file with PHP</pre>\n</body>\n</html>\n";
     $filename = "webpage.txt";
      $file = fopen($filename,"w");
      fwrite($file,$webpagecontent);
      fclose($file);
  }

  createwebpage();

  $filename = "webpage.txt";
  $file = fopen( $filename, "r" );

  if( $file == false )
   {
    echo ( "Error when opening file" );
```

```php
   exit();
   }

   $filesize = filesize( $filename );
   $filecontents = fread( $file, $filesize );
   fclose( $file );

   echo "$filecontents";

?>
```

In this sample, we're employing file inclusion in order to use the set of functions defined in the **commonfunctions.php** file. The `createwebpage()` function stores the webpage's HTML code in a variable named `$webpagecontent`, and then writes this content in the **webpage.txt** file, assuming the PHP script has permissions that allow writing to the current directory. Now, to display the webpage, we call `createwebpage()` first, and then we read the **webpage.txt** file into the `$filecontents` variable. At the end, we display the `$filecontents` variable, which is the webpage itself.

Connecting to MySQL databases

As explained in Chapter 1, PHP supports a wide range of Database Management Systems (RDBMS), with MySQL being the most-used database system. PHP 7 includes an extension named *mysqli* (MySQL improved), which allows us to access MySQL 4.1 and above. This extension is implemented through a class named `mysqli`. For the purposes of this book, we're going to use this class for making connections to MySQL databases.

Prerequisites

In order to work with the exercises explained in this section, your computer should have the latest version of MySQL and the MySQL Workbench utility installed.

Installing MySQL in the local computer

MySQL provides an installation program for the Windows environment. This program can be downloaded [here](#).

Once the program is downloaded, we just need to double-click the filename, and the installation process will display the following dialog box.
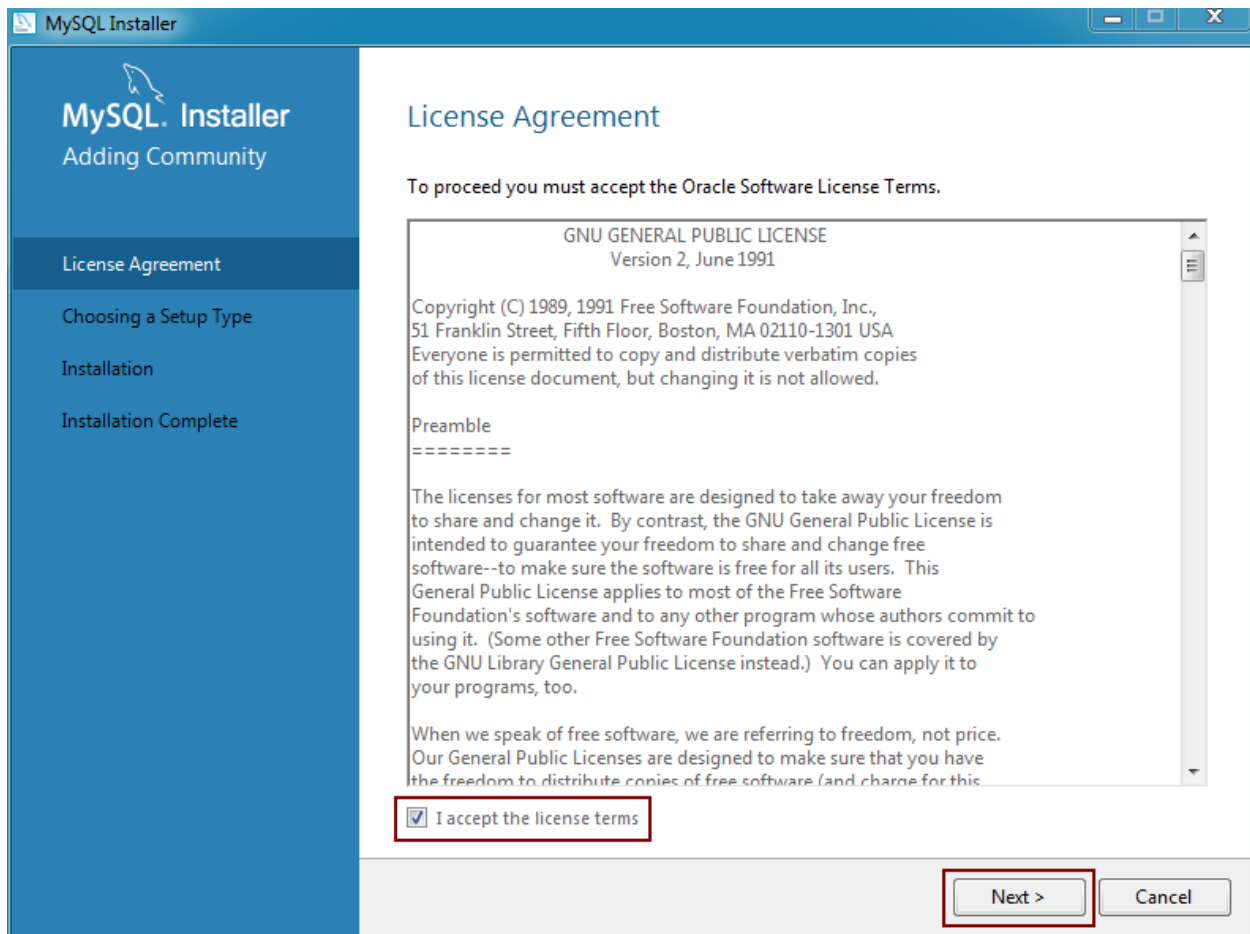
Figure 29: MySQL Installer License Agreement

To continue the process, we should check the **I accept the license terms** checkbox, and then click **Next**.

Now, the installer asks for a setup type that suits our installation case. Since we're practicing connecting MySQL using PHP, a **Developer Default** type would be okay for us. This is the default setup type for the installation process, so we only need to click **Next**, and the process will continue.
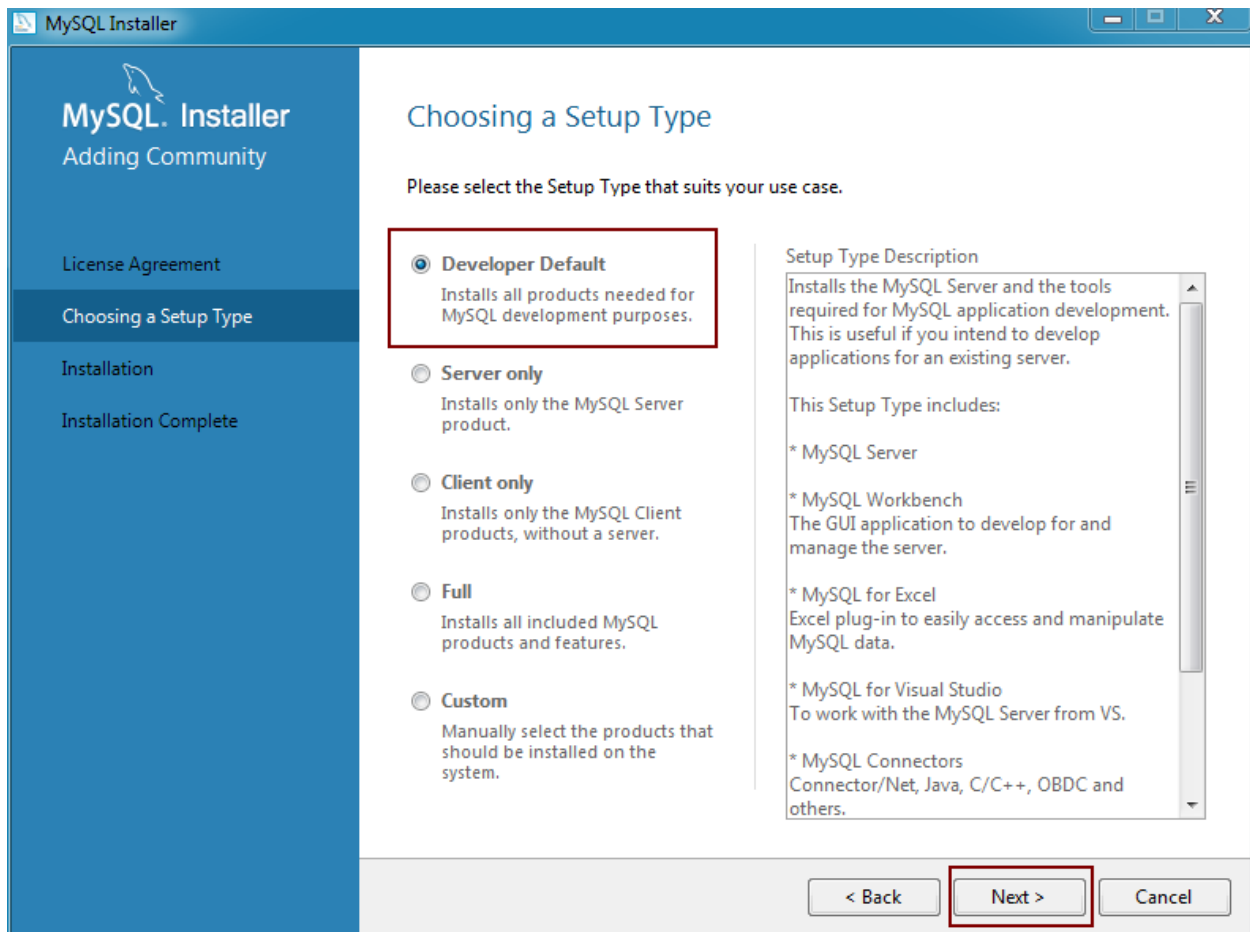
Figure 30: Choosing Setup Type

