CHAPTER 1

Introduction and Setting Up

Before you get started with learning PHP, you will need to have a working development environment. Ultimately, PHP will be running from your web server, either on the Internet or on an intranet. However, for development (and training) purposes, it makes sense to set up a local web server. This can be any machine on your network, but it's very easy to set up everything you need on your own desktop or laptop computer.

The good thing about web development is that it is culturally agnostic. It doesn't matter whether you develop on Macintosh, Windows, or even Linux: the product will be the same. What does matter is that you have all the right software in the right place.

For the impatient:

If you're anxious to get going, you can ignore all of this and skip over to the "Web Server Setup" section. It's very important and will make the rest of the book much easier to work with.

How the Web Works

When you open a page in your web browser, typically there's a lot of activity before you actually see anything. The process is something like this:

- You enter a URL in your browser's address bar.

  Modern browsers will hide part of the URL—notably the `https://` protocol. However, it's all there when you highlight it and copy it.

- The **domain name** needs to be **resolved** to an **IP address**.

  The domain name is something like `sample.example.com`. That could be anywhere, so the browser needs to search for it in the following locations.

- The **hosts** file is on the browser's computer and has a hard-coded list of domain names and IP addresses. The one you're looking for is almost certainly not there, but it doesn't hurt to look.

- If the hosts file doesn't have the domain name, the browser then looks in a distributed database called the **Domain Name System** (**DNS** to its friends).

  Presumably, it's found a matching IP address. If it hasn't, you'll get an error message.

- The browser then makes a connection to the server at the IP address.

  The server may well have many services to run, and each has been assigned a **port** number to identify which service you want.

For a live web server, the port number is almost always port 80, so that's the default. That may be included in the URL, but you probably won't see that. For a testing or development server, it's often a different number, which will need to be specified.

- The web server typically manages multiple sites at the same IP address in various directories. It then uses the domain name to choose which directory has what you want.

  The multiple domains on a single server are referred to as **virtual hosts**.

- If a page (or some other resource) has been specified, the server will try to satisfy your request. If it hasn't, the server will assume that you want a default page, typically called `index.php` or `index.html`.

  Of course, if it can't be found, you'll get a response telling you as much.

- If the request is for a PHP page, which is the whole point of this book, then the server will pass the file on to the PHP processor for further processing. You'll see more of this later.

  The result, with or without additional PHP processing, will be sent back to your computer, where it's up to the browser to work out what to do with it.

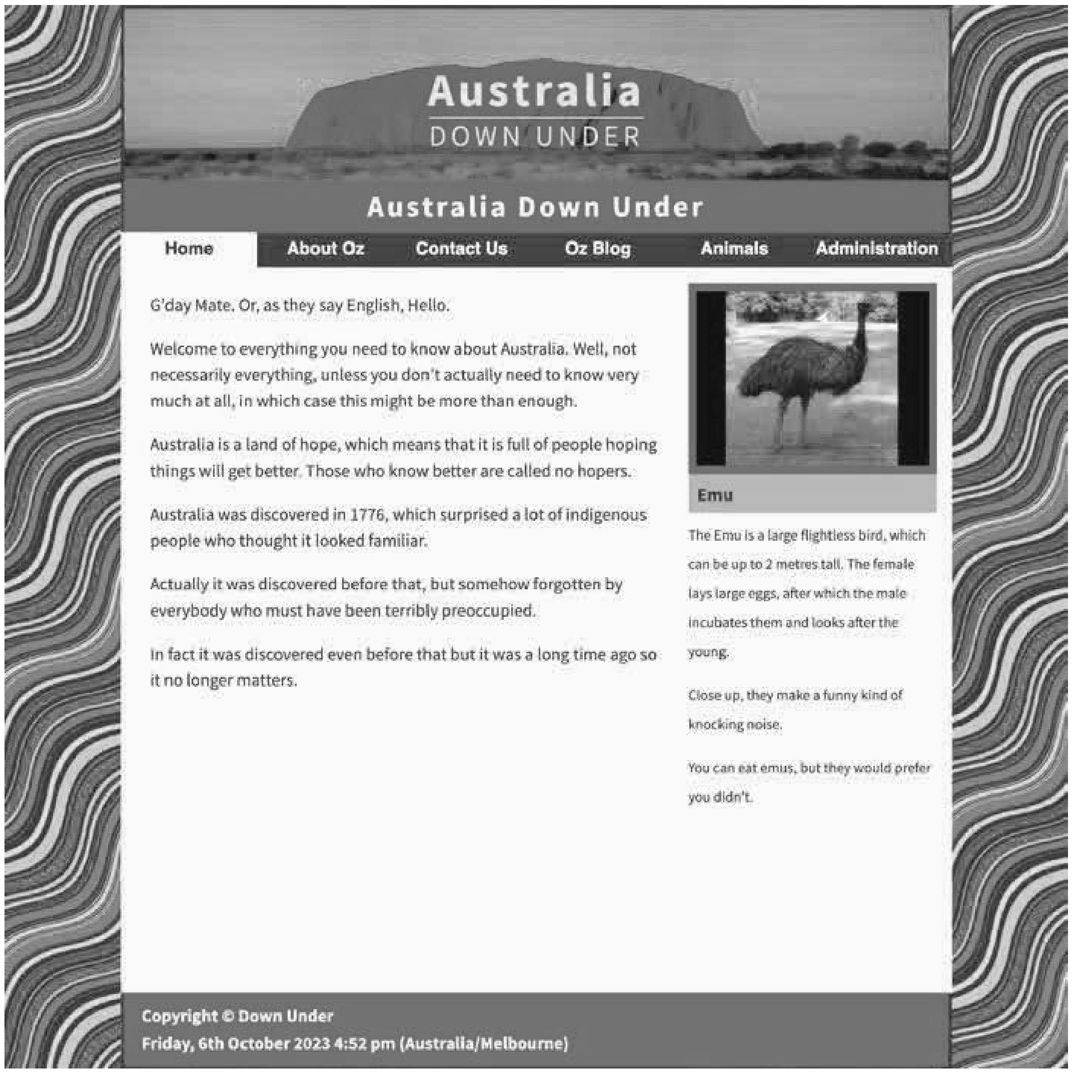
For learning and development purposes, we can create a simplified environment which works the same way, but stays on your own machine. You'll see this in the setup section.

The Sample Project

For learning purposes, we'll work on a sample site about Australia, because why not? You can see what the site will eventually look like here:

`https://down-under.net`

When you've finished the job, it should look like Figure 1-1.



Australia Down Under

G'day Mate. Or, as they say English, Hello.

Welcome to everything you need to know about Australia. Well, not necessarily everything, unless you don't actually need to know very much at all, in which case this might be more than enough.

Australia is a land of hope, which means that it is full of people hoping things will get better. Those who know better are called no hopers.

Australia was discovered in 1776, which surprised a lot of indigenous people who thought it looked familiar.

Actually it was discovered before that, but somehow forgotten by everybody who must have been terribly preoccupied.

In fact it was discovered even before that but it was a long time ago so it no longer matters.

The Emu is a large flightless bird, which can be up to 2 metres tall. The female lays large eggs, after which the male incubates them and looks after the young.

Close up, they make a funny kind of knocking noise.

You can eat emus, but they would prefer you didn't.

Copyright © Down Under
Friday, 6th October 2023 4:52 pm (Australia/Melbourne)

Figure 1-1. *The Sample Project Site: Australia Down Under*

The sample project is what you might call a “brochure site.” If you can imagine what a color glossy brochure would look like if it was implemented on the Web, it would be something like this.

Here are some of the things you would expect of a brochure site:

- An introduction to whatever it is the brochure is about
- A contact form to allow visitors to get in touch
- A collection of photos
- A collection of articles

Not much else. We’re not trying to sell anything—managing online commerce is a tricky and risky business and left to another book. We’re not starting discussions—that would also involve managing and registering users as well as moderating user content. Again, best left to another book.

You can quite easily adapt the site to any other context. It could be your amateur theater group, or hobby group, or advertising your flower shop or some sort of club. You’ll probably need to put some work into the CSS to give it a more suitable appearance, of course, but any decent designer can do that for you.

The point is that you’ll have the skills to generate and maintain the content of the site in any way you want.

The Tools

Presumably, you’ve already got a computer. It doesn’t matter what the operating system is, as long as you can install whatever software you need.

The main tools you'll need are

- A working web server, which is described in the next section.
- A web browser. Almost any current web browser will do, but it's probably a good idea in web development to test your results in more than one browser. For PHP development, we won't stretch the browser features too far.
- A good coding editor.

You'll also need a copy of the sample project, as we'll describe in the Web Server Setup section.

A Coding Editor

Most of the files for a website are text files. There are also images and the database, but all of your actual coding will involve text.

That means you can edit these files with any old text editor. More realistically, you'll want to use a text editor designed for writing code. Here are the features you'll want:

- There is a view of the project folder, making it easy to switch between files.
- The coding editor supports **syntax highlighting**, which recognizes the type of document you're editing (HTML, CSS, JavaScript, etc.) and highlights the parts of the language.

You may already have your preferred coding editor. If you don't, you can try one of these:

- **Pulsar**: `https://pulsar-edit.dev/`

  This is the successor to the Atom Text Editor.

- **VSCodium**: `https://vscodium.com/`

  This is the open source of Microsoft's Visual Studio Code (`https://code.visualstudio.com/`), without Microsoft's specific telemetry and proprietary features.

Both are free, cross-platform, and open source. They also have additional extension packages available to customize your editor.

If you have another coding editor already, that will do as well.

