CHAPTER 17

Importing Files

The same code often needs to be called on multiple pages. This can be done by first placing the code inside a separate file and then including that file using the include statement. This statement takes all the text in the specified file and includes it in the script, just as if the code had been copied to that location. Just like `echo`, `include` is a special language construct and not a function, so parentheses should not be used.

```
<?php
include 'myfile.php';
?>
```

When a file is included parsing changes to HTML mode at the beginning of the target file and resumes PHP mode again at the end. For this reason any code inside the included file that needs to be executed as PHP code must be enclosed within PHP tags.

```
<?php
// myfile.php
?>
```

Include path

An include file can either be specified with a relative path, an absolute path or without a path. A relative file path will be relative to the importing file's directory, and an absolute file path will include the full file path.

```
// Relative path
include 'myfolder\myfile.php';

// Absolute path
include 'C:\xampp\htdocs\myfile.php';
```

When a relative path or no path is specified, include will first search for the file in the current working directory, which defaults to the directory of the importing script. If the file is not found there, include will check the folders specified by the include_path[1] directive defined in php.ini before failing.

```
// No path
include 'myfile.php';
```

In addition to include there are three other language constructs available for importing the content of one file into another: `require`, `include_once` and `require_once`.

Require

The `require` construct includes and evaluates the specified file. It is identical to `include`, except in how it handles failure. When a file import fails `require` will halt the script with an error, whereas `include` will only issue a warning. An import may fail either because the file is not found or because the user running the web server does not have read access to it.

```
require 'myfile.php'; // halt on error
```

Generally it is best to use `require` for any complex PHP application or CMS site. That way the application will not attempt to run in case a key file is missing. For less critical code segments and simple PHP websites `include` may suffice, in which case PHP will go on and show the output even if the included file is missing.

Include_once

The `include_once` statement behaves like `include`, except that if the specified file has already been included it will not be included again.

```
include_once 'myfile.php'; // include only once
```

Require_once

The `require_once` statement works like `require`, but will not import a file if it has already been imported before.

```
require_once 'myfile.php'; // require only once
```

The `include_once` and `require_once` statements may be used instead of `include` and `require` in cases where the same file might be imported more than once during a particular execution of a script. This avoids errors caused by, for example, function and class redefinitions.

[1] http://www.php.net/manual/en/ini.core.php#ini.include-path

Return

It is possible to execute a return statement inside an imported file. This will stop the execution and return to the script that called the file import.

```
<?php
// myimport.php
return 'OK';
?>
```

If a return value is specified, the import statement will evaluate to that value just as a normal function.

```
<?php
// myfile.php
if ((include 'myimport.php') == 'OK')
  echo 'OK';
?>
```

Auto load

For large web applications the number of includes required in every script may be substantial. This can be avoided by defining an `__autoload` function. This function is automatically invoked when an undefined class or interface is used in order to try to load that definition. It takes one parameter, which is the name of the class or interface that PHP is looking for.

```
function __autoload($class_name){
  include $class_name . '.php';
}

// Attempt to auto include MyClass.php
$obj = new MyClass();
```

A good coding practice to follow when writing object-oriented applications is to have one source file for every class definition and to name the file according to the class name. Following this convention the `__autoload` function above will be able to load the class, provided that it is in the same folder as the script file that needed it.

```
<?php
// myclass.php
class MyClass {}
?>
```

If the file is located in a subfolder the class name can include underscore characters to symbolize this. The underscore characters would then need to be converted into directory separators in the `__autoload` function.