The MySQL Installer now checks the computer system for all external requirements needed to install the products selected in the previous dialog. The installer can download and install some prerequisites, but in some cases a manual intervention is required. In these cases, we should click over those requirements tagged with a Manual legend in order to download the files needed. Once we finish installing manual prerequisites, we can use the **Check** button to review if those requirements are now met. Then, we need to click on the **Execute** button to perform an automatic installation of those requirements still missing. At the end, when all requirements are installed, we should click **Next** to continue with the process.
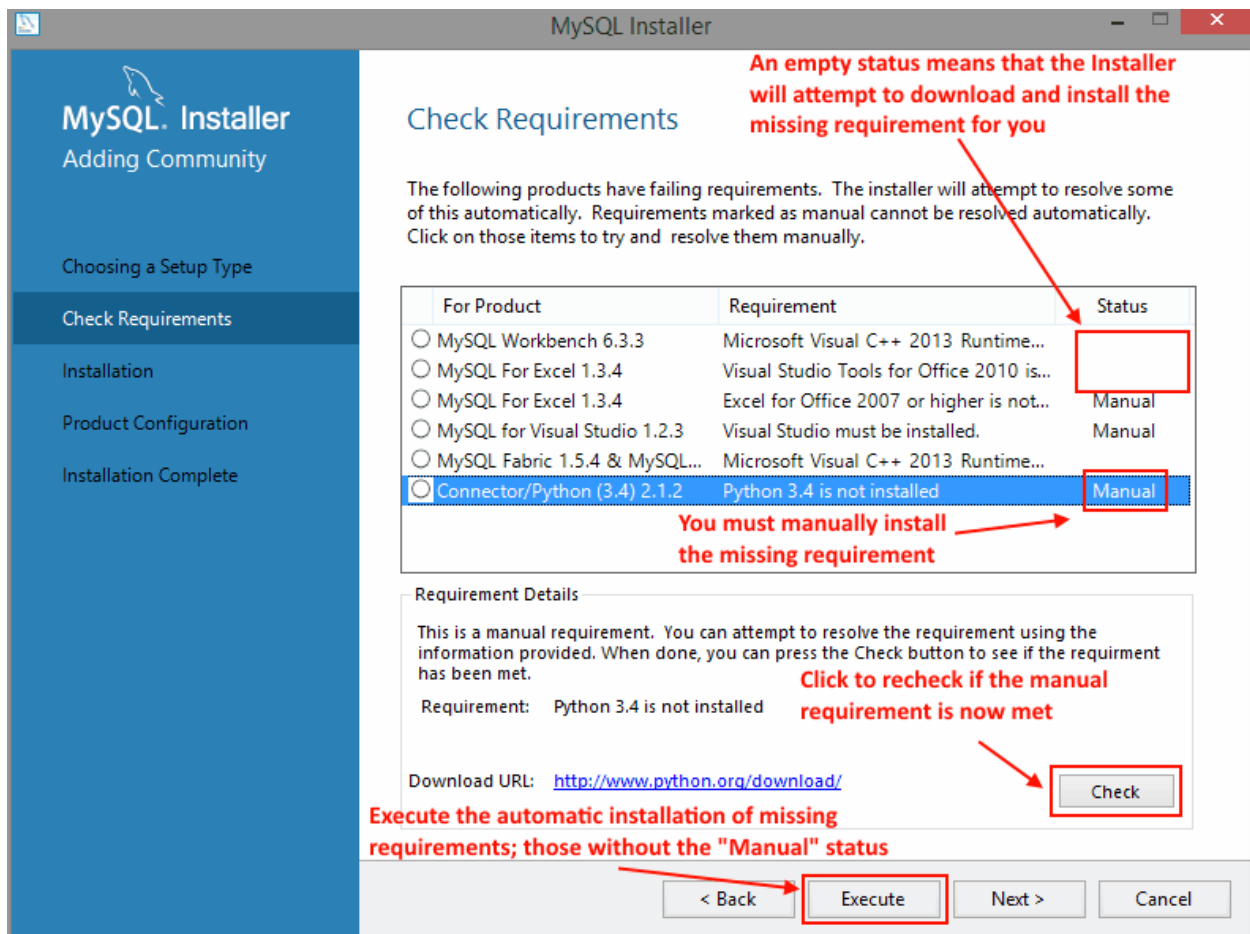
Figure 31: Checking Installation Requirements

The next dialog box displayed by the MySQL installer lists all products scheduled for installation. Click the **Execute** button, and the installation program will begin to deploy the products. A progress bar will be displayed below the Status column of the list. When a product deployment is finished, a status of "Complete" will be shown.
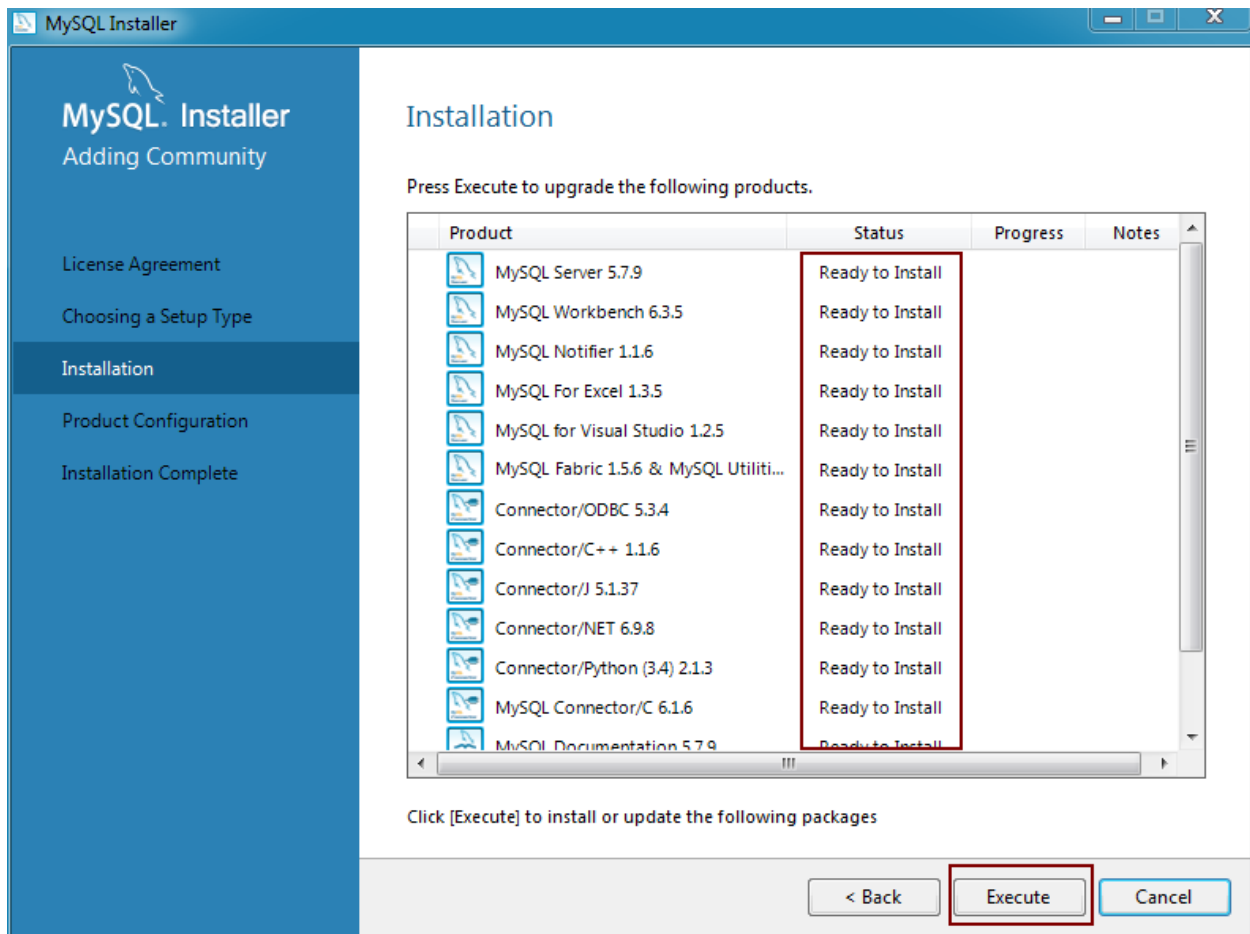
Figure 32: Ready to Install Dialog Box

After all products are installed, the installation program displays the **Configuration** dialog box, listing all the products that need to be configured. Usually, MySQL Server, Samples, and Examples products are configurable, so they appear listed in the dialog box with a "Ready to Configure" status. We should click **Next** to begin product configuration.
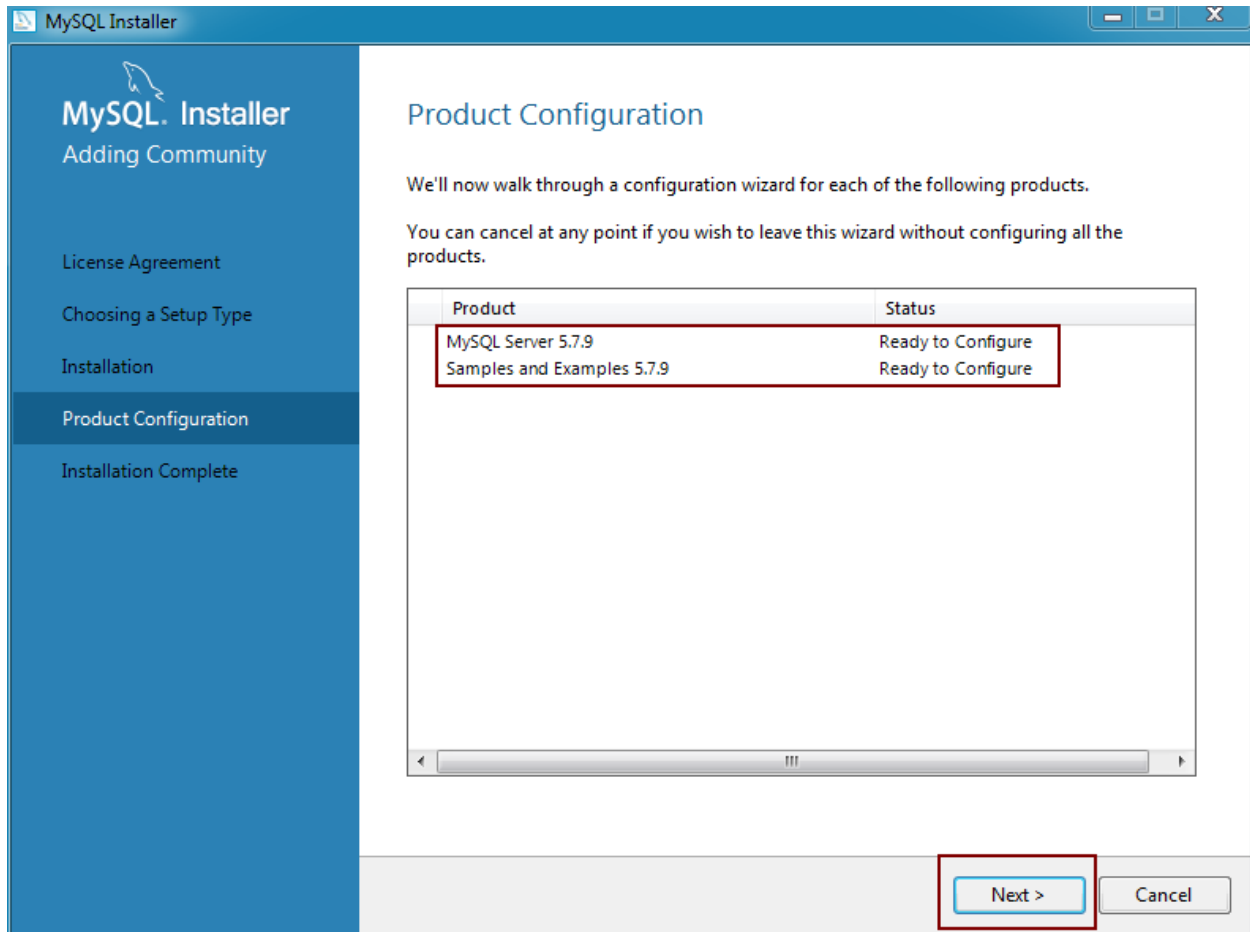
Figure 33: Product Configuration Dialog Box

At this point, a dialog box will appear showing the installation type we chose at the beginning of the installation process, and asking for changes to the default connectivity values. This dialog box allows us to display or hide the advanced configuration options. Even though the next figure shows the **Advanced Options** checked, hiding this dialog and letting MySQL installer set the default advanced values is recommended. It is also recommended that you leave the default connectivity values, since they ensure proper database functionality.

To continue the configuration process, click **Next**.
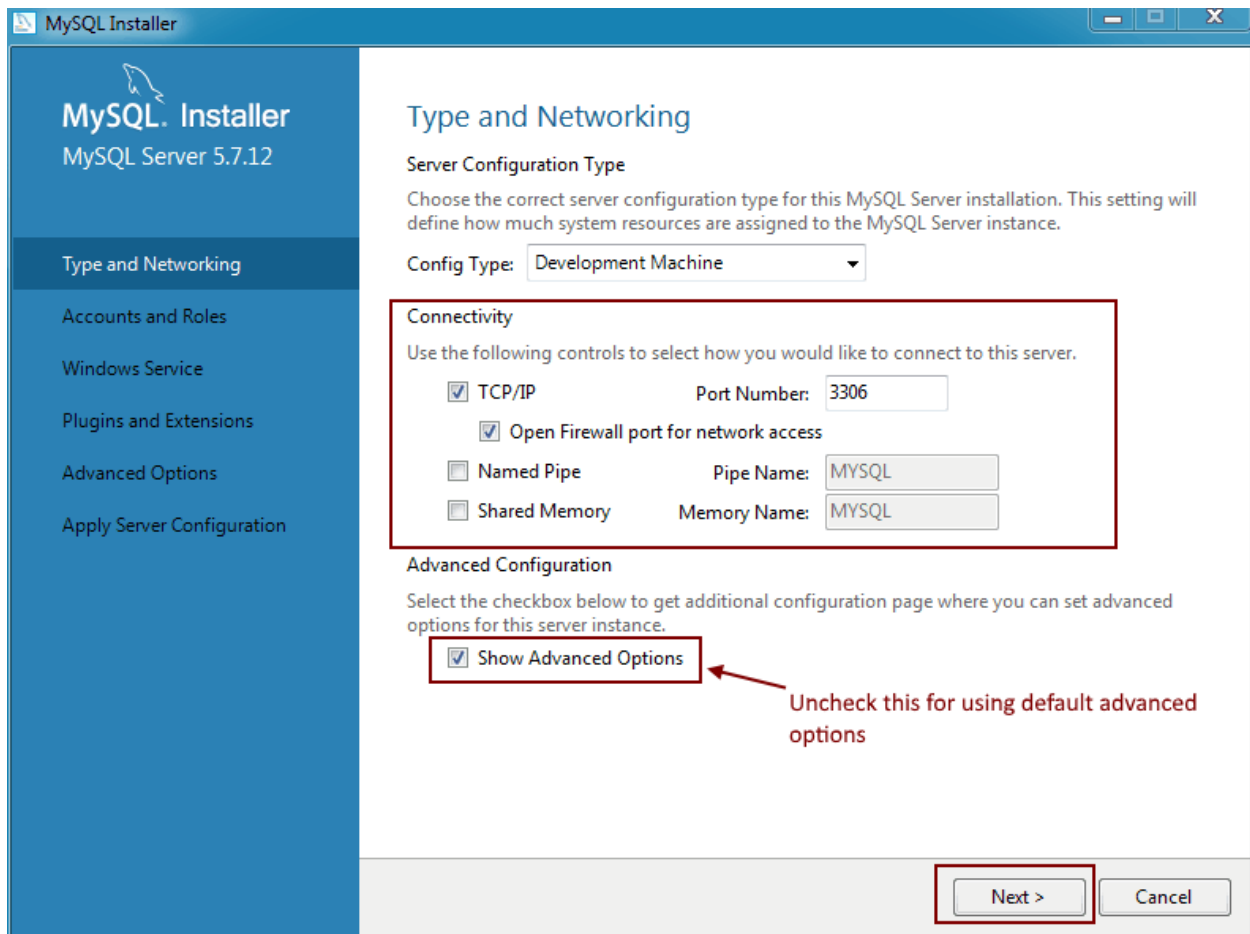
Figure 34: MySQL Server Configuration

The next dialog box displayed is **Accounts and Roles**. This is where we're going to establish the password for the root user. I recommend entering a password that the configuration program considers strong. After doing this, click **Next** to continue.
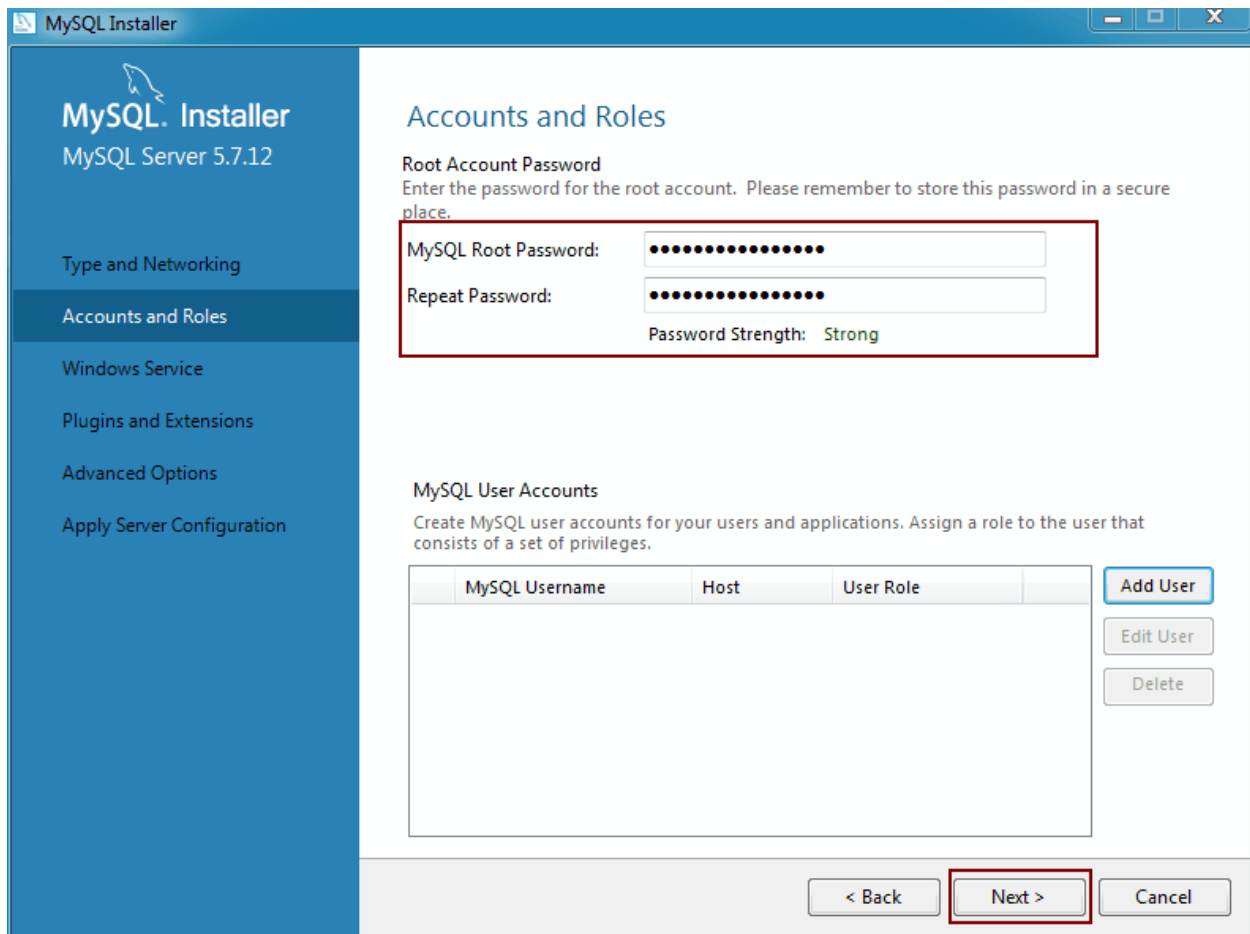
Figure 35: Setting the Root Account Password

The next step is to configure the Windows Service details, such as service name, how MySQL Server Windows service is executed, and whether or not MySQL server should be loaded at startup. The configuration program displays a series of default values. If you have no particular needs related to this specific configuration, I recommend using the default values.