Web Server Setup

If you're going to work with PHP, you'll need to have a web server running PHP and a database. If you're doing this on Linux, you'll probably have that already, though there's still going to be some setting up to do.

If you're doing this on Macintosh or Windows, you'll need to download and install a web server for development. Fortunately, it's easy to install.

If you are doing this on Linux, you may prefer not to use the preinstalled web server and database for learning or development. One of the following recommended packages also has a version for Linux.

To begin with, you'll need to download a few things:

- Download and install a web server package. We recommend XAMPP or MAMP, and the following instructions presume that you're using one of these.

  You can download a package from the following locations:

  - **XAMPP**: `www.apachefriends.org/download.html`. Download and install the latest version for your operating system.
  - **MAMP**: `www.mamp.info/en/downloads/`. You won't need the MAMP Pro version for anything in this book.

- Download the sample site from `https://github.com/Apress/Introduction_to_PHP`. Unzip it and put it in a convenient location. For example (depending on your username, of course):

  ```
  /Users/fred/Documents/australia
  ```

  ```
  C:\Users\fred\Documents\australia
  ```

  Wherever you put the folder, your convenient location should not include spaces in the path name. The Apache Web Server really doesn't like spaces, so you may have issues if you use them.

- Download the **Virtual Hosts** application from `https://github.com/manngo/virtual-hosts/releases/latest`.

You'll use the Virtual Hosts application to set up the virtual site. It also includes a PHP runner to test sample PHP code.

You can set up everything without the Virtual Hosts application, of course, but it will involve hunting around for the files and editing them yourself.

One of the problems with editing these files is that you may need to have administrative permissions. The Virtual Hosts application will ask you.

For this book, we're going to set up a virtual host called `australia.example.com`. The `example.com` (as well as `example.net` and `example.org`) domains are reserved for testing and training and will never be used in live websites. That makes them ideal for this sort of thing.

Using the Virtual Hosts application (or not, if you prefer), you'll go through the following steps:

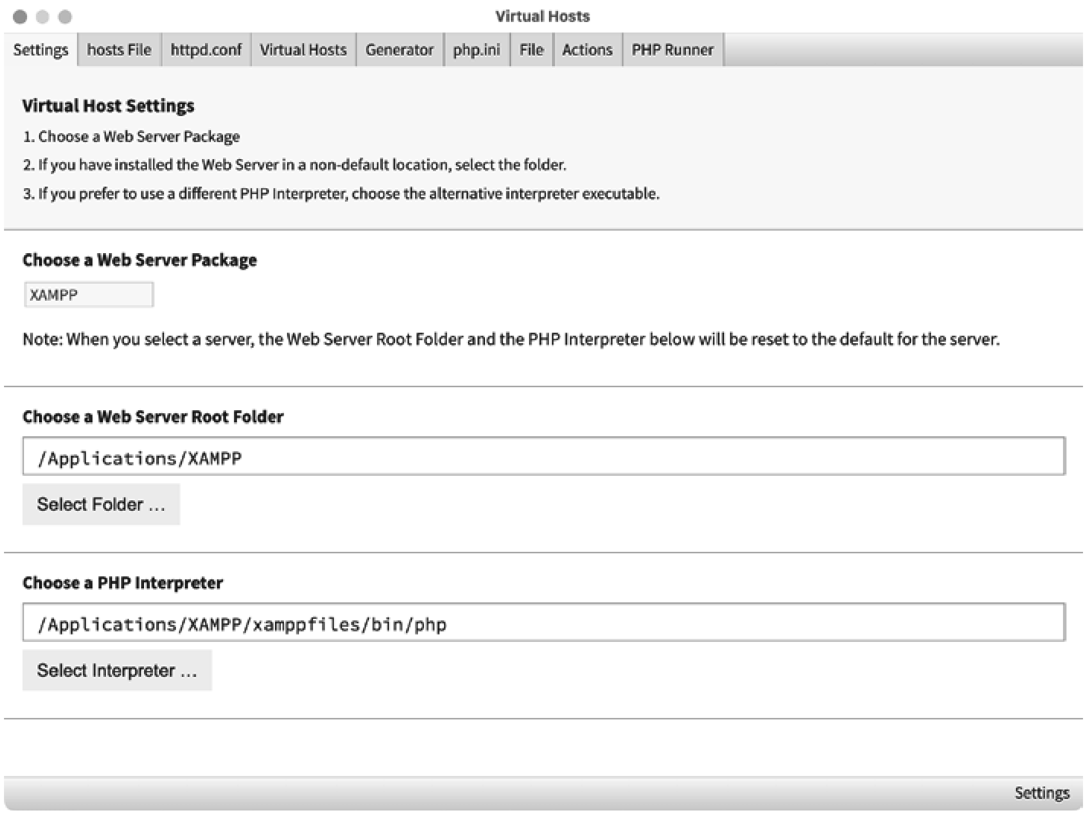
- Select your installed web server.
- Edit the `hosts` file to add the sample domain name.
- Edit the server's `httpd.conf` file to allow virtual hosts. You'll also need to change the user and group if you're on a Macintosh.
- Add the virtual host details for your sample.
- Edit the server's `php.ini` file to enable a fake mail server and to enable working with graphics.

Some of the files, particularly the `hosts` file, are in protected areas, so you may be asked for your admin password when saving your changes.

Using the Virtual Hosts Applications

The Virtual Hosts application will run without any special installation, so you can use it immediately.

The first step is to set up for your installed web server (Figure 1-2):



Virtual Host Settings

1. Choose a Web Server Package
2. If you have installed the Web Server in a non-default location, select the folder.
3. If you prefer to use a different PHP Interpreter, choose the alternative interpreter executable.

Choose a Web Server Package

XAMPP

Note: When you select a server, the Web Server Root Folder and the PHP Interpreter below will be reset to the default for the server.

Choose a Web Server Root Folder

/Applications/XAMPP

Choose a PHP Interpreter

/Applications/XAMPP/xamppfiles/bin/php

Figure 1-2. *Virtual Host Settings*

- Choose your web server software.

  It will prefill the Root Folder and PHP Interpreter values.

- If you have installed your web server other than in the default location, choose the location for your **Web Server Root Folder**.

- If you have a preferred PHP interpreter you want to use for testing code samples, choose the executable file for your **PHP Interpreter**.

  Changing this PHP interpreter is purely for the PHP Runner tab. For the website, the web server will still use the standard built-in PHP interpreter.

Save these settings for next time. They will be saved in a file called `.virtual-hosts.json` in your home directory.

Adding the Domain Name

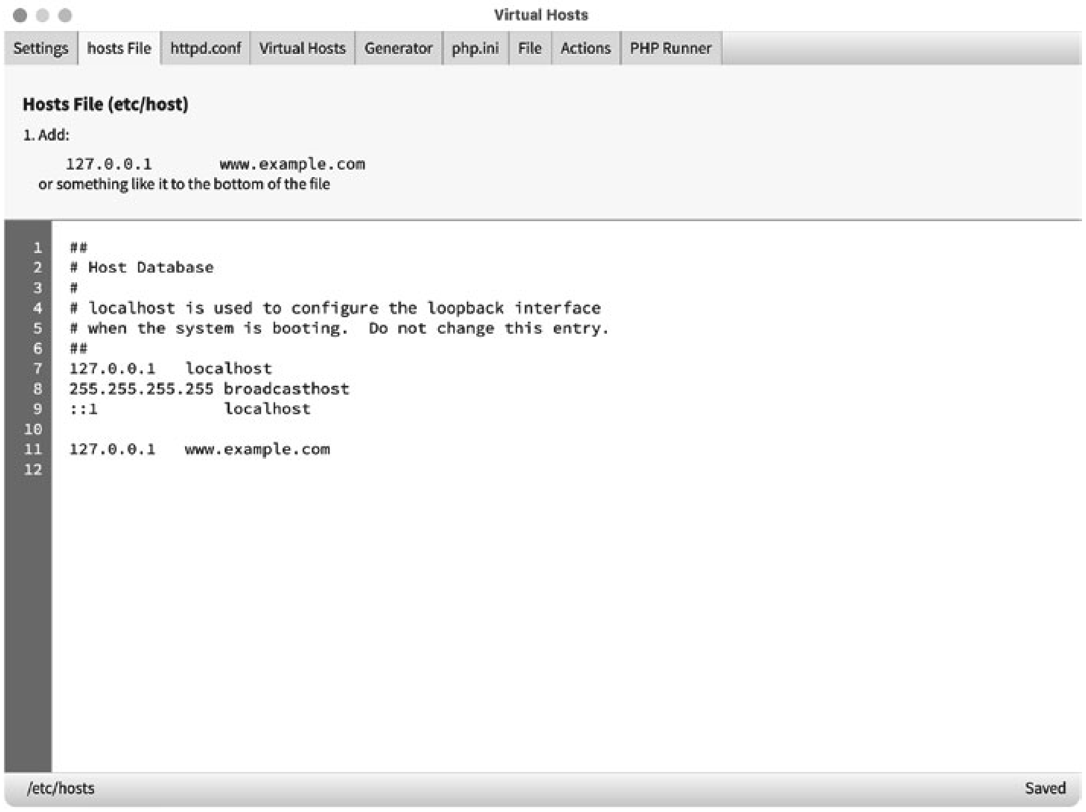
The `hosts` file is where the browser will look first when resolving a domain name. The location of the `hosts` file is

- Macintosh and Linux: `/etc/hosts`
- Windows: `C:\Windows\System32\drivers\etc\hosts`

This is independent of the installed web server software. It's used by all interested applications.