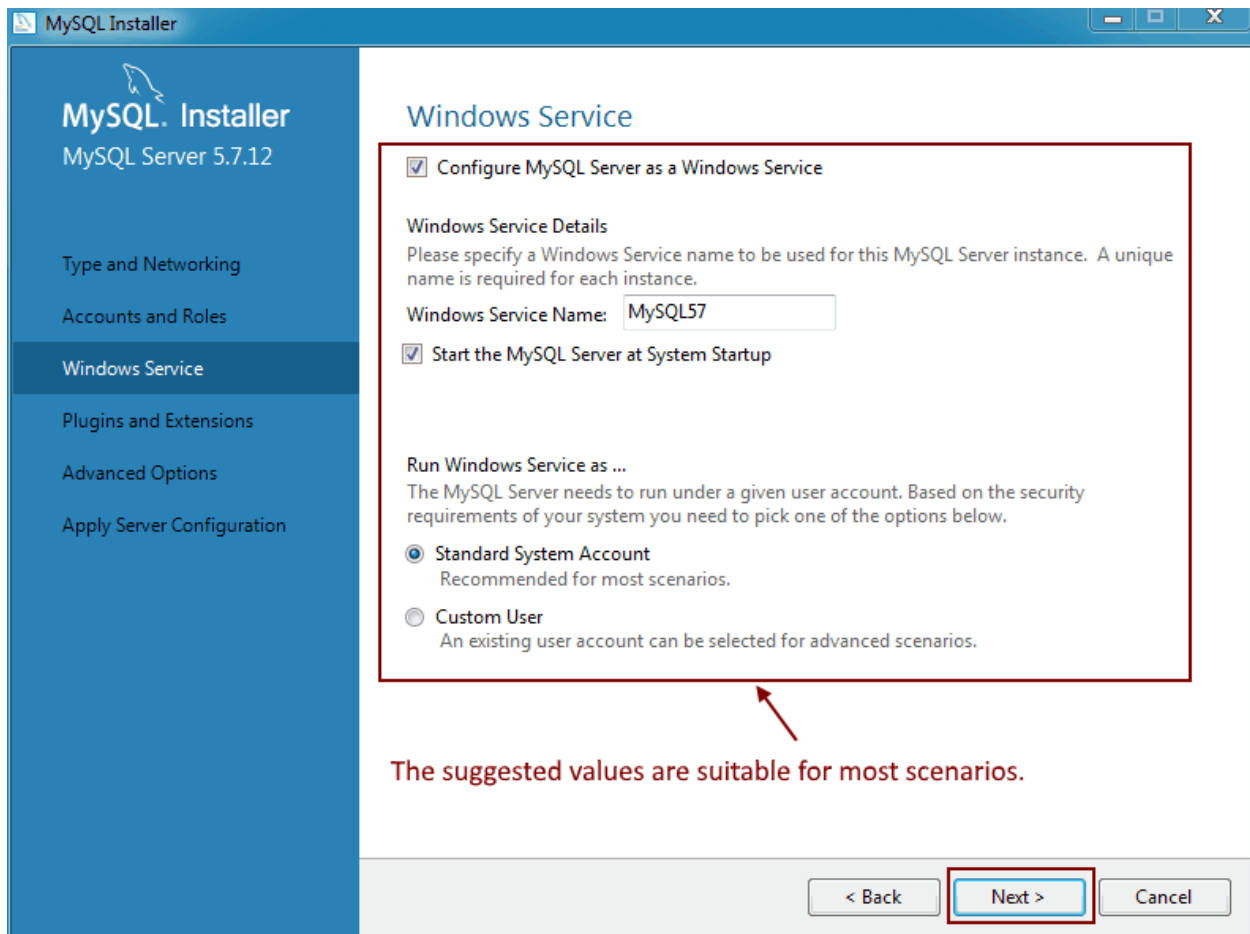
Figure 36: Setting up Windows Service

Plug-ins and extensions are beyond the scope of this book, and since the **Advanced Options** checkbox was disabled, the **Apply Server Configuration** dialog box will appear next. Click the **Execute** button to begin the configuration process. When this process is finished, the opening dialog box is reloaded and we can perform another installation and configuration.
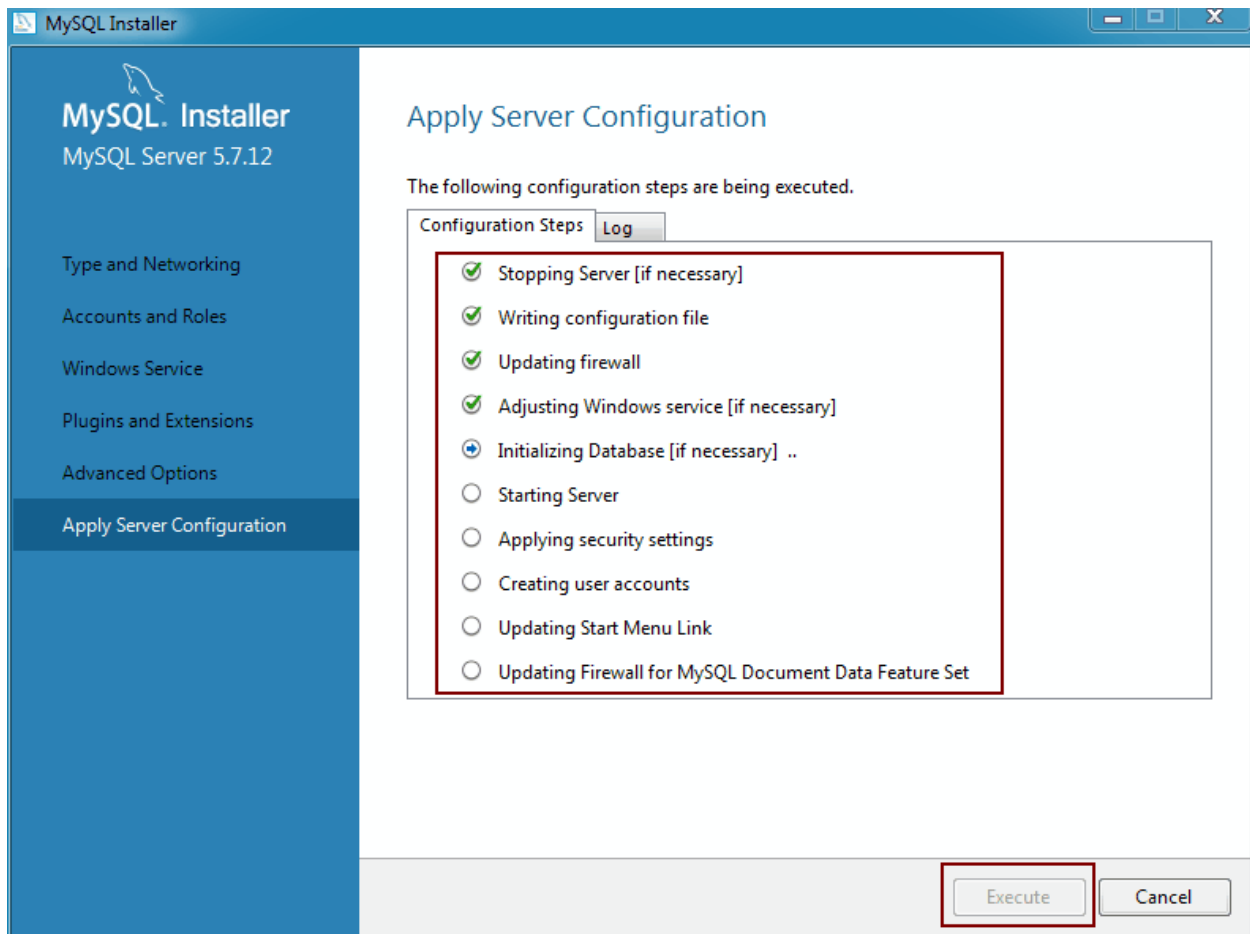
Figure 37: Applying Configuration Values to the Server

Using MySQL Workbench to create a database

A utility program called MySQL Workbench is installed along with the MySQL Server Development installation type. This program will help us to manage our server, including the ability to create databases. I recommend making a desktop shortcut for this program so you can quickly access it anytime you need it.

After you create the shortcut, double-click it to launch the MySQL Workbench utility. Once the program is launched, the following dialog box will appear.
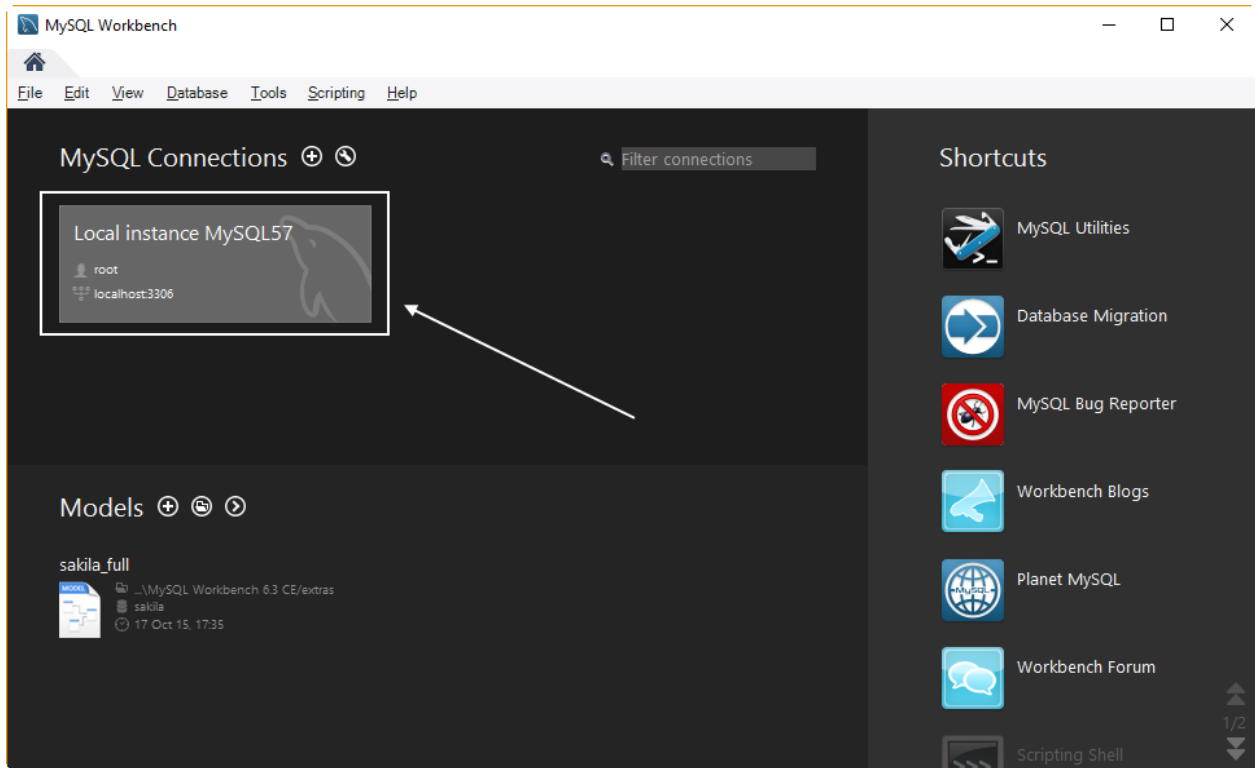
Figure 38: MySQL Workbench Utility Main Screen

Figure 38 shows the MySQL Workbench main screen. This main screen is divided in three areas: MySQL connections, which holds one or more shortcuts pointing to a particular MySQL server; Models, which holds shortcuts that point to a database model file (discussion of these files are beyond the scope of this book); and Shortcuts, which holds shortcuts to other MySQL utilities or forums.

By default, a shortcut to our local MySQL server instance appears in the main screen. So, to connect to this instance, we should click the shortcut. Then, main screen will look like the following figure.
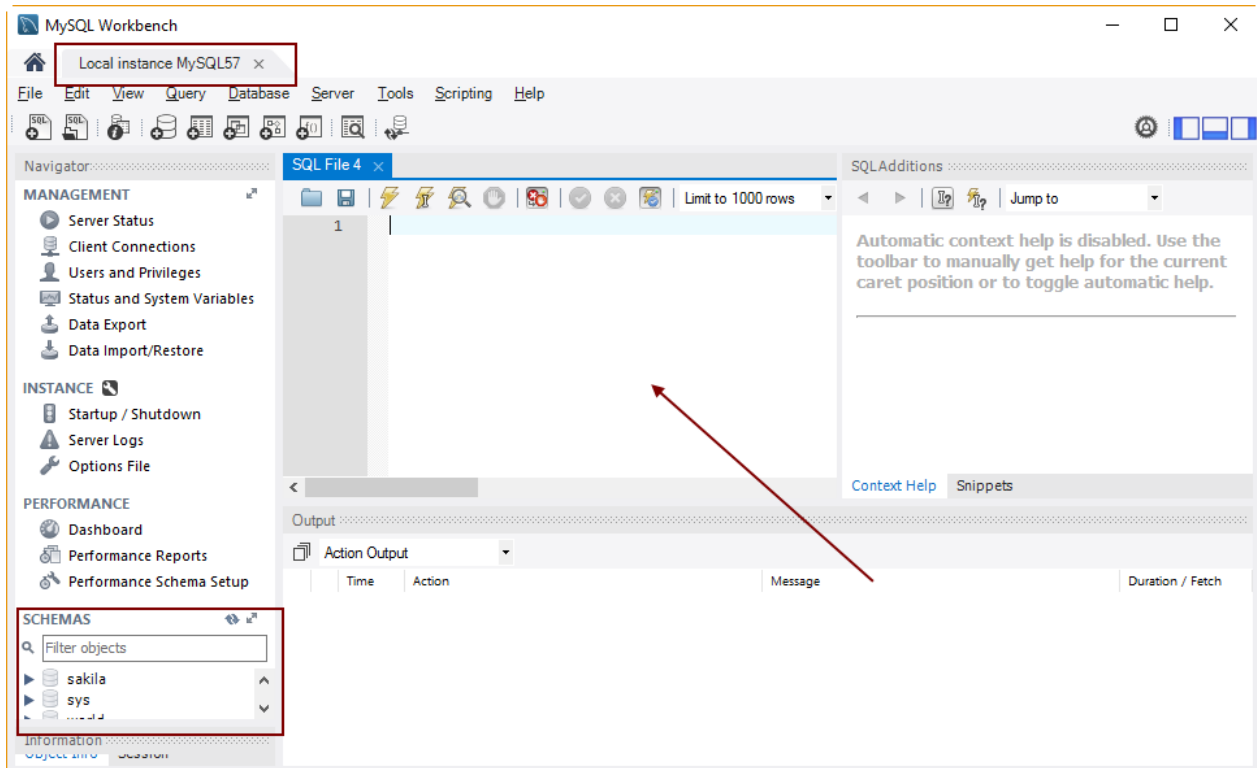
Figure 39: MySQL Local Instance Main Window

Figure 39 shows the MySQL Local Instance Manager. This is the place where we are going work with the database server. The area indicated by the red arrow is called the Query Tab, which is used to enter all SQL statements needed to manage the server. These statements include database and table creation. The area surrounded by the red square is where all available schemas in the server are displayed. For practical purposes, a database is the same as a schema.

Now, we're going to create our database.

The contactinfo database

The exercises discussed in the following sections will rely on a database named **contactinfo**, which will contain a single table to save our contacts' information. The following code snippet will create this database using MySQL Workbench utility.

Code Listing 35: Creating the contactinfo Database