The `hosts` file is in a protected area, so you may have trouble editing it yourself.

Using Virtual Hosts, select the **hosts File** tab (Figure 1-3).



Hosts File (etc/host)

1. Add:

```
127.0.0.1       www.example.com
```

or something like it to the bottom of the file

```
##
# Host Database
#
# localhost is used to configure the loopback interface
# when the system is booting.  Do not change this entry.
##
127.0.0.1   localhost
255.255.255.255 broadcasthost
::1             localhost

127.0.0.1   www.example.com

```

/etc/hosts Saved

Figure 1-3. *Hosts File*

Add the following to the end:

```
127.0.0.1   australia.example.com
```

The IP address `127.0.0.1` is the virtual IP address for your own computer. It means that `australia.example.com` will resolve to your own computer which is where your development server (probably) is.

For future projects, you can have as many additional domain names as you like, and they can all resolve to `127.0.0.1`. It's up to the web server to associate each name to a different web directory.

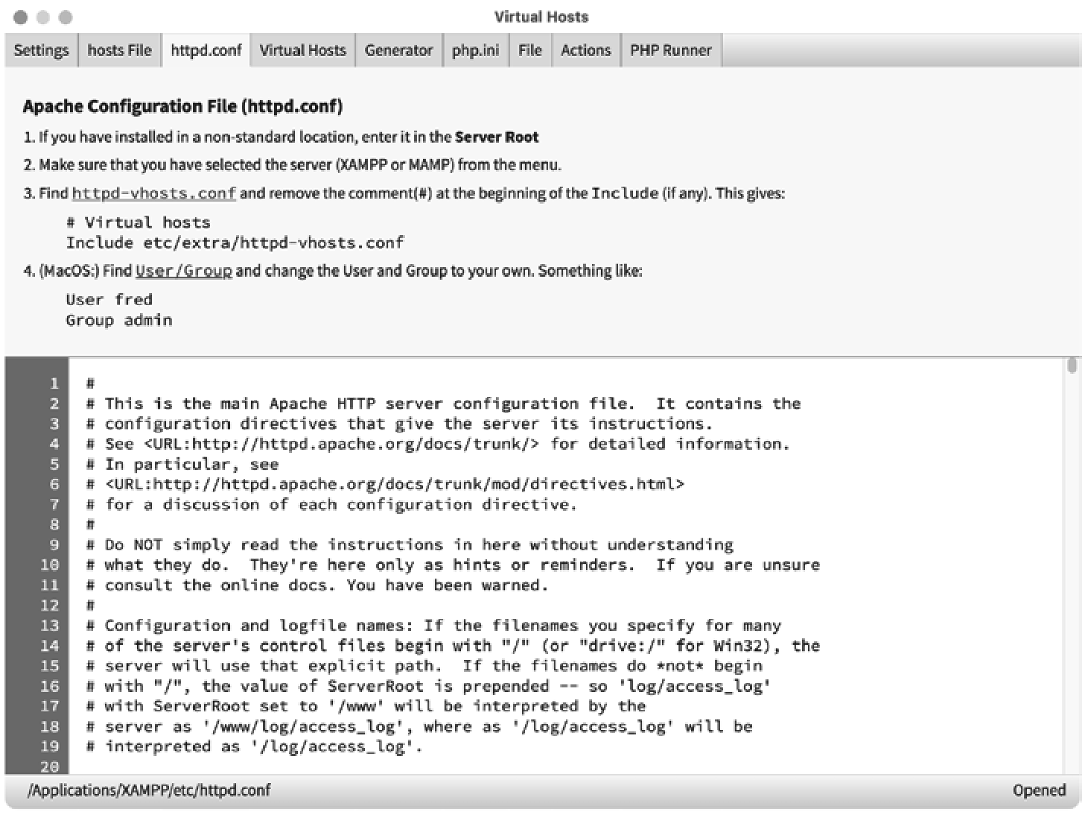
Don't forget to save the file and, if requested, enter your password.

Editing the httpd.conf File

The Apache Web Server uses the `httpd.conf` for its main settings. It's normally located in

- **XAMPP (macOS)**: `/Applications/XAMPP/etc/httpd.conf`
- **XAMPP (Windows)**: `C:\xampp\apache\conf\httpd.conf`
- **MAMP (macOS)**: `/Applications/MAMP/conf/apache/httpd.conf`
- **MAMP (Windows)**: `C:\MAMP\conf\apache\httpd.conf`

Using Virtual Hosts, select the **httpd.conf** tab (Figure 1-4).



Apache Configuration File (httpd.conf)

1. If you have installed in a non-standard location, enter it in the **Server Root**
2. Make sure that you have selected the server (XAMPP or MAMP) from the menu.
3. Find httpd-vhosts.conf and remove the comment(#) at the beginning of the Include (if any). This gives:

```
# Virtual hosts
Include etc/extra/httpd-vhosts.conf
```

4. (MacOS:) Find User/Group and change the User and Group to your own. Something like:

```
User fred
Group admin
```

```
#
# This is the main Apache HTTP server configuration file.  It contains the
# configuration directives that give the server its instructions.
# See <URL:http://httpd.apache.org/docs/trunk/> for detailed information.
# In particular, see
# <URL:http://httpd.apache.org/docs/trunk/mod/directives.html>
# for a discussion of each configuration directive.
#
# Do NOT simply read the instructions in here without understanding
# what they do.  They're here only as hints or reminders.  If you are unsure
# consult the online docs. You have been warned.
#
# Configuration and logfile names: If the filenames you specify for many
# of the server's control files begin with "/" (or "drive:/" for Win32), the
# server will use that explicit path.  If the filenames do *not* begin
# with "/", the value of ServerRoot is prepended -- so 'log/access_log'
# with ServerRoot set to '/www' will be interpreted by the
# server as '/www/log/access_log', where as '/log/access_log' will be
# interpreted as '/log/access_log'.
```

/Applications/XAMPP/etc/httpd.conf Opened

Figure 1-4. *`httpd.conf` File*

The instruction section at the top of the page includes shortcut links to some sections you'll need to find.

- Find the comment line which says `# Virtual Hosts`. There's a shortcut by clicking the `httpd-vhosts.conf` link in the instruction section.

  Remove the comment (`#`), if any, at the beginning of the `Include ...` line below it. Don't change the actual `Include` statement. Depending on your operating system, you should have something like

  ```
  # Virtual hosts
  Include etc/extra/httpd-vhosts.conf
  ```

  or

  ```
  # Virtual hosts
  Include conf/extra/httpd-vhosts.conf
  ```

- For Macintosh users, find the `User/Group` section, and change the settings to something like the following, using your own username, of course:

  ```
  User fred

  Group admin
  ```

- Don't forget to save the file and, if requested, enter your password.

Adding a Virtual Host

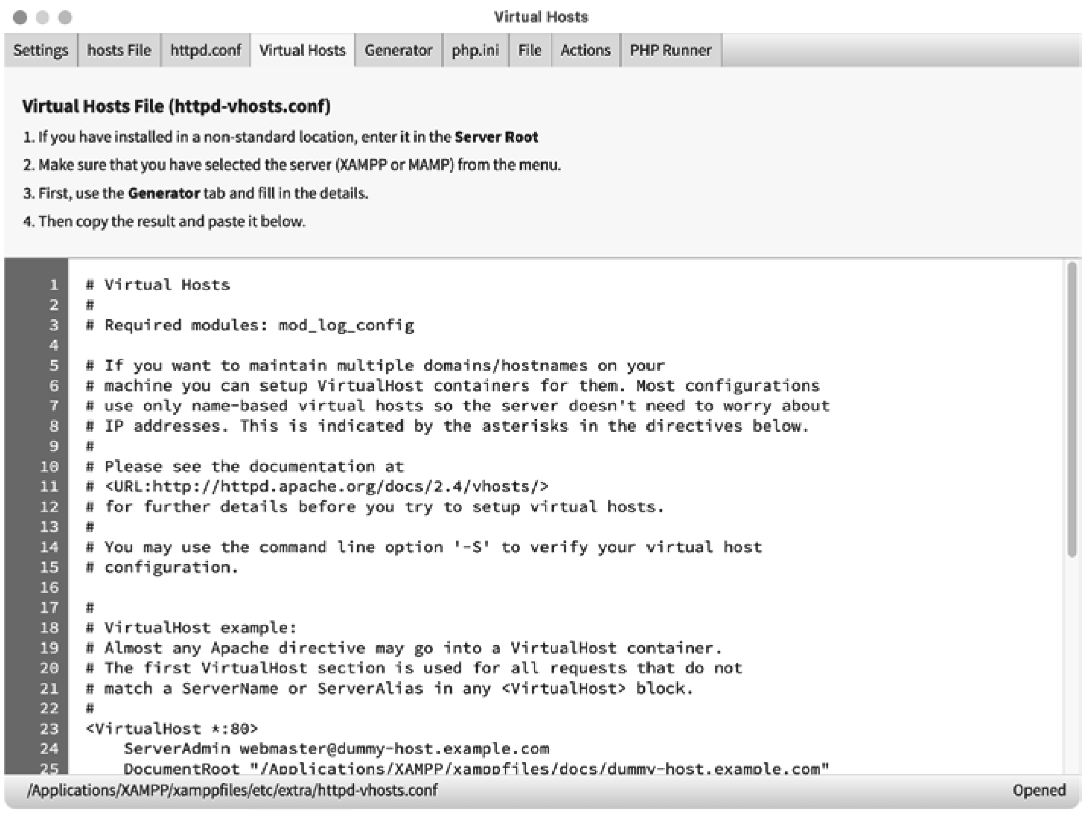
The actual Virtual Hosts file is where you give the details of your virtual host and where the directory is located. It can be found at the following locations:

- **XAMPP (macOS)**: `/Applications/XAMPP/xamppfiles/etc/extra/httpd-vhosts.conf`
- **XAMPP (Windows)**: `C:\xampp\apache\conf\extra\httpd-vhosts.conf`
- **MAMP (macOS)**: `/Applications/MAMP/conf/apache/extra/httpd-vhosts.conf`
- **MAMP (Windows)**: `C:\MAMP\bin\apache\conf\extra\httpd-vhosts.conf`

The web server won't bother looking for virtual hosts unless it has been set in the `httpd.conf` file, which is what you did in the previous step.

Setting up the virtual host will involve changing some data in the **Virtual Hosts** tab, using data from the **Generator** tab.

Using the Virtual Hosts application, select the **Virtual Hosts** tab (Figure 1-5).



Virtual Hosts File (httpd-vhosts.conf)

1. If you have installed in a non-standard location, enter it in the **Server Root**
2. Make sure that you have selected the server (XAMPP or MAMP) from the menu.
3. First, use the **Generator** tab and fill in the details.
4. Then copy the result and paste it below.

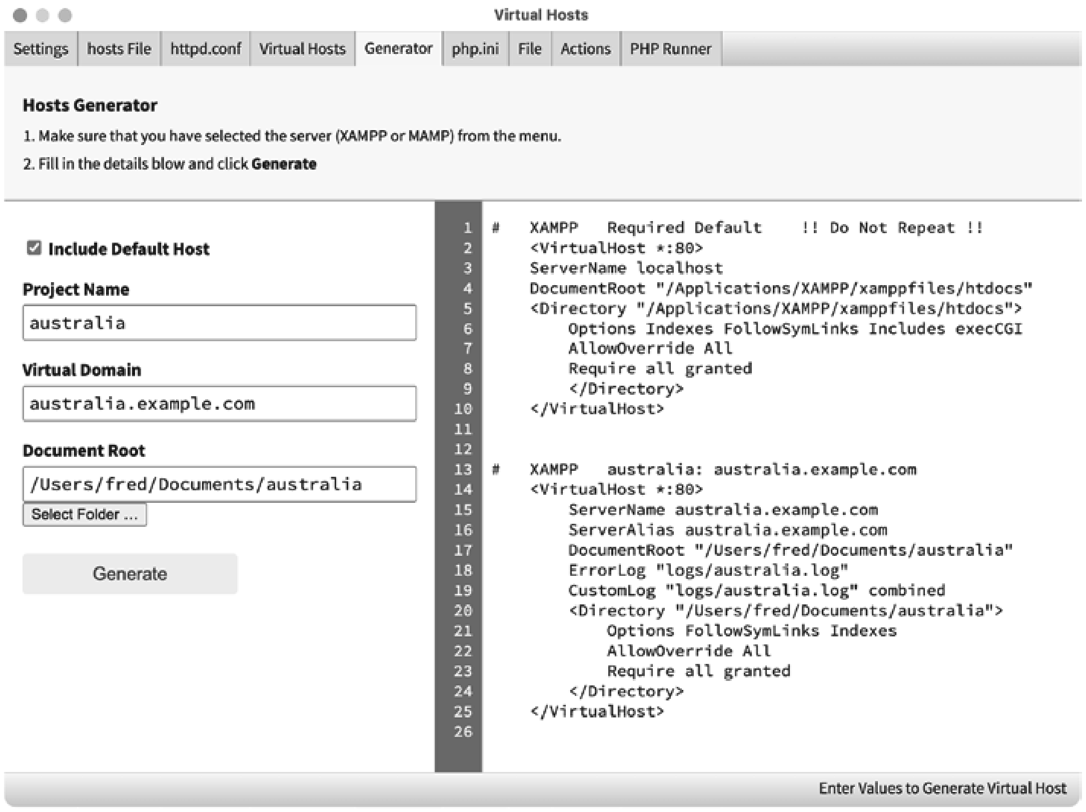
```
# Virtual Hosts
#
# Required modules: mod_log_config

# If you want to maintain multiple domains/hostnames on your
# machine you can setup VirtualHost containers for them. Most configurations
# use only name-based virtual hosts so the server doesn't need to worry about
# IP addresses. This is indicated by the asterisks in the directives below.
#
# Please see the documentation at
# <URL:http://httpd.apache.org/docs/2.4/vhosts/>
# for further details before you try to setup virtual hosts.
#
# You may use the command line option '-S' to verify your virtual host
# configuration.

#
# VirtualHost example:
# Almost any Apache directive may go into a VirtualHost container.
# The first VirtualHost section is used for all requests that do not
# match a ServerName or ServerAlias in any <VirtualHost> block.
#
<VirtualHost *:80>
    ServerAdmin webmaster@dummy-host.example.com
    DocumentRoot "/Applications/XAMPP/xamppfiles/docs/dummy-host.example.com"
```

/Applications/XAMPP/xamppfiles/etc/extra/httpd-vhosts.conf

Figure 1-5. *Virtual Hosts*

There may be some commented instructions as well as some default settings. You can safely delete all of this and replace it with the data from the Generator tab in the Virtual Hosts application.

After clearing out the old contents, select the **Generator** tab (Figure 1-6).



Hosts Generator

1. Make sure that you have selected the server (XAMPP or MAMP) from the menu.
2. Fill in the details blow and click **Generate**

☑ Include Default Host

Project Name: australia

Virtual Domain: australia.example.com

Document Root: /Users/fred/Documents/australia

```
#    XAMPP   Required Default    !! Do Not Repeat !!
     <VirtualHost *:80>
     ServerName localhost
     DocumentRoot "/Applications/XAMPP/xamppfiles/htdocs"
     <Directory "/Applications/XAMPP/xamppfiles/htdocs">
         Options Indexes FollowSymLinks Includes execCGI
         AllowOverride All
         Require all granted
         </Directory>
     </VirtualHost>


#    XAMPP   australia: australia.example.com
     <VirtualHost *:80>
         ServerName australia.example.com
         ServerAlias australia.example.com
         DocumentRoot "/Users/fred/Documents/australia"
         ErrorLog "logs/australia.log"
         CustomLog "logs/australia.log" combined
         <Directory "/Users/fred/Documents/australia">
             Options FollowSymLinks Indexes
             AllowOverride All
             Require all granted
         </Directory>
     </VirtualHost>

```

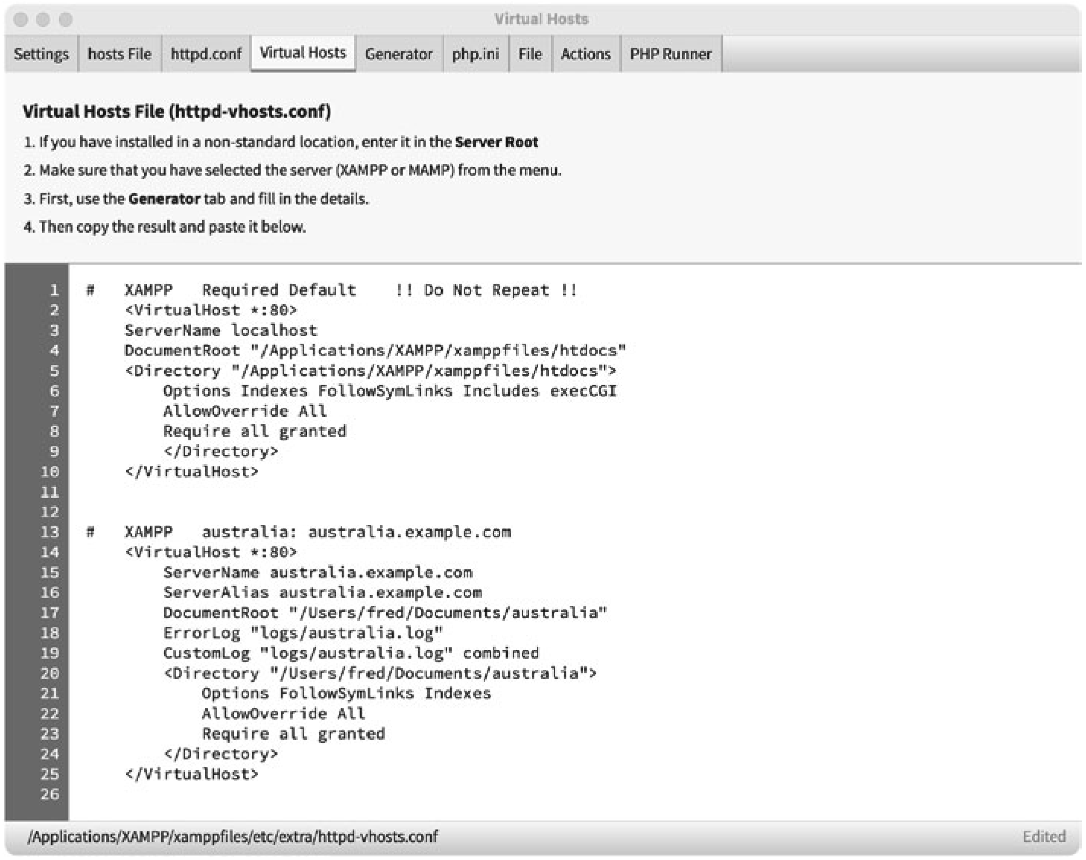
Enter Values to Generate Virtual Host

Figure 1-6. *Generator Tab*

- If this is your first (or only) virtual host, be sure to select `Include Default Host` if available. You'll probably need this for XAMPP but not for MAMP.
- Select a Project Name. You can call it anything you like, but it's best to use a simple lower case name without spaces, such as `australia`.
- Enter the Virtual Domain that you added to the `hosts` file earlier.
- Select the folder of your sample project.