```
CREATE SCHEMA contactinfo;

USE contactinfo;

CREATE TABLE contacts (ID INT AUTO_INCREMENT PRIMARY KEY,
NAME VARCHAR(200) DEFAULT '' NOT NULL,
EMAIL VARCHAR(300) DEFAULT '' NOT NULL,
```

```
PHONENUMBER VARCHAR(50) DEFAULT '' NOT NULL,
SUBJECT VARCHAR(200) DEFAULT '' NOT NULL,
MESSAGE TEXT);
```

Now, the query tab of MySQL Workbench will look like the following figure.
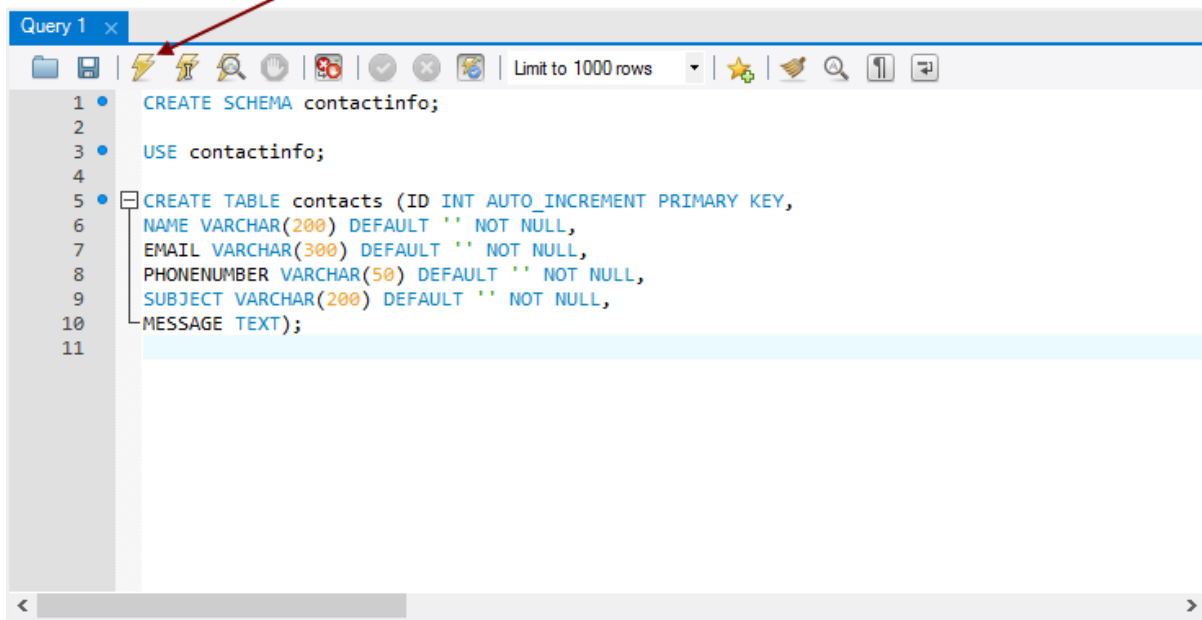
Figure 40: Creating the Database from a Query Tab

As shown in Figure 40, we need to click the lightning icon to execute the code and create the database. After code execution, the **contactinfo** database will be displayed in the schemas list. (You may have to click on the **Refresh** icon to see it.)
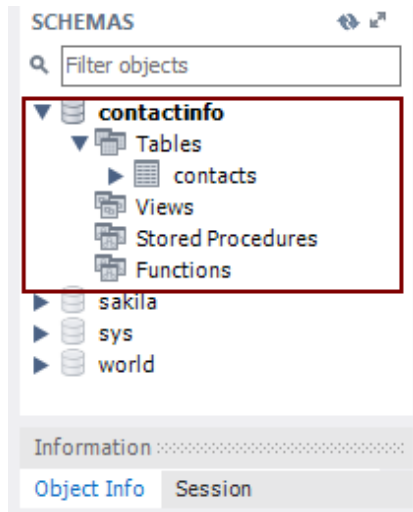
Figure 41: contactinfo Database in the Schemas List

Now, we are ready to use PHP to access our database.

Our first database connection

The first thing we're going to do is test a MySQL connection to the **contactinfo** database. This can be done by executing the following code.

Code Listing 36: Testing a Connection to the contactinfo Database

```php
<?php

  $dbhost = 'localhost';
  $dbuser = 'root';
  $dbpass = 'your password';
  $database = 'contactinfo';
$mysqli = new mysqli($dbhost, $dbuser, $dbpass, $database);

if ($mysqli->connect_errno) {
    echo "We're sorry. The website can not connect to the database";
    echo "Error: MySQL connection failed: \n";
    echo "Errno: " . $mysqli->connect_errno . "\n";
    echo "Error: " . $mysqli->connect_error . "\n";

    exit;
}
echo "MySql connection succeeded";
$mysqli->close();

?>
```

This code attempts to connect to the **contactinfo** database. The connection credentials are supplied in the $**dbhost**, $**dbuser**, and $**dbpass** variables, and the database name is stored in a variable named $**database**. We use the **mysqli** class in order to create the connection. The credentials for establishing the connection are passed when we attempt to create an instance of the class and save it into the $**mysqli** variable. If the property **connect_errno** of the class evaluates to **true**, a series of error messages are displayed, indicating that it was not possible to make a connection. Then, the **exit** statement ends the script.

Assuming everything is executed successfully, the output displayed by the web browser should look like the following figure.
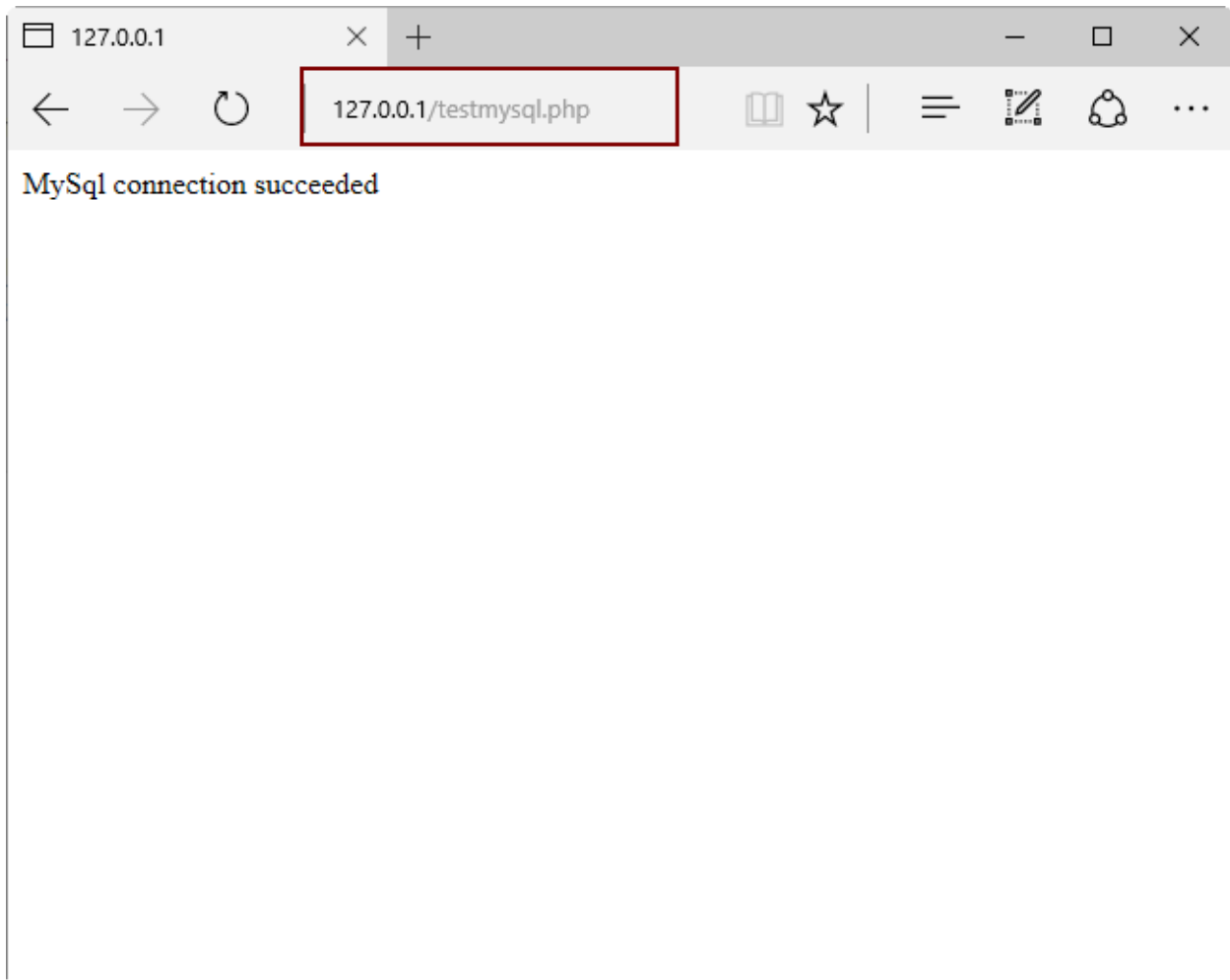
Figure 42: A Successful Connection to MySQL

Inserting a row in the contacts table

Since our database is recently created, it is empty. So, we are not able to perform any kind of query because there's no data to view. Therefore, the first thing we will do is insert a couple of rows into the contacts table.

```php
<?php

  $dbhost = 'localhost';
  $dbuser = 'root';
  $dbpass = 'mypassword';
  $database = 'contactinfo';
$mysqli = new mysqli($dbhost, $dbuser, $dbpass, $database);

if ($mysqli->connect_errno) {
    echo "We're sorry. The website can not connect to the database <br />";
    echo "Error: MySQL connection failed: <br />";
    echo "Errno: " . $mysqli->connect_errno . "<br />";
    echo "Error: " . $mysqli->connect_error . "<br />";

    exit;
}

$sql = "INSERT INTO contacts (name,email,phonenumber,subject,message)
VALUES ('John Doe', 'johndoe@myemaildomain.com', '(253)001-2345','Test data
row','Testing data insertion')";

if ($mysqli->query($sql) === TRUE) {
    echo "New record created successfully";
} else {
    echo "Error: " . $sql . "<br>" . $mysqli->error;
}

$mysqli->close();

?>
```

The previous code uses a **SQL INSERT** statement assigned to the variable $**sql** to add a new row in the contacts table. The first thing the code does is try to connect to the database. If connection succeeds, the **SQL INSERT** statement is executed by the **query()** method of the **$mysqli** class instance. If the value returned by the method evaluates to **true**, a message indicating a successful operation is displayed. Otherwise, an error message is shown. In both cases, the connection is closed at the end using the **close()** method of **mysqli** class.

Inserting data using parameters

SQL statements allow the use of parameters so that a query can receive data to be processed as a function does. The following code shows a parameterized SQL statement.

```
INSERT INTO contacts (name, email, phonenumber, subject, message) VALUES
(?,?,?,?,?);
```

The question marks (**?**) in this code sample correspond to the parameters' declaration. In this case, the statement has five parameters to receive the data that will be inserted in the table. In order to execute the statement successfully, the SQL statement must receive the corresponding data before its execution. To perform this action, we need to bind the parameters with the variables that hold the data. This can be done by using the **bind_param** method of the **mysqli** class **statement** property. But, unlike the insertion code sample discussed in the previous section, **statement** demands the SQL statement to be compiled before binding data. Compiling the statement does not mean executing it, but reviewing it to check if it is okay. To perform this operation, we should use the **prepare** method of the **statement** property. Now, let's take a look at the following code.

Code Listing 39: Inserting Rows Using Parameters

```php
<?php

  $dbhost = 'localhost';
  $dbuser = 'root';
  $dbpass = 'mypassword';
  $database = 'contactinfo';
$mysqli = new mysqli($dbhost, $dbuser, $dbpass, $database);

if ($mysqli->connect_errno) {
    echo "We're sorry. The website can not connect to the database <br />";
    echo "Error: MySQL connection failed: <br />";
    echo "Errno: " . $mysqli->connect_errno . "<br />";
    echo "Error: " . $mysqli->connect_error . "<br />";

    exit;
}
$contact_name = "Another John Doe";
$email_addr = "anotherjohndoe@myemaildomain.com";
$phonenumber = "(654)290-4567";
$subject = "Adding rows with parameters";
$message = "This row was added using parameters";

$sql = "INSERT INTO contacts (name,email,phonenumber,subject,message)
VALUES (?,?,?,?,?)";

$statement = $mysqli->stmt_init();

if ($statement->prepare($sql))
{
```

```php
    $statement-
>bind_param("sssss",$contact_name,$email_addr,$phonenumber,$subject,$messag
e);
      $statement->execute();
      $statement->close();
}

$mysqli->close();
?>
```

The first thing to note in this code is the parameterized SQL statement that is assigned to the **$sql** variable. Before the statement is assigned, a set of five variables are declared, and the data that will be inserted in the new row are assigned to each one of them. The $**statement** variable receives a statement object by means of the **stmt_init()** method. Now, the SQL sentence is compiled by using the **prepare()** method, and if it is alright, the program binds the parameters declared in the SQL sentence with the variables that hold the data.

To perform data binding, the method **bind_param()** is employed. The first parameter of this method is a string formed by a sequence of characters, as long as the number of parameters to be bound. In this case, there are five parameters that appear in the SQL statement, so the string is five characters long. The **s** in the first position of the string stands for the string data type of the first parameter appearing in the SQL sentence. Because all parameters for the sentence are string data typed, the sequence of characters is formed by **s** only.

Now, to do the insert operation, the **execute()** method of the statement object is performed. After that, the **close()** method of the statement object is executed to free all resources employed. At the end, the connection is closed using the **close()** method of **mysqli** class.

Querying the contacts table

Now that the contacts table has data that can be queried, we're going to create a PHP script that displays the contents of the first three columns from all rows in the contacts table.

Code Listing 40: A Script Which Queries the Contacts Table

```php
<?php

  $dbhost = 'localhost';
  $dbuser = 'root';
  $dbpass = 'mypassword';
  $database = 'contactinfo';
$mysqli = new mysqli($dbhost, $dbuser, $dbpass, $database);

if ($mysqli->connect_errno) {
    echo "We're sorry. The website can not connect to the database <br />";
    echo "Error: MySQL connection failed: <br />";
    echo "Errno: " . $mysqli->connect_errno . "<br />";
```

```php
    echo "Error: " . $mysqli->connect_error . "<br />";

    exit;
}

$sql = "SELECT contacts.* FROM contacts ORDER BY contacts.name";
$resultset = $mysqli->query($sql);

if ($resultset->num_rows > 0)
{
      while ($datarow = $resultset->fetch_assoc())
      {
            echo "Contact Id: " . $datarow["ID"] .  " - Contact Name: " .
$datarow["NAME"] . " - Contact Email: " . $datarow["EMAIL"] . "<br />";
      }
}
else
{
   echo "No contacts available";
}
$mysqli->close();
?>
```

As we can see in Code Listing 40, the contents of the **$sql** variable have changed to a SQL
SELECT statement. Although the **query** method of **mysqli** class is used, the approach in this
sample is a little different. The result returned from the method is stored in a variable named
$resultset, and it should be a dataset containing all rows from the contacts table. The
conditional **if** statement inquires about the number of rows returned. If this number is greater
than zero, a **while** loop iterates through all the dataset and displays the contents of the **ID**,
NAME, and **EMAIL** columns for each row. Now, assuming that the previous sample was saved in
a file named **queryingcontacts.php,** if we type **http://127.0.0.1/queryingcontacts.php** into the
address bar of the web browser, the output displayed should look like the following figure.
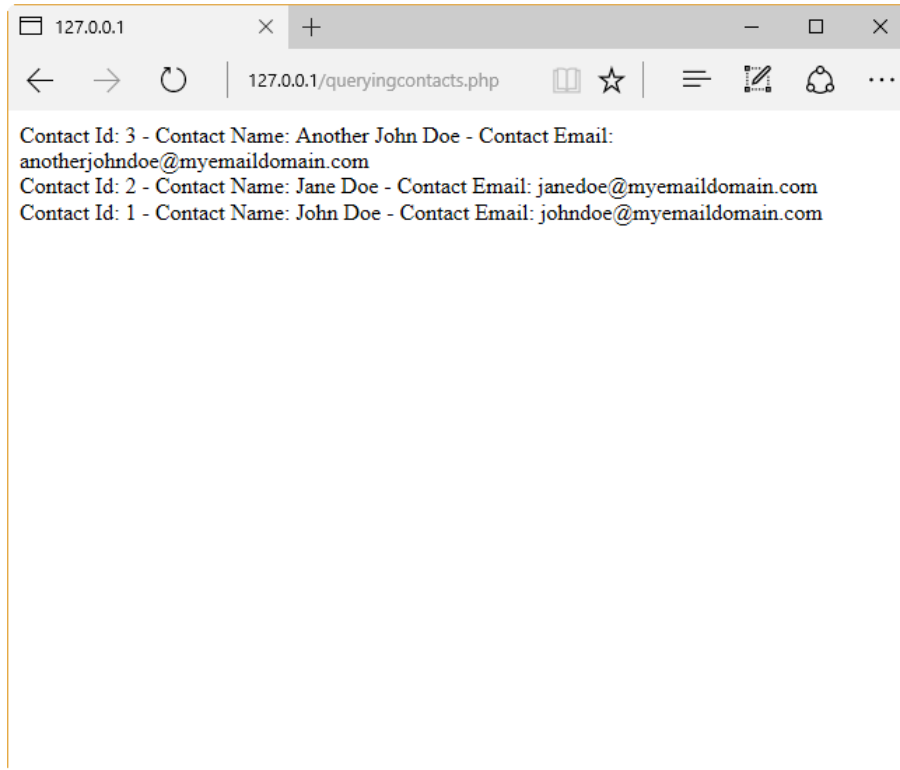
Figure 43: Displaying Contacts in the Web Browser

Displaying contacts in a webpage

In this section we're going to use data from the contacts table to display an HTML table in the web browser. The following code should be saved in a file named **contactswebpage.php**.

Code Listing 41: Displaying Contacts in an HTML Table

```php
<?php

  $dbhost = 'localhost';
  $dbuser = 'root';
  $dbpass = 'mypassword';
  $database = 'contactinfo';
$mysqli = new mysqli($dbhost, $dbuser, $dbpass, $database);

if ($mysqli->connect_errno) {
    echo "We're sorry. The website can not connect to the database <br />";
    echo "Error: MySQL connection failed: <br />";
    echo "Errno: " . $mysqli->connect_errno . "<br />";
    echo "Error: " . $mysqli->connect_error . "<br />";

    exit;
}
```

```php
$sql = "SELECT contacts.* FROM contacts ORDER BY contacts.name";
$resultset = $mysqli->query($sql);

echo "<!DOCTYPE html>\n<html>\n";
echo "<title>Displaying Contacts List</title>\n";
echo "</head>\n";
echo "<body>\n";

if ($resultset->num_rows > 0)
{
   echo "<section>\n";
   echo "<div style=" . '"color:#FFFFFF; background-color:#5F5F5F; text-
align: center;"' . ">\n";
   echo "<h3>OUR CONTACT LIST</h3>\n";
   echo "</div>\n";
   echo "<div>\n";
   echo "<table width=100%>\n";
   echo "<thead>\n";
   echo "<tr><th style=" . '"color:#FFFFFF; background-color:#5F5F5F; text-
align: center;"' . ">ID</th>\n";
   echo "<th style=" . '"color:#FFFFFF; background-color:#5F5F5F; text-
align: center;"' . ">Contact Name</th>\n";
   echo "<th style=" . '"color:#FFFFFF; background-color:#5F5F5F; text-
align: center;"' . ">Contact Email</th>\n</tr>\n</thead>\n";
   echo "<tbody>\n";
      while ($datarow = $resultset->fetch_assoc())
      {
            echo
"<tr>\n<td>".$datarow["ID"]."</td><td>".$datarow["NAME"]."</td>
<td>".$datarow["EMAIL"]."</td>\n</tr>\n";
      }
   echo "</tbody>\n";
   echo "</table>\n</div>\n";
   echo "</section>\n";
}
else
{
   echo "<section>\n<p>No Contacts available</p>\n</section>";
}

echo "<footer>\n<div style=" . '"color:#FFFFFF; background-color:#5F5F5F;
text-align: center;"' . ">\n<p>Copyright (C)2016 All PHP Web
Developers</p>\n</div>\n</footer>\n";
echo "</body>\n</html>\n";

$mysqli->close();

?>
```

This code starts making a connection to MySQL in **localhost**, as the previous data access code samples did. If the connection fails, the script ends. Otherwise, the script queries the contacts table using the **query** method of the **mysqli** class. Data returned by the query is stored in the $**resultset** variable. After that, the script begins to create the HTML document that will be displayed in the web browser. In this case, the HTML **head** and the beginning of the **body** section are sent as a part of the response from the web server. The next part of the HTML document depends on the number of rows returned by the **query** method. If no rows are returned, a paragraph with the sentence "No Contacts available" is placed in the HTML document; otherwise, an HTML table is created using the contents of all data rows in the $**resultset** variable. Every data row corresponds to an HTML table row. The script ends by creating the **footer** section of the webpage and closing the connection to MySQL.

Now, if we type **http://127.0.0.1/contactswebpage.php** into the address bar of the web browser, we should get the following output.
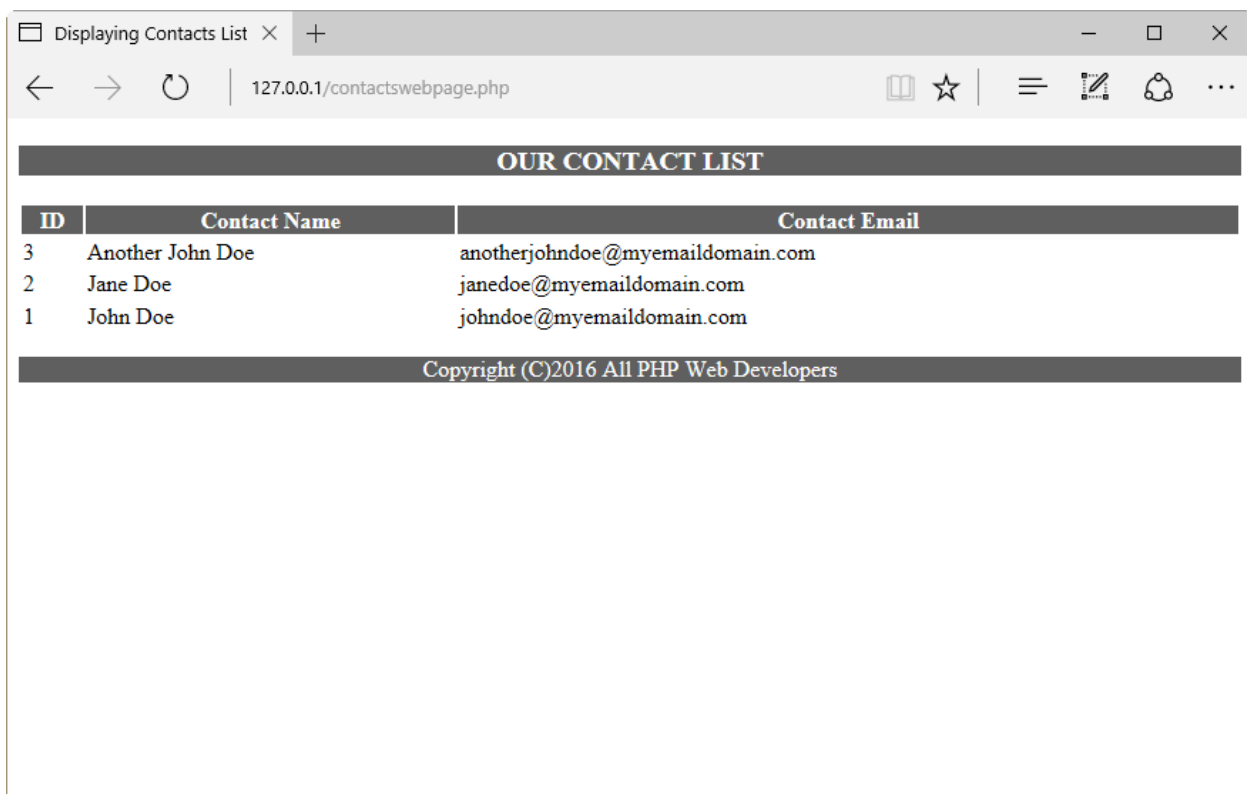
Figure 44: Contacts Displayed in an HTML Table

The HTML that is generated by the PHP script is the following:

Code Listing 42: HTML Dynamically Generated Code

```
<!DOCTYPE html>
<html>
<title>Displaying Contacts List</title>
</head>
<body>
```