You can then click the **Generate** button (or just click in the text area on the right).

Copy the generated text and paste it into the `Virtual Hosts` tab (Figure 1-7).



Virtual Hosts File (httpd-vhosts.conf)

1. If you have installed in a non-standard location, enter it in the **Server Root**
2. Make sure that you have selected the server (XAMPP or MAMP) from the menu.
3. First, use the **Generator** tab and fill in the details.
4. Then copy the result and paste it below.

```
#   XAMPP   Required Default    !! Do Not Repeat !!
    <VirtualHost *:80>
    ServerName localhost
    DocumentRoot "/Applications/XAMPP/xamppfiles/htdocs"
    <Directory "/Applications/XAMPP/xamppfiles/htdocs">
        Options Indexes FollowSymLinks Includes execCGI
        AllowOverride All
        Require all granted
        </Directory>
    </VirtualHost>


#   XAMPP   australia: australia.example.com
    <VirtualHost *:80>
        ServerName australia.example.com
        ServerAlias australia.example.com
        DocumentRoot "/Users/fred/Documents/australia"
        ErrorLog "logs/australia.log"
        CustomLog "logs/australia.log" combined
        <Directory "/Users/fred/Documents/australia">
            Options FollowSymLinks Indexes
            AllowOverride All
            Require all granted
        </Directory>
    </VirtualHost>
```

/Applications/XAMPP/xamppfiles/etc/extra/httpd-vhosts.conf Edited

Figure 1-7. *Virtual Hosts Copied*

For additional projects, you can repeat the process (with different names and directories, of course), but you shouldn't repeat the `Do Not Repeat` section.

Don't forget to save the file and, if requested, enter your password.

Fixing the php.ini File

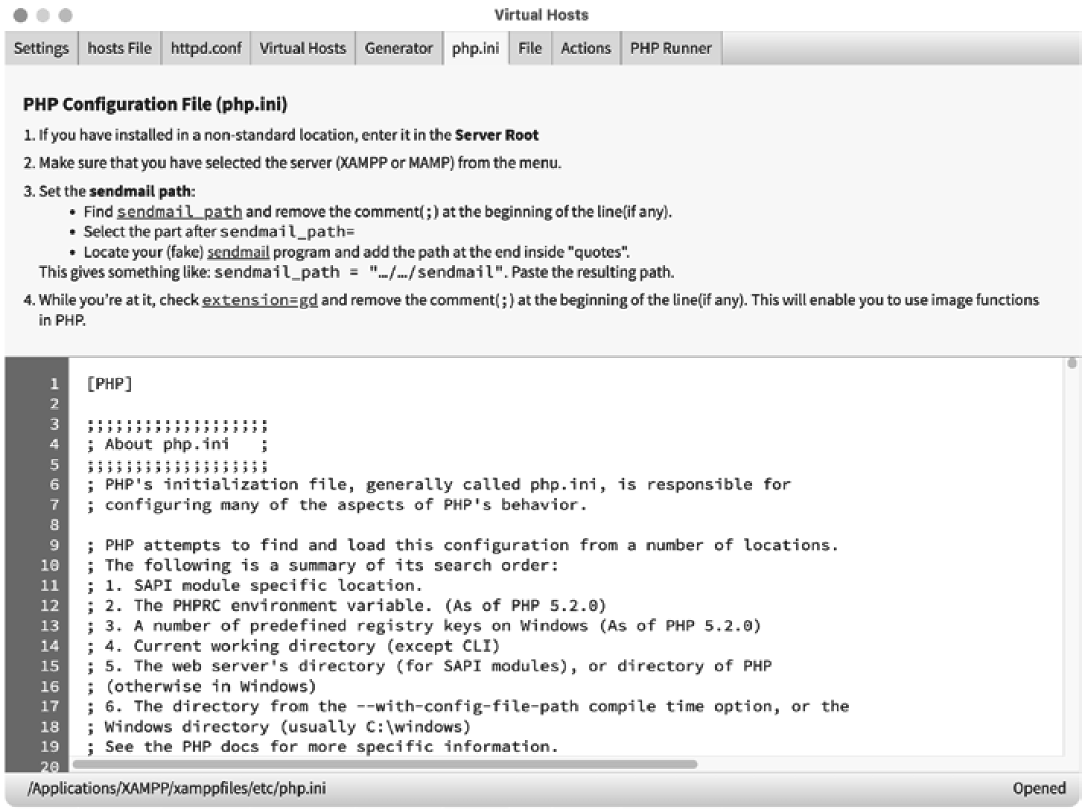
The `php.ini` file contains default settings for how PHP works. Many of these settings can be changed later, and we'll look at doing that later. However, there are two settings which need to be configured before PHP starts up.

The `php.ini` file can be found at

- **XAMPP (macOS)**: `/Applications/XAMPP/xamppfiles/etc/php.ini`
- **XAMPP (Windows)**: `C:\xampp\php\php.ini`
- **MAMP (macOS)**: `/Applications/MAMP/bin/php/php{version}/conf/php.ini`
- **MAMP (Windows)**: `C:\MAMP\conf\php{version}\php.ini`

Note that in MAMP, there are several versions of PHP and so several locations for the `php.ini` file. The default will be the latest version.

Using the Virtual Hosts application, select the **php.ini** tab (Figure 1-8).



PHP Configuration File (php.ini)

1. If you have installed in a non-standard location, enter it in the **Server Root**
2. Make sure that you have selected the server (XAMPP or MAMP) from the menu.
3. Set the **sendmail path**:
   - Find `sendmail_path` and remove the comment(`;`) at the beginning of the line(if any).
   - Select the part after `sendmail_path=`
   - Locate your (fake) sendmail program and add the path at the end inside "quotes".

   This gives something like: `sendmail_path = ".../.../sendmail"`. Paste the resulting path.
4. While you're at it, check `extension=gd` and remove the comment(`;`) at the beginning of the line(if any). This will enable you to use image functions in PHP.

```
[PHP]

;;;;;;;;;;;;;;;;;;;
; About php.ini   ;
;;;;;;;;;;;;;;;;;;;
; PHP's initialization file, generally called php.ini, is responsible for
; configuring many of the aspects of PHP's behavior.

; PHP attempts to find and load this configuration from a number of locations.
; The following is a summary of its search order:
; 1. SAPI module specific location.
; 2. The PHPRC environment variable. (As of PHP 5.2.0)
; 3. A number of predefined registry keys on Windows (As of PHP 5.2.0)
; 4. Current working directory (except CLI)
; 5. The web server's directory (for SAPI modules), or directory of PHP
; (otherwise in Windows)
; 6. The directory from the --with-config-file-path compile time option, or the
; Windows directory (usually C:\windows)
; See the PHP docs for more specific information.
```

/Applications/XAMPP/xamppfiles/etc/php.ini Opened

Figure 1-8. *`php.ini` File*

The instruction section at the top of the page includes shortcut links to some sections you'll need to find.

One of the things we'll be doing involves email. PHP doesn't actually send the email, but passes it on to a separate email server. In a live server, there would be a proper email server to do the job, but in this case, we'll work with a fake mail server. The fake mail server will accept whatever is passed to it as an email, but will save it into a file instead of sending it on.

- Find the `sendmail_path`, and remove the comment semicolon (`;`) before it.
- Click the **locate (fake) sendmail** link and locate the fake sendmail file. You'll get a file dialog where you can choose a fake mail program.

  There's one in the `resources` directory in your sample project. For macOS, you should use `fakemail.sh`, while for Windows, you should use `fakemail.bat`.

  Once you have selected the file, paste it at the end. The result should be something like this:

  ```
  sendmail_path = ...
  ```

The other thing you'll need to fix is a reference to the **gd** library. This library is used by PHP to manipulate images.

- Find the reference to `extension=gd` (it might be something like `extension=php_gd2.dll`).
- Remove the comment semicolon (`;`) before it. It should look something like this:

  ```
  extension=php_gd2.dll
  ;   or, maybe this:
  extension=gd
  ```

  The actual library name will depend on your server version.

If you're on a Macintosh, you may find the only reference to GD is a `.dll` file. If so, don't bother enabling it—DLLs are for Windows only. GD is probably enabled anyway.

Configuring PHP

PHP allows you to configure its behavior in many useful respects. Here, we'll discuss just a few which may be of most use to you as a web developer.

At its most fundamental level, you can reconfigure PHP by compiling it with different options or by including different libraries to supplement its behavior. However, the following three techniques are used for a working implementation of PHP:

- `php.ini`

  PHP uses settings from the php.ini file when it starts up. These settings affect all PHP scripts on the server, so changes to this file are global.

  Generally, php.ini is inaccessible if you are not using your own personal server, such as a hosted or corporate server. If you do make changes to this file, then you will need to restart your web server to reload these settings.

- `.htaccess`

  If you are running PHP as a module under the Apache Web Server, such as with XAMPP or WAMP, then you may be able to use the `.htaccess` file to adjust your configuration. (The dot at the beginning of its name tends to hide it from view on Unix-type servers such as Linux and is common for configuration files.)

The `.htaccess` file should be placed in a directory where you want the changes to be made. It will be applicable to the current directory as well as all of the subdirectories. You can have different `.htaccess` files in different directories: the current settings will always override the settings of parent directories.

Apache will apply the settings when it loads a file. This means that changes are instantaneous, and you will not have to restart your server. It also means there is a slight overhead of additional file handling when you load pages.

The `.htaccess` file may also contain some settings for the web server other than for PHP. In fact, that's its main purpose—to customize the web server.

There is a `.htaccess` file in the root directory of the sample project.

- `.user.ini`

  This is an alternative settings file for PHP, but it only applies when PHP is not running as an Apache Module. Even if this is the case, there may be some additional settings in the `.htaccess` file for the web server alone.

  There is a .user.ini file in the root directory of the sample project.

- PHP `ini_set()` functions

  PHP will also allow you to reconfigure some settings using PHP functions. This is most suitable if you want to make changes for a single script at a time. It is also an option if your server cannot support `.htaccess` files or if it deliberately ignores them.

  Not all PHP settings can be changed this way. In particular, ‘some changes will be made too late, since they may be required if there are problems with processing the script altogether.

The supplied `.htaccess` and `.user.ini` files have the default or most useful settings for the current project. In the next section, we’ll discuss these particular settings; in later chapters, we’ll also revisit some of these settings in context.

Setting PHP Options

Generally, each setting has a name and a value. In the `php.ini` or `.user.ini` file, the format is

```
setting_name = value
```

For `.htaccess` files, which may include other non-php settings, the format is

```
php_value setting-name value
```

For a PHP function, you use the function `ini_set(name, value)`. In this function, both parameters are supposed to be strings, though the second parameter (value) may be a number:

```
ini_set(name, value);
```

Some Useful Configuration Options

The following are some PHP configuration options which you may find useful for individual scripts or projects.

PHP Execution

The following settings may be useful for some scripts with unusually large processing requirements:

```
max_execution_time = 30
memory_limit = 16M
```

This is the default timeout and memory limit for a single task. If you have a particularly long or memory-intensive task, you may need to increase these. Note that you can't afford to have too much of this going on on a single server.

File Uploads

Two settings are useful when uploading files, such as image files:

```
upload_max_filesize = 2M
post_max_size = 8M
```

Again, these are the defaults. It means that we'll accept a maximum of 2Mb for a single file and 8Mb for the total upload for a single submit. If you're uploading video files or some image files, you may need to change these.

Legacy Settings

PHP originally used some settings to make scripting easier. Unfortunately, they also backfired and so should be disabled. In modern versions, they are removed altogether, or at least disabled by default, in which case these settings will be ignored. If, however, you're stuck on a very old web server, these settings may still need to be turned off:

```
register_globals = Off
magic_quotes_gpc = Off
```

The `register_globals` setting was to do with how PHP automatically accepted incoming form data, which can be risky. The `magic_quotes_gpc` was associated with modifying incoming data for the database, which was messy and unreliable.

Modern techniques are safer and more reliable.

Sessions

PHP sessions allow you to keep track of user-related data between pages. We'll be working with sessions later.

```
session.gc_maxlifetime = 1400

session.gc_probability = 1
session.gc_divisor = 100
```

The `session.gc_maxlifetime` setting is an idle time, in seconds, for session data, after which the data can be expired.

The `session.gc_probability` and `session.gc_divisor` settings combine to determine how often expired data is actually deleted.

These settings may well be adjusted for your particular application. They may also be adjusted between the development and final deployment versions of your site.

Error Reporting

PHP will normally report on errors, either on the screen or in a log file. Error reporting can be set to different levels, and you may choose whether PHP will display only serious errors or less serious errors from which PHP can continue.

```
error_reporting = E_ALL
display_errors = On
log_errors = On
```

The `error_reporting` setting determines how much you want to report; this includes minor warnings to fatal errors. For development, you'll want to set it to maximum.

The `display_errors` setting is whether you want errors displayed on the screen—for development, yes; for deployment, probably not. The `log_errors` setting is whether you want the errors logged to a file.

Other Settings

The following settings are purely a matter of choice, but suggested anyway:

```
php_value date.timezone Australia/Melbourne
```

If you're working with dates in PHP, and you will in this project, you will need to have your time zone set. It may already have been set in your `php.ini` file, but it may not be to your liking. We'll look at using this setting in Chapter 2.

```
php_value mail.log /var/log/phpmail.log
php_flag mail.add_x_header Off
```

We'll be sending email in Chapter 3. These settings determine where any log files are stored and whether PHP should add its own custom header to the email:

```
php_flag output_buffering Off
```

When working with cookies and session data, we'll encounter issues with outputting data before the cookies have been set, which is fatal. Output buffering helps in solving these issues, so it's actually convenient to have this setting turned to **On**. However, it's turned off here, so that we can experience the issues and learn how to fix the code.

Starting Up

When everything is set, you can start up your web server and database. How you do this depends on your software and operating system. Open the application control panel:

- For **XAMPP** on macOS, select the **Manage Servers** tab and start both the **MySQL Database** and the **Apache Web Server**.
- For **XAMPP** on Windows, select the **Start** button for both **Apache** and **MySQL**.
- For **MAMP** on macOS, select the **Start** button, which will start both the web server and the database server.
- For **MAMP** on Windows, select the **Start Servers** button, which will start both the web server and the database server.

You should now be ready to view your project site.

The Project Configuration Files

The sample project includes both `.htaccess` and `.user.ini` files, and you can see the preceding settings as described. The syntax for the `.user.ini` file is different from the syntax in the `.htaccess` file, but the intention is clear enough.

If you're running XAMPP or MAMP, or indeed many other web server packages, you'll be running PHP as a **module**, which means that PHP will be an extension of the Apache (or other) web server. In that case, the `.htaccess` file will be used, and the `.user.ini` file will be totally ignored.

It's not so simple if you're running PHP as a separate process.

If you're not sure, both XAMPP and MAMP have a link to `phpinfo`, which will dump a lot of information about PHP and its setup.

Somewhere near the top (thankfully) is a setting for the **Server API**:

- If its value is **Apache 2.0 Handler**, then Apache is running as a module.
- If the value is something like **CGI**/**FastCGI** or **FPM/ FastCGI**, then it's running as a separate process.

The reason why this is important is that if PHP is running as a separate process, and not as a module, then any PHP settings in the `.htaccess` file will cause errors, as they won't be recognized. For that reason, you'll see the PHP settings inside a conditional block:

```
<IfModule php_module>
    ...
</IfModule>
```

This is supposed to include the PHP settings only if the PHP module is running.

Unfortunately, the PHP module isn't always called `php_module`, so the test may not work.

If you're sure that PHP is running as a module, and you're sure that the preceding block isn't working, you can try commenting out the conditional lines:

```
#<IfModule php_module>
    ...
#</IfModule>
```

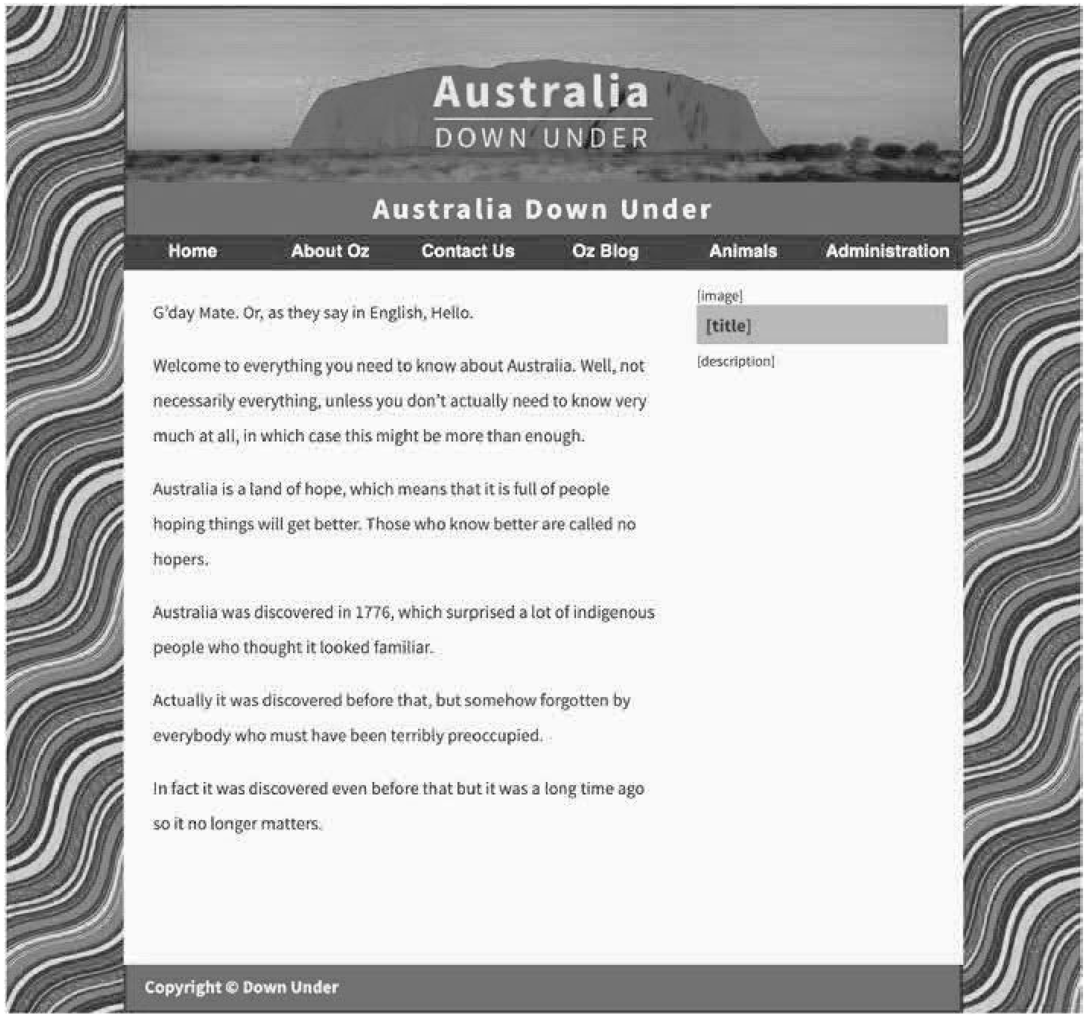
The # at the beginning of the line treats the rest of the line as a comment, and the whole line will be ignored. The settings between will then be loaded as expected.