```
<section>
<div style="color:#FFFFFF; background-color:#5F5F5F; text-align: center;">
<h3>OUR CONTACT LIST</h3>
</div>
<div>
<table width=100%>
<thead>
<tr><th style="color:#FFFFFF; background-color:#5F5F5F; text-align:
center;">ID</th>
<th style="color:#FFFFFF; background-color:#5F5F5F; text-align:
center;">Contact Name</th>
<th style="color:#FFFFFF; background-color:#5F5F5F; text-align:
center;">Contact Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td><td>Another John Doe</td>
<td>anotherjohndoe@myemaildomain.com</td>
</tr>
<tr>
<td>2</td><td>Jane Doe</td> <td>janedoe@myemaildomain.com</td>
</tr>
<tr>
<td>1</td><td>John Doe</td> <td>johndoe@myemaildomain.com</td>
</tr>
</tbody>
</table>
</div>
</section>
<footer>
<div style="color:#FFFFFF; background-color:#5F5F5F; text-align: center;">
<p>Copyright (C)2016 All PHP Web Developers</p>
</div>
</footer>
</body>
</html>
```

This was created dynamically at the server side. This means that this code will grow larger as the number of contact data rows increases in the table.

Chapter summary

The purpose of this chapter was to explain how to perform input and output operations with files using PHP, and how to connect to a MySQL database to insert or retrieve data.

PHP provides a series of functions that help us to manipulate files by doing operations such as opening, reading, writing, and closing a file. The functions **fopen()**, **filesize()**, **fread()**, and **fclose()** should be used together, in order to read the contents of a file. If we want to write text to a file, we need to use **fwrite()** function instead of **fread()**.

As explained in Chapter 1, PHP supports a wide range of Database Management Systems (RDBMS), and MySQL is the database system most-often used in conjunction with PHP. PHP 7 includes an extension named **mysqli** (MySQL improved), which allows you to access MySQL 4.1 and above. For the purposes of this book, using MySQL with PHP requires that you have an active instance of MySQL installed in the computer used as a web server and have the MySQL Workbench utility installed.

The exercises discussed in this chapter relied on a database named **contactinfo**, which contains a single table to save our contacts information. We used MySQL Workbench utility to create this database. After that, a series of exercises to insert data and query the contacts table were explained. These exercises used the **mysqli** extension through a class also named **mysqli**. This class works in the following way: the constructor **mysqli()** creates a connection to a MySQL server and uses the property **connect_errno** to inform if the connection was successful. In case of success, we can use the **query()** method to insert or retrieve data. Also, we can use parameterized SQL sentences through the **statement** object, binding parameters to data variables with the **bind_param()** method.

Finally, we explained a code sample in order to create an HTML table with the contents of the contacts table.

Chapter 6 A Contact List Website

The purpose of this chapter is to gather all the themes discussed previously, and turn them into a simple contact list website using the contacts database. The final result should look like the following figure.

Figure 45: The Personal Contact List Website

As noticed in the previous design, the website homepage has been divided into four sections: a header, which shows the name of the website; a toolbar, which holds a button that allows you to add contacts to the table; a data table, which displays all contacts stored in the database; and a footer, which displays some copyright information. Also, a **Close** button should be displayed at the right side of the header section.

This kind of design suggests a complex programming. Coding this website into a single file seems very impractical and hard to maintain. So, we're going to employ the file inclusion technique discussed in Chapter 4. As a result, we're going to create a main program file and one program file for each section described in the design. We're also going to code database connections in another separate file.

Website entry point: index.php

During PHP deployment, which was explained in Chapter 2, we defined a default document named **index.php**. In other words, we established that document as one of the entry points for our deployed website, so every time we type http://127.0.0.1 into the address bar of the web browser, the server executes **index.php** automatically, if it exists. So, for the purposes of the exercise detailed in this section, we're going to save the code for our main file as **index.php**.

Creating a basic HTML structure

The first thing we're going to program is a basic HTML document structure. A file named **contactswebsitehtmlsections.php** will store a couple of functions to create this structure.

Code Listing 43: contactswebsitehtmlsections.php

```php
<?php

  function GetHtmlHeader()
  {
        $result = "<!DOCTYPE html>\n<html>\n<head>\n<title>Personal Contact
List</title>\n";
        $result .= '<link rel="stylesheet" type="text/css"
href="css/contactform.css">';
        $result .= '<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js"></sc
ript>';
        $result .= '<script src="js/contactform.js"></script>';
        $result .= "\n</head>\n<body>\n";
        $result .= '<div class="container">' . "\n";

        return $result;
  }

  function GetHtmlFooter()
  {
      $result = "</div>\n</body>\n</html>";
    return $result;
  }

?>
```

The **GetHtmlHeader()** and **GetHtmlFooter()** functions of this code return each one a string, with the opening and closing tags needed to create an empty HTML document. Now, we're going to use this file in **index.php**, as shown in the following code sample.

Code Listing 44: index.php

```php
<?php

  require("contactswebsitehtmlsections.php");

  echo GetHtmlHeader();
  echo GetHtmlFooter();

?>
```

Note the use of the **require()** function. This function includes the contents of the **contactswebsitehtmlsections.php** file into **index.php**. After that, the **echo** statement calls each one of the functions coded in the included file. Now, if we type **http://127.0.0.1** in the address bar, the web browser will show a blank page. However, if we ask for the source code of the webpage, we will look at the following HTML document.

Code Listing 45: Basic HTML Document Structure

```html
<!DOCTYPE html>
<html>
<head>
<title>Personal Contact List</title>
</head>
<body>
<div class="container">
</div>
</body>
</html>
```

Creating the website header

Now, we're going to create the website header. This header will display the website's title and a button to close the web browser's window. The code for doing this will be saved in a file named **websiteheader.php**.

Code Listing 46: websiteheader.php

```php
<?php

  function GetWebSiteHeader()
  {
```

```
        $result = '<header style="width:100%; height:50px; background-color:
#8D8D8D; color: white;">';
        $result .= '<div style="display: inline-block; width:90%; text-
align: center;"><h1 style="margin: 0px 0px 0px 0px;">Personal Contact List
Website</h1></div>';
        $result .= '<div style="display: inline-block; width:10%;"><a
href="#" onclick="window.close();return false;"><img
src="images/closebutton.png" alt="Close" height="36" width="36"
/></a></div></header>';
        return $result;
    }

?>
```

This code constructs a string with the **`<header>`** HTML tag. This tag is used to define a header section into our webpage. Let's look at the **`style`** attribute defined in the tag. This attribute controls the way any element is displayed in the web browser by assigning to it a series of properties enclosed in double quotes. In this case, the **`width`** property tells the web browser that the header will cover the entire width of the browser window (100%). The **`height`** property tells the browser that the header will be 50 pixels in height (50px). The **`background-color`** property sets a gray scale color for the header background, and the **`color`** property sets a white color for all text displayed.

According to the design displayed at the beginning of this chapter, the website's header will display a title for the site and a button to close the web browser's window. In order to do this, the **`header`** section needs to be divided in two subsections. The **`<div>`** tag is used to accomplish this task.

The first **`<div>`** tag creates the subsection in which the title will be displayed. The **`style`** attribute for this subsection tells the browser that the title will cover 90% of the web browser's window width, the text will be centered within the boundaries of the subsection, and the subsection belongs to the same row in which the close button will be displayed (**`display: inline-block;`**).

The second **`<div>`** tag will show the Close button. For this purpose, an image named **closebutton.png** saved in the **images** folder (located into the website's root folder) is used as a hyperlink (**`<a>`** tag). Two lines of code are assigned to the **`onclick`** hyperlink event's attribute. This code will be executed when the user clicks over the Close button's image. The first line calls the **`close()`** method of the window object (the window object is equal to the web browser's window), and the second line of code returns a **`false`** value in order to prevent the browser from jumping to an nonexistent link.

Creating the website toolbar

After the **`header`** section, the website should display a toolbar with one command button: Add Contact. By clicking on this button, the user can display a dialog box in order to insert data for a new contact in the database.

```php
<?php

    function GetWebSiteToolbar()
      {
            $result = '<section style="margin-top: 5px; height: 60px;"><div
class="container">';
            $result .= '<button type="button" style="height: 56px;"
onclick="div_show(); return false;">Add Contact</button>';
            $result .= '</div></section>';
            return $result;
      }
?>
```

The code for creating the toolbar begins with a **`<section>`** tag. In HTML, this tag is used to define sections in a document, such as headers, footers, and of course, custom sections such as a toolbar. The **`style`** attribute defined in the **`<section>`** tag will use the **`margin-top`** property to apply a five-pixel margin top, starting from the end of the previous section (the website's header). The **`height`** property assigns a 60-pixel height to the **`<section>`**.

Now, to display the command button in the toolbar, we will create a subsection using the **`<div>`** tag. Then, the command button is created by using the **`<button>`** tag along with the **`type="button"`** attribute. The button is set with a 56-pixel height using the **`style`** attribute along with the **`height`** property, and a function to handle the click event is assigned with the **`onclick`** attribute.

Creating the website footer

Even though the data table section is placed before the website footer, we're going to review the code for website footer creation first, since the data table section requires the most of our attention.

The purpose of the website footer is to display a copyright message. The footer was considered in the design in order to include all main elements for a webpage. We're going to save this code in a file named **websitefooter.php**.

Code Listing 48: websitefooter.php

```php
<?php

    function GetWebSiteFooter()
    {
        $result = '<footer style="clear: both; position: fixed; left: 0;
bottom: 0; height: 50px; margin-top: -50px; width: 100%; background-color:
#8D8D8D; color: white;">';
```

```php
      $result .= '<div style="display: inline-block; width:80%; text-align:
 left;"><p style="margin: 12px 0px 0px 0px;">Copyright (C)2016 All PHP Web
 Developers</p></div>';
      $result .= "</footer>";

      return $result;
    }

?>
```

Likewise, in the website header's code, we have a **<footer>** tag to define the webpage's footer section. Looking at the design displayed at the beginning of this chapter, we can see that the footer for the webpage should be placed at the bottom of the browser's window. To accomplish this task, we should assign values to a series of properties and assign them to the **style** attribute. These properties are:

- **clear** – Prevents floating elements beside the declaring element. In this case, the value avoids floating elements at both the left and right sides of the **<footer>** section.
- **position** – Specifies the method used for positioning an element. The fixed value positions the **<footer>** section relative to the browser window.
- **left** – Specifies the left coordinate at which to place the element.
- **bottom** – Specifies the y coordinate at which to place the element, starting from the bottom of the browser window. In this case, **0** indicates that the footer will be drawn starting at the bottom of the window.
- **height** – Defines a 50-pixel height for the footer section.
- **margin-top** – The negative value is equal to the footer height, and ensures the footer section is always pulled up, starting from the bottom of the browser's window.

To display the copyright message, we use a **<div>** section that covers the 80 percent of the **<footer>** section. A top margin of 12 pixels is assigned to the message's paragraph.

At this point, the code for **index.php** should look like the following sample.

Code Listing 49: index.php, So Far

```php
<?php

  require("contactswebsitehtmlsections.php");
  require("websiteheader.php");
  require("websitetoolbar.php");
  require("websitefooter.php");

  echo GetHtmlHeader();
  echo GetWebSiteHeader();
  echo GetWebSiteToolbar();
  echo GetWebSiteFooter();
  echo GetHtmlFooter();

?>
```

If we type **http://127.0.0.1** into the address bar of the web browser, the output should look like the following figure.
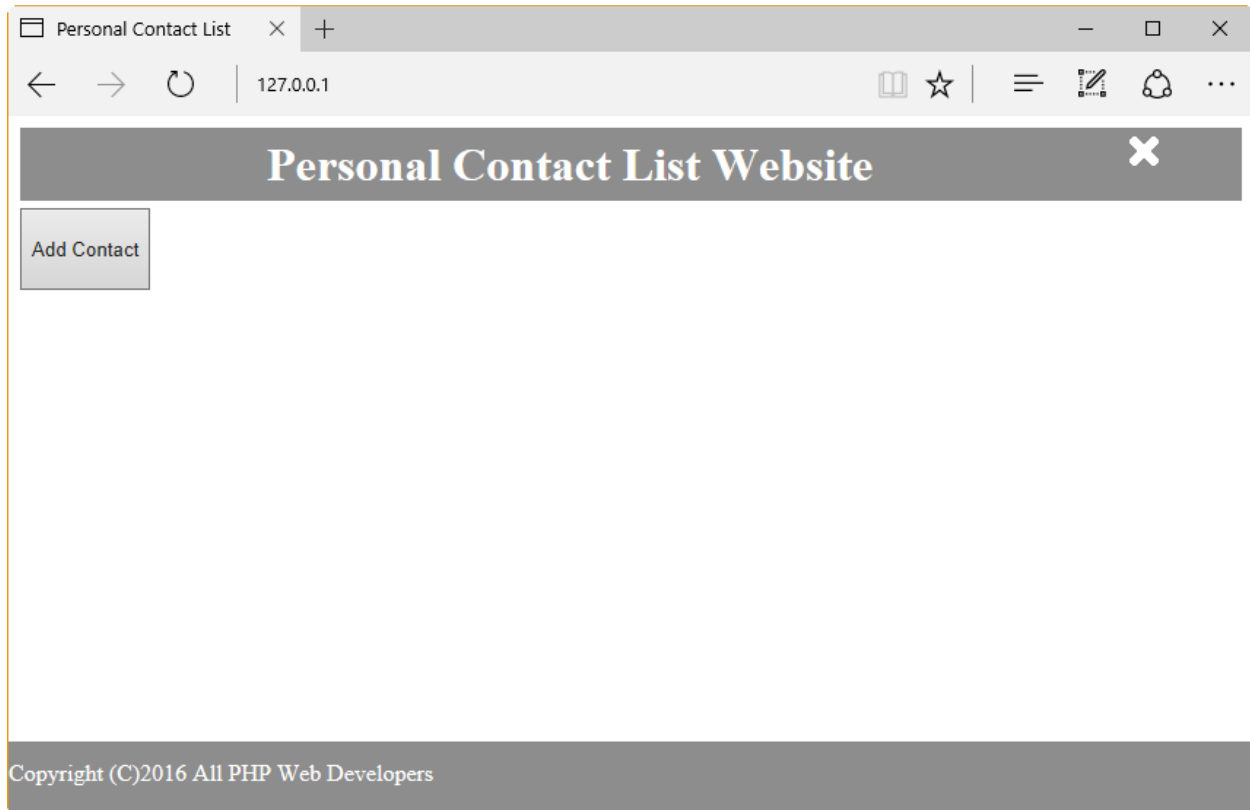
Figure 46: Personal Contact List Website, So Far

Note: The X icon in the upper-right corner is the image file closebutton.png, which you must generate yourself or get from the files that accompany this e-book.

Creating the data table section

The data table section displays contacts info using a HTML table. To accomplish this task, the following requirements should be fulfilled:

- Create a `<section>` tag to contain the data table section.
- Create a `<div>` tag to contain the HTML table.
- Create the HTML table using the `<table>` tag.
- Create the HTML table headers using the `<th>` tag.
- Connect to the contacts database using the `mysqli` class.
- Perform a SQL `SELECT` query using the `query()` method.
- Iterate through all rows in the returned dataset to create each HTML table row using `<tr>` and `<td>` tags.

We're going to create a script to perform all these steps. The script will be saved in a file named **datatablesection.php**.