Finishing Up

If all is working, you can enter the following URL:

```
australia.example.com
```

When you do, you should see the page shown in Figure 1-9.



G'day Mate. Or, as they say in English, Hello.

Welcome to everything you need to know about Australia. Well, not necessarily everything, unless you don't actually need to know very much at all, in which case this might be more than enough.

Australia is a land of hope, which means that it is full of people hoping things will get better. Those who know better are called no hopers.

Australia was discovered in 1776, which surprised a lot of indigenous people who thought it looked familiar.

Actually it was discovered before that, but somehow forgotten by everybody who must have been terribly preoccupied.

In fact it was discovered even before that but it was a long time ago so it no longer matters.

Figure 1-9. *Australia Down Under Home Page (Incomplete)*

It's not as complete as the live sample, and if you click any of the menu items, it will all fall apart.

That's the whole point of this book. We are going to look at how to write the code that will complete the rest of the site.

To begin with, we'll look at how we actually write PHP in Chapter 2.

CHAPTER 2

Working with PHP

PHP has a rather confusing beginning—it was not originally intended to be a programming language. Instead, it was a collection of tools to be mixed in with HTML.

Things have moved on since then, and there is quite a lot of programming in PHP, and we'll certainly be doing quite a lot of it ourselves. However, PHP on the Web is still designed to be mixed in with HTML, and you'll especially see that when you include the results of your PHP coding in your output page.

As a result of this hybrid nature, you'll see two particular features in PHP coding:

- All PHP code will be in files with the `.php` extension. This causes the web server to send the code to a separate PHP processor.

  As you'll see later, PHP files can include other files. Those included files don't technically need to have the `.php` extension, because they'll be regarded as part of the original PHP file.

- All PHP code needs to be enclosed in special PHP blocks enclosed in PHP tags. Anything not included in the PHP tags will be simply passed through.

> For a normal web page, any text not inside a PHP block is presumably HTML. It doesn't matter—PHP will just leave it alone and let the browser deal with it. On the other hand, code in PHP blocks *may* also add to the output to be included with the rest of the text.

In this chapter, we'll look at setting up our pages to work with PHP. This includes renaming pages to have the correct extension, adding PHP blocks to do any processing that needs to be done, and including special blocks to output any processed results.

We'll also look at delegating some of the code to separate files to make your code management easier.

In the process, we'll start to learn about how to write PHP statements and storing data in simple and complex variables. We'll also learn a little about some of PHP's built-in functions.

Finally, we'll look at *not* writing code, that is, how to put additional comments into our PHP code.

Before We Start

Programming is a matter of stating things precisely. For this reason, we will also need to be clear about our terminology.

Grouping: Brackets and Friends

There are three sets of symbols used to group items:

Symbol	Name	Use
( )	Parentheses	Functions and function dataCalculations
{ }	Braces	Blocks of code
[ ]	[Square] Brackets	Array values and keys

Note that the proper name for the `[ ]` symbol is simply **brackets**. However, to avoid confusion, we will stick to calling them **square brackets**. However, we will also stick to the proper names for the other symbols and never call them brackets.

Sometimes, you'll see braces referred to as "curly braces," but not in this book.

Data Types

PHP, like most programming languages, works with different types of values. We'll look at more complex types later, but there are three simple types:

- **Numbers** are stored in a binary format and are used whenever you need to count something. They are normally written plain, though there are some special formats.
- **Strings** are strings of characters. This may be interpreted as text, but can contain non-text characters.
- **Boolean** values are special values for `true` and `false`, so-called after the mathematician George Boole. They are very often used indirectly, such as when comparing values, such as `a < b`.

There are other special values, such as `null`, which we'll see more later. When you write the value in code, it is called a **literal**. For example:

- `23` is a numeric literal.
- `"hello"` and `'goodbye'` are both string literals. String literals are normally written in single or double quotes, depending on how you're going to use them.
- The values `true` and `false` are the only boolean literals.

You can store values in **variables**, which are named placeholders for values.

How PHP Processes Files

The web server is responsible for fetching and sending all requested files back to the user's browser. With PHP installed, the web server will send the file first to the PHP processor, but, to save wasted effort, this happens only if the file has `.php` extension.

Often, there will be another step. Many sites, including this sample project, maintain much of the data in a database. Here, PHP will be used to send requests to the database server and to process the results.

Figure 2-1 illustrates the process.

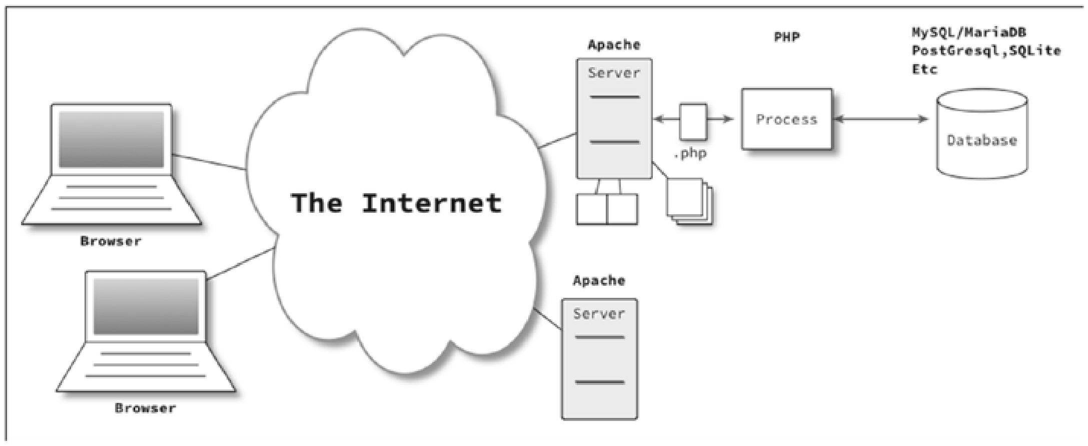


Figure 2-1. *Serving Web Pages with PHP*

In a PHP page, the PHP process will only process code which is inside PHP blocks (`<?php ... ?>`). The content of these blocks will be replaced by their output, if any, or removed. All text outside the PHP blocks will be left as normal. When the web server gets the file back from PHP, the entire document will be PHP free. This is illustrated in Figure 2-2.

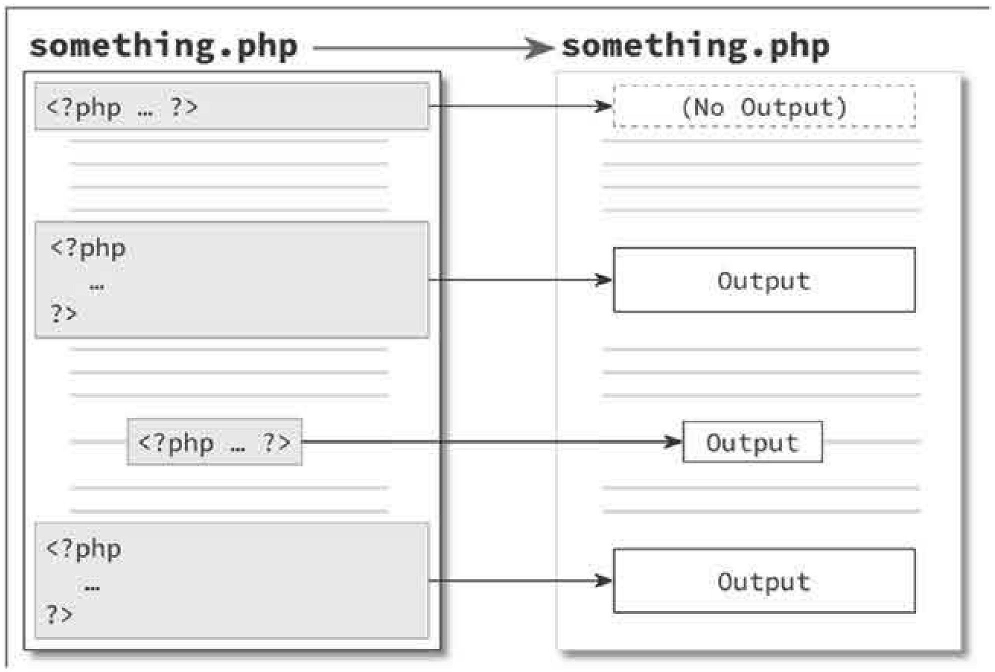


Figure 2-2. *PHP Processing*

In the example, some of the blocks might include some sort of output statement; the first one doesn't. Anything outside the PHP blocks is output.

The point is that the finished product bears no sign of PHP's involvement, and, presuming that you haven't made a mess in the PHP itself, the browser will just get what appears to be an ordinary HTML page.

Renaming the Files

To begin with, we'll rename the `index.html` file to `index.php`. If you reload the sample site, you won't see any difference.