```php
<?php

    require("contactsquery.php");

    function GetDataTableSection()
    {
        $result  = "<section>\n";
        $result .= '<div style="width: 100%; height: 90%; overflow: auto;">';
        $result .= "\n";
          $result .= '<table width="100%">';
          $result .= "\n";
        $result .= '<tr style="background-color: #E1E1E1; color: #000000;
text-align: center;">';
          $result .= "\n";
        $result .= '<th width="5%">ID</th>';
        $result .= "\n";
        $result .= '<th width="35%">Name</th>';
        $result .= "\n";
        $result .= '<th width="35%">Email address</th>';
        $result .= "\n";
        $result .= '<th width="25%">Phone number</th>';
        $result .= "\n</tr>\n";

          $resultset = ContactsDataSet();

          if ($resultset != null)
          {
           if ($resultset->num_rows > 0)
            {
                    $rownumber = 0;

                while ($datarow = $resultset->fetch_assoc())
                 {
                            $rownumber++;

                            if ($rownumber % 2 == 0)
                            {
                            $result .= '<tr style="background-color: #E1E1E1;
color: #000000;">';
                            }
                            else
                            {
                            $result .= '<tr style="background-color: #FFFFFF;
color: #000000;">';
                            }
```

```php
                        $result .= '<td style="text-align: right;">' .
$datarow["ID"] . '</td>';
                        $result .= '<td>' . $datarow["NAME"] . '</td>';
                        $result .= '<td>' . $datarow["EMAIL"] . '</td>';
                        $result .= '<td>' . $datarow["PHONENUMBER"] .
'</td>';

                        $result .= '</tr>';
                        $result .= "\n";
                }
            }
        }

    $result .= "</table>\n";
    $result .= "</div>\n";
    $result .= "</section>\n";

    return $result;
    }

?>
```

The first line of Code Listing 50 includes a file named **contactsquery.php**. This file contains a script with a function named **ContactsDataSet()**, which connects to the contacts database and returns a data set that is used to populate the table. After file inclusion, the script defines a function called **GetDataTableSection()**. This function will return the necessary HTML code to create the table that will display all rows from the contacts database. The table is built within the **<section>** and the **<div>** tags. The **<section>** tag delimits the **DataTable** section into the webpage. The **<div>** tag is used to contain the table. The **overflow** property defined in the **style** attribute tells the web browser that scroll bars should be displayed if the contents of the table are larger than the section height (**overflow: auto**). Then, the table headers are created, and the script calls the **ContactDataSet()** function, storing the result in the **$resultset** variable. If a null value is returned from the function, or if the **$resultset** variable contains no rows, table population does not happen. Otherwise, the script iterates through all rows and fills the table.

Code Listing 51: contactsquery.php

```php
<?php
  function ContactsDataSet()
  {
    $dbhost = 'localhost';
    $dbuser = 'root';
    $dbpass = 'userpassword';
    $database = 'contactinfo';
    $mysqli = new mysqli($dbhost, $dbuser, $dbpass, $database);

    if ($mysqli->connect_errno) {
            return null;
```

```
    }

    $sql = "SELECT contacts.* FROM contacts ORDER BY contacts.name";
    $resultset = $mysqli->query($sql);
    $mysqli->close();
    return $resultset;
  }
?>
```

At this point, **index.php** should look like the following code sample.

Code Listing 52: index.php, So Far

```php
<?php

  require("contactswebsitehtmlsections.php");
  require("websiteheader.php");
  require("websitetoolbar.php");
  require("datatablesection.php");
  require("websitefooter.php");

  echo GetHtmlHeader();
  echo GetWebSiteHeader();
  echo GetWebSiteToolbar();
  echo GetDataTableSection();
  echo GetWebSiteFooter();
  echo GetHtmlFooter();

?>
```

Creating the Add New Contact dialog box

Now, we're going to add the HTML code needed to create a dialog box to add a new contact.
This dialog box should look like the following figure.
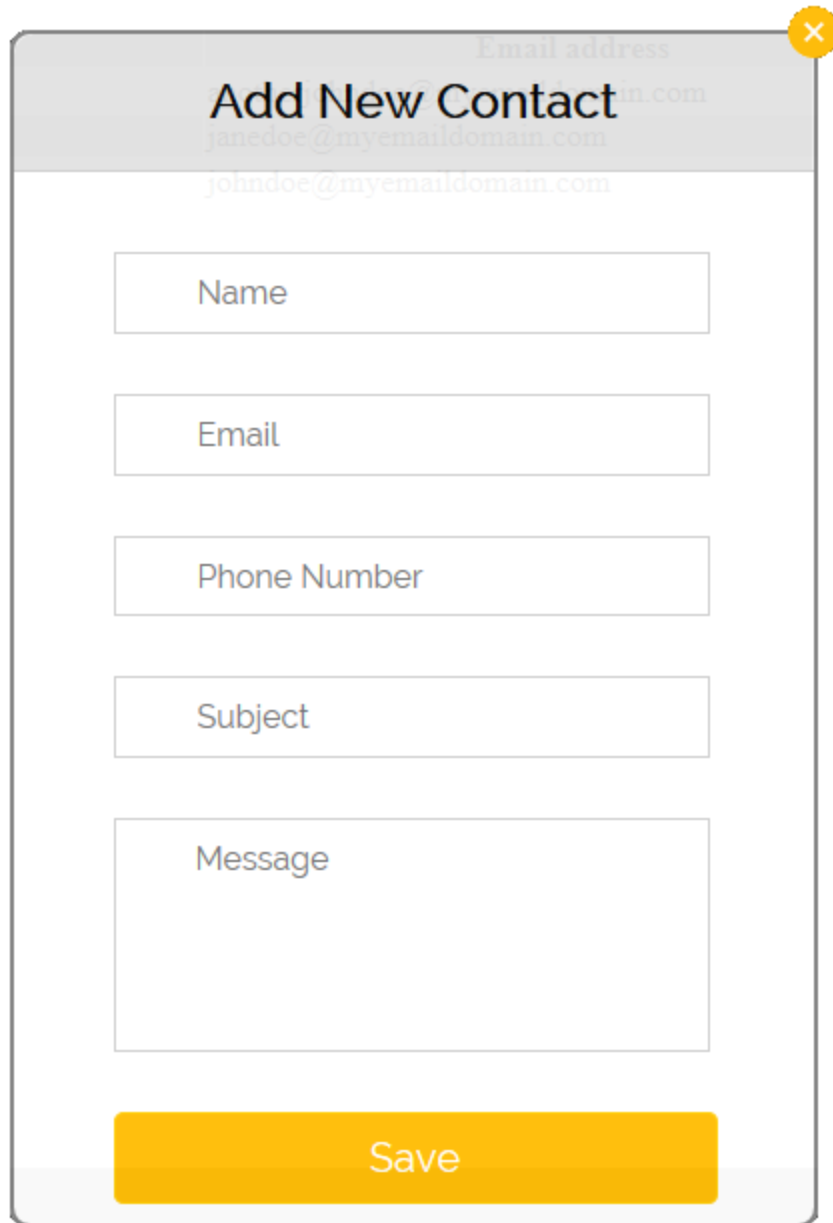
Figure 47: Add New Contact Dialog Box

The code for including this dialog box should be saved in a file named **getcontactform.php**.

Code Listing 53: getcontactform.php

```php
<?php

  function GetContactForm()
  {
      $result = '<div id="newcontact" style="display: none;">';
      $result .= "\n";
    $result .= '<div id="popupContact">';
```

```php
        $result .= "\n";
    $result .= '<form action="#" id="form" method="post" name="form">';
        $result .= "\n";
    $result .= '<img id="close" src="images/close.png" onclick
="div_hide()">';
        $result .= "\n";
        $result .= '<h2>Add New Contact</h2>';
        $result .= "\n";
        $result .= '<hr>';
        $result .= "\n";
        $result .= '<input id="name" name="name" placeholder="Name"
type="text">';
        $result .= "\n";
        $result .= '<input id="email" name="email" placeholder="Email"
type="text">';
        $result .= "\n";
        $result .= '<input id="phonenumber" name="phonenumber"
placeholder="Phone Number" type="text">';
        $result .= "\n";
        $result .= '<input id="subject" name="subject" placeholder="Subject"
type="text">';
        $result .= "\n";
        $result .= '<textarea id="message" name="message"
placeholder="Message"></textarea>';
        $result .= "\n";
        $result .= '<button type="button" name = "submit" id="submit"
onclick="savecontact()">Save</button>';
        $result .= "\n";
        $result .= '</form>';
        $result .= "\n";
        $result .= '</div>';
        $result .= "\n";
        $result .= '</div>';
        $result .= "\n";

    return $result;
  }

?>
```

As noticed in the previous code, the dialog box is an HTML form placed into a **<div>** section. The way in which the dialog box is displayed is established by using CSS styles properties. These properties are saved in a file named **css/contactform.css**. The **display: none** property inside the **style** attribute prevents the dialog box from being displayed when we load the website for the first time.

At this point, some JavaScript functions were added to the website in order to manage the **Add Contact** button's click event, the **Save** button's click event, the **Close** button, the validation of data entries, and the process of inserting contact info in the database.

```
function savecontact() {
  if (   document.getElementById('name').value == ""
      || document.getElementById('email').value == ""
        || document.getElementById('subject').value == ""
        || document.getElementById('message').value == ""
        || document.getElementById('phonenumber').value == "")
  {
        alert("You must fill all entries!");
        return;
  }

  document.getElementById('submit').disabled = true;

  $.post("insertcontact.php",
    {
        name: document.getElementById('name').value,
        email: document.getElementById('email').value,
            phone: document.getElementById('phonenumber').value,
            subject: document.getElementById('subject').value,
            message: document.getElementById('message').value
    },
    function(data, status){
        if (data == "OK")
            {
                    window.location.reload();
            }
            else
            {
             alert(data);
            }
    }).fail(function(data,status){alert("Error " + status);});

  document.getElementById('submit').disabled = false;

}

//Function to Display Popup
function div_show() {
document.getElementById('newcontact').style.display = "block";
}
//Function to Hide Popup
function div_hide(){
document.getElementById('newcontact').style.display = "none";
}
```

There are three functions in the previous code. The last two allow you to display or hide the **Add New Contact** dialog box. The trick is simple: to show the dialog box, the **display** property for

the **'newcontact' div** section receives a value of **"block"** after the **div_show()** function is executed. Hiding the dialog box is performed by the **div_hide()** function, by setting the value of the **display** property to **"none"**.

📝 ***Note: The*** *div_show()* ***function is executed when the user clicks over the Add Contact button located at the toolbar, and the*** *div_hide()* ***function is executed when the user clicks on the Close button located at the top-right corner of the Add New Contact dialog box.***

Now, let's review the **savecontact()** function. First, the function checks for values stored in all entries. If one of these entries has no value, the function fires an alert and finishes the execution. Otherwise, the function disables the **Save** button and uses the **$.post** jQuery method to execute the **insertcontact.php** script, passing its name and location as the first parameter of the method. The second parameter of the **$.post** method is an array with the variables, and their values, which will be passed to the script. The third parameter of the **$.post** method is a function that is performed if the execution of the script is successful. This function will be used to know if the contact was added to the database. That happens if the data variable passed to the function has an **"OK"** value. In this case, we use the **window.location.reload()** method to refresh the web browser window and display new data.

The script that inserts contact info in the database is displayed in the following code sample.

Code Listing 55: insertcontact.php

```php
<?php

  $name = $_POST['name'];
  $email = $_POST['email'];
  $phone = $_POST['phone'];
  $subject = $_POST['subject'];
  $message = $_POST['message'];

  $dbhost = 'localhost';
  $dbuser = 'root';
  $dbpass = 'Netdevserver';
  $database = 'contactinfo';
  $mysqli = new mysqli($dbhost, $dbuser, $dbpass, $database);

 if ($mysqli->connect_errno) {
    echo "We're sorry. The website can not connect to the database <br />";
    echo "Error: MySQL connection failed: <br />";
    echo "Errno: " . $mysqli->connect_errno . "<br />";
    echo "Error: " . $mysqli->connect_error . "<br />";

    exit;
 }
```

```php
$sql = "INSERT INTO contacts (name,email,phonenumber,subject,message)
VALUES ('$name', '$email', '$phone','$subject','$message')";

if ($mysqli->query($sql) === TRUE) {
    echo "OK";
} else {
    echo "Error: " . $sql . "<br>" . $mysqli->error;
}

$mysqli->close();

?>
```

This script is similar to the one explained in Chapter 5, except that the values to be inserted in the contacts database are taken from the **$_POST** associative array. In this case, the name of the parameters sent by the **$.post** method are used as keys to retrieve the corresponding value for each parameter. These values are passed to a set of variables used later as a part of the **INSERT** SQL statement. If data insertion is successful, the **echo** statement sends **"OK"** as a response. Otherwise, the response is an error message.

The result: A functional Personal Contact List Website

At the end, we should get a functional website that looks like the following figure.
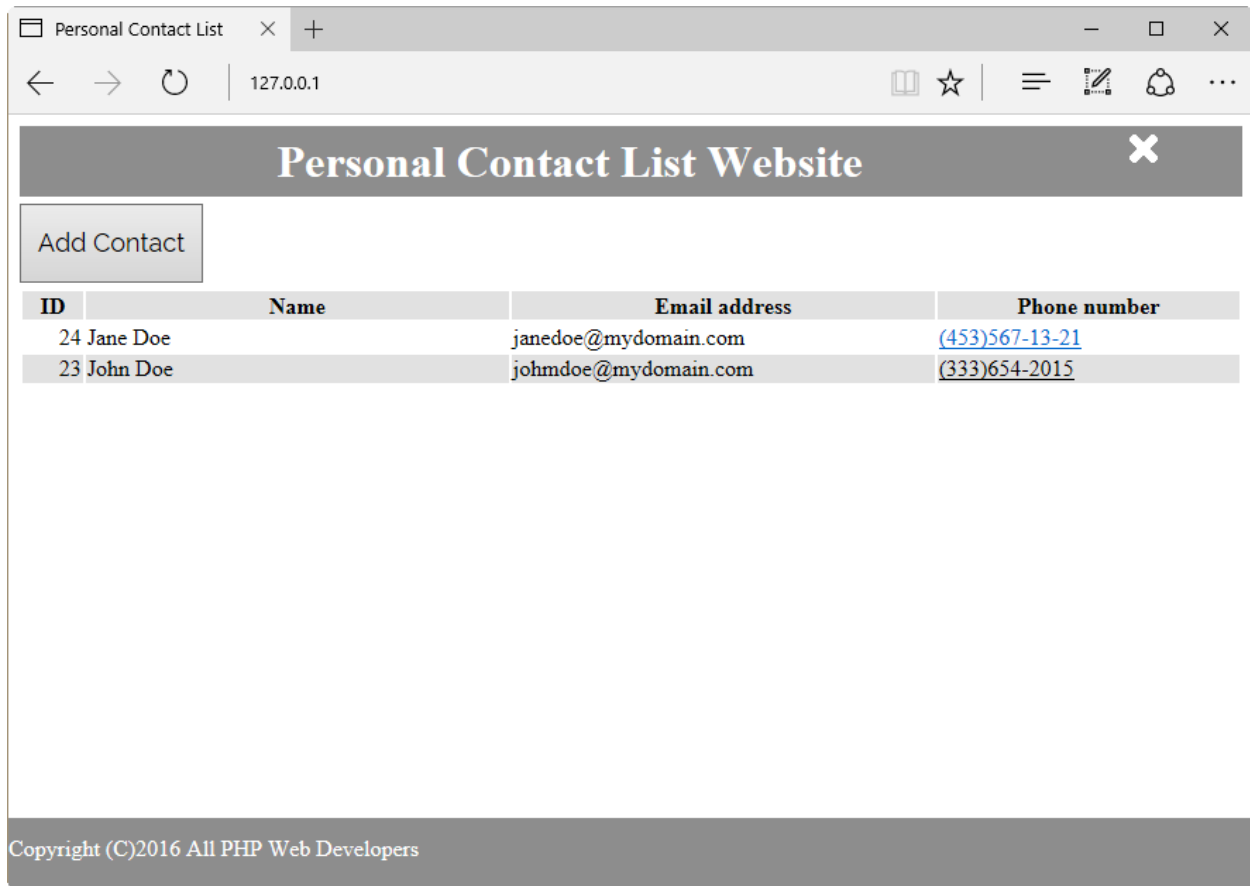
Figure 48: Personal Contact List Website

Chapter summary

The purpose of this chapter was gathering all themes discussed previously to turn them into a simple contact list website. The design for the website divided the homepage into four sections: a header, intended to show the name of the website; a toolbar, which holds a button intended to add contacts to the table; a data table, intended to display all contacts stored in the database; and a footer, which displays some copyright information. Also, a **Close** button was placed at the right side of the header section. This kind of design employs a complex programming, so we used the file inclusion technique, discussed in Chapter 4, to develop this website.

We used the **index.php** document as the website point of entry. Then, we programmed every section of the website's home page in a separate file. After that, we used the `require()` statement to include all those files in **index.php**.

Chapter 7 General Summary

PHP is an open source, general-purpose scripting language oriented for web development. This language was originally created by Ramsus Lerdorf in 1994, and it was known as Personal Home Page / Forms Interpreter. The first version of PHP/FI was released in 1995. Later, PHP/FI 2.0 appeared in 1997. PHP is now supported by The PHP Group, and the PHP acronym stands for PHP: Hypertext Preprocessor. At this time, the current stable release is 7.0.13. The most important topics about PHP are summarized in the following list.

- PHP is an acronym for PHP: Hypertext Preprocessor.
- PHP is an open source language, free to download and use.
- PHP is executed on the server.
- PHP can generate dynamic page content.
- PHP can connect to a wide range of databases.
- PHP can run on various platforms (Windows, Linux, UNIX, Mac OS X, etc.).
- PHP is compatible with almost all web servers used today (IIS, Apache, etc.).
- PHP is easy for newcomers to learn.

PHP can be deployed in a Windows environment using IIS (Internet Information Services) as a web server. To perform this deployment, the following requirements should be fulfilled:

- The computer should have a Windows operating system installed and running.
- IIS should be installed and configured.

To install IIS in a Windows 10 computer, you should go to the **Programs** section of the **Control Panel**, and then click on the **Turn Windows features on or off** link. Next, check the **Internet Information Services** checkbox in the **Windows Features** dialog box, in order to install IIS with the default features for a web server.

Now, to install PHP in the computer, you should download a zip installation package from this location for a 32-bit system, or from this location for a 64-bit system. After the download is complete, you should unpack the zip file in a folder named **C:\PHP**.

To configure PHP, rename the file **php.ini-development** to **php.ini**. Then, you can edit it to adjust some PHP working parameters to comply with IIS requirements.

As a final step, you should add **C:\PHP** to the Path system variable, and open the IIS Manager to set up PHP as the program that will handle all .php web requests coming from any client within the network.

Now, the installation process can be tested by creating a text file named **phpinfo.php** in the website root folder (commonly **C:\inetpub\wwwroot**). The file should contain the following programming code: `<?php phpinfo(); ?>`. Next, type **http://127.0.0.1/phpinfo.php** into a web browser to display the PHP installation info.

PHP programming relies on scripts. A script in PHP is a text file that contains pure PHP programming code, or PHP programming code embedded into HTML, and is executed in the web server.

As in most programming languages, the main way to store data in a PHP program is by using variables, which are identifiers intended to hold data dynamically, so that data stored in variables can change during the execution flow.

Variables in PHP are declared by denoting their names with a leading dollar sign (**$**), and then starting with a letter or underscore. Variables can be converted from one data type to another automatically.

A variable can be available in a certain section of a script, starting from the program in which the variable is declared. This is known as variable scope. PHP has the following variable scopes: *local*, for variables that are available only in the program where they are declared; *global*, for variables that can be accessed in any part of the executed program; *function parameters*, which are variables available within the function where they're employed; and *static*, which are variables declared inside functions that keep their values between every function call.

PHP also allows you to use constants. A constant is an identifier that holds a simple value and cannot be changed during the execution of the script. Constants' identifier names are case sensitive. The best practices in PHP dictate that constant names should be uppercase. Constants are defined by the **define()** function.

PHP uses expressions to perform calculations. A set of symbols are used in order to perform these calculations. These symbols are called *operators,* and the identifiers declared between the operators are called *operands*. PHP has the following types of operators: arithmetic operators, which are used to perform operations with numbers; comparison operators, which are used to check if criteria between two operands are met; logical operators, which are employed to get a true or a false value depending the logical state of two operands; assignment operators, which are employed to store the value of an expression into an operand; and conditional operators, which are employed to perform an inline decision making.

When an expression contains several operators, calculations are performed following a strict order. This order is known as operator precedence. To explain this precedence, we can classify operators into the following categories: unary, which are operators preceding a single operand; binary, which take two operands; ternary, which take three operands, evaluating either the second or the third depending on the value of the first one; and assignment operators, which store a value into an operand. Operator precedence is rather complicated. Common operators in an expression are executed in the following order: increment and decrement, unary, multiplicative and division, addition and subtraction, relational, equality, bitwise, ternary, assignment, logical **AND**, logical **XOR**, logical **OR**.

In PHP, we can use sequences of characters stored in variables or directly placed at the right of a statement. These sequences are called strings. Strings can be delimited either by single or double quotes. PHP treats strings in a different way depending on how they're delimited. Every PHP statement is considered a single-quoted string literal, but when variable names are present in a double-quoted string, PHP replaces the name of the variable with its contents.

When we need to store several values of similar type, PHP provides us with a data structure known as an array. We can use this structure instead of declaring many variables. In PHP we have the following kind of arrays: numeric arrays, which store values that can be accessed using a numeric index; associative arrays, which use strings as indexes and associate them to the values stored; and multidimensional arrays, which contain one or more arrays accessing their values using multiple indexes. An array can be created using the `array()` function, or by declaring a variable followed by an index enclosed in brackets.

PHP provides a set of statements to take a course of action based on a condition. These statements are known as decision-making statements, and they are: `if … elseif … else`, which executes a code block when the condition after `if` statement is true, or the code block within the `elseif` statement, if the condition of `if` statement is `false`, and the condition of the `elseif` statement is `true`, or executes the code within the `else` statement in case both conditions are `false`; and the `switch` statement, which executes a block of code depending on a comparison of equality for an expression with a series of values, placed each one after a `case` clause, which also contains the code to be executed if the expression value is equal to the value associated to this particular `case` clause.

Looping statements allow us to execute a particular code block repeatedly, either while a certain condition is met, a specific number of times, or until a series of elements from a data structure have been all iterated. These looping statements are: `for`, which loops through a code block a specified number of times; `while`, which loops through a code block while a certain condition is met; `do … while`, which loops through a code block once, and repeats the execution as long as the condition established is true; and `foreach`, which loops through a code block many times as elements exist in an array. PHP provides two special keywords to be used within a loop: `break`, which terminates the execution of a loop prematurely; and `continue`, which halts the execution of a loop and starts a new iteration.

PHP lets the user create functions. A function is a piece of code which receives data by using a set of variables named parameters, then processes this data and returns a value. In PHP, we have user-defined and built-in functions. A user-defined function is created by the developer by using the reserved keyword `function`, followed by the name of the function.

When you need to pass data to a function, you should use a series of identifiers named parameters. These are a series of identifiers declared after the function name, enclosed in parentheses. You can declare as many parameters as you need. All these parameters will act as variables within the function. A function can return a value to the calling program employing the `return` statement.

We can set function parameters to have a default value, in case the calling program doesn't pass any value to any of them. Default values are defined by placing the desired value at the right side of the parameter name, leading by an equal assignment operator (=).

A function can be called dynamically by storing its name into a string variable. Then, we can use this variable as we would the function name itself.

PHP has a large set of built-in functions that can be classified in categories. The most relevant categories are: array functions, which allow the developer to interact with and manipulate arrays; date and time functions, which get the date and time from the server in which scripts are running; string functions, which allow the developer to manipulate strings; character functions, which check whether a string or character falls into certain class; file system functions, which access and manipulate the file system; and directory functions, which are used to manipulate directories.

PHP allows code reusing, which is important for maintaining complex applications with minimal effort. Code reusing is handled by means of file inclusion. File inclusion is the mechanism used to insert the content of a PHP file into another one, and it is performed by two functions: `include()`, which copies all the text in the specified file into the script, generating a warning message when a problem occurs; and `require()`, which is similar to `include()`, except that it halts script execution when a problem occurs.

PHP provides a series of functions that help us to manipulate files by doing operations such as opening, reading, writing and closing a file. The functions `fopen()`, `filesize()`, `fread()`, and `fclose()` should be used together, in order to read the contents of a file. On the other hand, if we want to write text to a file, we need to use `fwrite()` function instead of `fread()`.

PHP also supports a wide range of Database Management Systems (RDBMS). MySQL is the database system most commonly used in conjunction with PHP. PHP 7 includes an extension named `mysqli` (MySQL improved) which allows you to access MySQL 4.1 and above. For the purposes of this book, using MySQL with PHP requires that you have an active instance of MySQL installed in the computer used as a web server, and have the MySQL Workbench utility installed.

The exercises in this book used a database named **contactinfo**, which contains a single table to save contact information. We used the MySQL Workbench utility to create this database. The exercises explained in this book inserted data and queried the contacts table belonging to the database. These exercises used the `mysqli` extension through a class also named `mysqli`. This class works in the following way: the constructor (`mysqli()`) creates a connection to a MySQL server and uses the property `connect_errno` to inform if the connection was successful. In case of a successful connection, we can use the `query()` method to insert or retrieve data. Parameterized SQL sentences are also allowed through the `statement` object, which binds parameters to data variables using the `bind_param()` method.

Finally, all themes discussed in this book were gathered and turned into a simple contact list website. The website divided the homepage in four sections: a header, intended to show the name of the website; a toolbar, which holds a button intended to add contacts to the table; a data table, intended to display all contacts stored in the database; and a footer, which displays some copyright information. We also placed a **Close** button at the right side of the header section. This kind of design employs a complex programming, so we used the file inclusion technique, discussed in Chapter 4, to develop it. We used the **index.php** document as the website point of entry. Then, we programmed every section of the website's homepage in a separate file and used the `require()` statement to include all those files in **index.php**.

General Conclusions

With a large community of developers, and a lot of blogs and websites dedicated to discussing it, PHP is by far the easiest way to enter into the web development world. Its relative simplicity for writing programs (scripts) allows the newbies to create code in hours, and for the skilled programmer, it offers the power of the object-oriented paradigm and structured programming in one package. Also, its multiplatform compatibility lets us to deploy our applications on a Windows or a Linux web server, with no changes in code. Furthermore, it's open source, so we can use the language in any kind of projects, even commercial projects.

Around the time my company adopted PHP as our primary web programming language, we also explored other technologies, such as ASP.NET. We realized that the learning curves for those technologies were huge, and there was a lack of experienced developers in our region. Because the time to market for our web software products was so short, we chose PHP—and we didn't regret it. Our first application prototype was created in 20 minutes using a simple text editor, and our first product was deployed in almost thirty days, so we can say that this decision was a great success.

PHP has been constantly upgrading to increase its performance and its features. Today, there's a large number of frameworks for PHP that make coding easier and more manageable, and its OOP style approach has been enhanced, almost like traditional OOP languages such as C# or Java. The number of database extensions has also been increasing in the latest versions of PHP, making data access easy. Many important platforms, such as MailChimp®, use PHP for processing thousands of web requests, creating a huge number of dynamic webpages, handling thousands of user logins, and even managing millions of email messages.

Although today's web development is a combination of several technologies, many programmers around the world still choose PHP as their primary programming language, and I don't think this will change anytime soon.