For your convenience, web servers will allow you to omit a page in the URL. If you do, they'll look for a default page. In the `httpd.conf` file, there is a directive listing what the default page will be. For example:

```
DirectoryIndex index.html index.php default.html index.php4
```

The list may be quite long and is in order of preference—if the first page doesn't exist, it will try the next. Only if it can't find any in the list will you get the dreaded `404` missing file error.

Since you've renamed `index.html` to `index.php`, you've only moved it down the list. What's more important is that you have now directed the server to pass the file to PHP for preprocessing.

Later, we can rename the rest of the files from their `.html` extension to a `.php` extension. This can be wasteful of server processing power if the files have no actual PHP script blocks to process, but it is otherwise harmless.

In our project, all of the files will eventually have PHP code in them, so the `.php` extension will be required anyway. Renaming them now means that you won't have to worry about linking to them later.

PHP Scripts

In a very real sense, every PHP document is a single PHP script. This script generally has a mixture of processing and output.

A PHP processing block is contained within PHP tags:

```
<?php
    doSomething();
    doSomethingElse();
?>
```

The spacing of the tags is not important, as long as there is some sort of spacing after the opening tag. For example, you may include simple processing in a single line:

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

What is important is that the tags are not themselves interrupted by space and that they are separated from the actual processing by space. The start of the block has six characters: `<?php` plus a space, tab, or line break. The following are both common errors:

```
<? php dosomething(); ?>
<?phpdosomething(); ?>
```

Any text not inside PHP processing tags will simply be output. PHP processing statements may also include additional output commands, but text outside of the PHP processing tags is always output.

PHP tags have one surprising feature: any space or line break after the closing tag will be removed. That may seem an odd thing to do, but it allows you to put one PHP block below another without adding extra line breaks.

If you're wondering why you want to avoid extra line breaks, some PHP code will cause problems if you (accidentally) output something before the rest of the code is finished. We'll see more on that in later chapters when we discuss HTTP headers.

A PHP script can have multiple PHP processing blocks. These separate blocks are regarded as part of the same script. Logically, you would normally not break up processing into separate blocks, but you can. Generally, you would do this if you have a lot of intervening output which would be more convenient if placed outside the PHP blocks.

Experimenting

From time to time, you might want to experiment with a little PHP to try out a new concept or to see how something works.

The Virtual Hosts application has a **PHP Runner** tab to allow you to try things out (Figure 2-3).

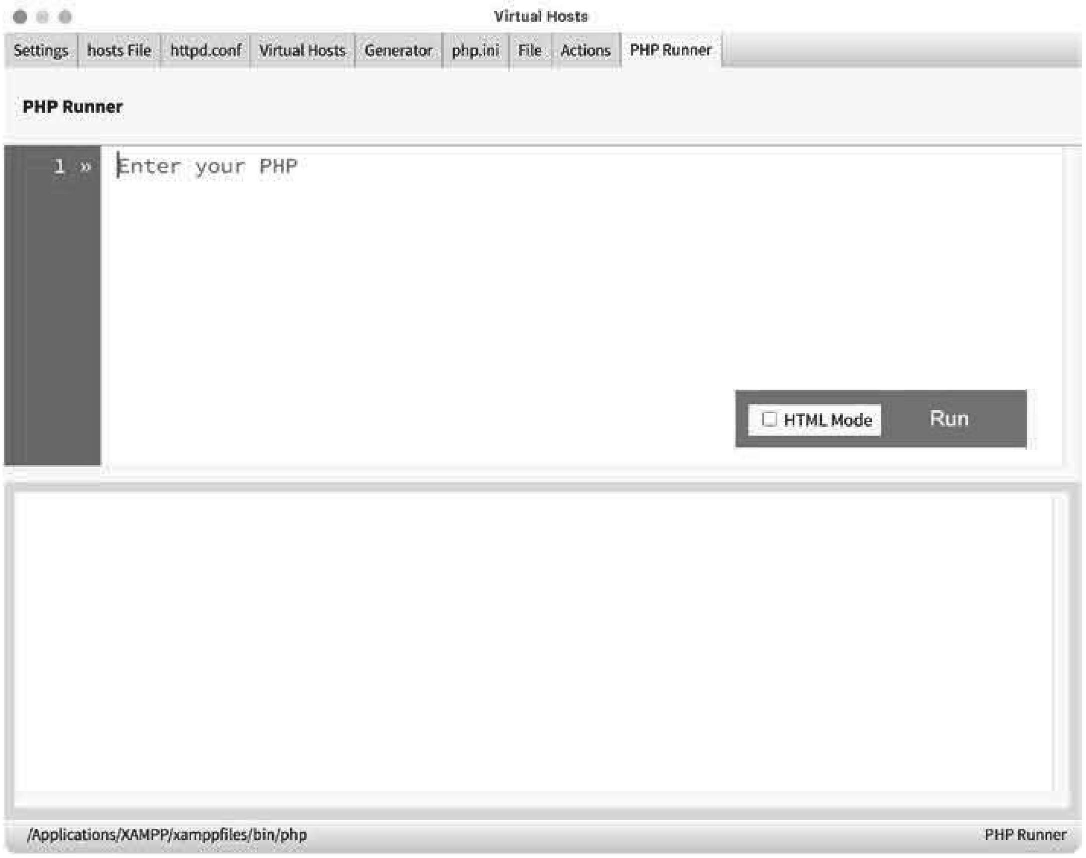


Figure 2-3. *The PHP Runner*

This uses the PHP interpreter which comes with your web server, unless you have gone to the trouble of selecting another. It runs in two modes:

- In non-HTML mode, you write pure PHP *without* PHP tags.
- In HTML mode, you need to include the PHP tags. This allows you to test PHP mixed in with HTML.

Adding Dynamic Data to the Page

Much of the data will come from a database, but PHP is also capable of generating its own data. One example is to display the current date and time in the footer.

It is always good practice to separate the main PHP processing from the HTML output. This can be done entirely in a separate file, but should at least be done by putting your main processing before the HTML part of the file.

Having some PHP processing in the HTML is practically unavoidable, since it will be required when you want to display the data previously processed and also when you need to select different sections of HTML. However, it should be kept to a minimum.

Although PHP was designed to be mixed in with HTML, you really should resist getting carried away. You'll find some code out there in the wild which constantly switches between PHP and HTML code, making the whole a very messy collection of fragments.

We recommend that all of the actual logic and processing code should be written in PHP blocks at the beginning, or even in separate files. That's certainly the approach taken in the book.

The PHP in the rest of the file—where the HTML is—will be mostly limited to using results of the PHP processing gone on before.

To display the date and time, we'll do it in two steps:

- At the beginning, we'll use PHP to generate the date and time.
- In the footer, we'll use PHP to display it.

We'll start by adding a PHP block at the beginning of the file:

```
<!-- Begin head.inc.php -->
<?php

?>
<!DOCTYPE html>
...
```

(The HTML comment `<!-- Begin head.inc.php -->` at the very beginning is for a task we'll be doing later in the chapter.)

PHP has a `date()` function which returns a formatted date, using special formatting codes. This formatted date can be saved for later or displayed using the PHP print statement. The date function takes one or two values:

```
date(format, time)
```

If the time is omitted, the time used will be the current date and time.

The format is a string with some counterintuitive letters. We will use the string:

```
<?php
    $date = date('l, jS F Y g:i a');
?>
```

which means

```
[Day Name], [Day Number][Ordinal] [Month Name][Year]
```

For example:

```
Sunday, 20th July 1969 8:17 pm
```

which is when the Lunar Module first landed on the Moon.

The result of the `date()` function is saved in the variable `$date`. A variable is a named reference to a value. In PHP, the variable *always* has a dollar sign (`$`) at the beginning to indicate that it's a variable. The `$` isn't actually part of the name—it's an indicator.

Variable names are case sensitive: it doesn't matter whether you use upper or lower case as long as you remember. In PHP, it's traditional to use lower case names.

Each PHP statement ends with a semicolon (`;`). Technically, you can omit the semicolon for the very last statement of the script, but that's risky and not something we'll encourage.

Note that this is the date and time on the *server*, where, obviously, PHP is doing the work. To make this clear, you might want to read the server's time zone into another variable:

```
<?php
    $date = date('l, jS F Y g:i a');
    $timezone = date_default_timezone_get();
?>
```

You can get more information on the `date()` and `date_default_timezone_get()` functions here:

- `www.php.net/manual/en/function.date.php`
- `www.php.net/manual/en/function.date-default-timezone-get.php`

The date and time are, of course, dependent on the time zone. Although the real server may be in one part of the world, it's possible to set its time zone to another. This has already been set in the special file `.htaccess` and in `.user.ini`, which are configuration files for the web server.

You can open `.htaccess` and `.user.ini` and change the time zone to your preferred zone. You can get a large list of supported time zones from `www.php.net/manual/en/timezones.php`.

We now need to display it.

Displaying Data

In the footer, we can display the contents of the two variables set up at the top. Currently, the footer has something like the following:

```
<!-- begin footer.inc.php -->
    <footer>
        Copyright © Down Under
    </footer>
<!-- end footer.inc.php -->
```

To print data in PHP, you can do something like this:

```
<?php
    ...
    print ... ;
    ...
?>
```

However, if that's all you're doing in a simple block, you can use a simpler version:

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

The expression will be completely replaced with the output.

Apart from the short PHP output tags (`<?= ... ?>`), PHP has two common output statements, `print` and `echo`. They are *nearly* the same, but not quite. The `print` statement can be used in more complex statements, while `echo` can't. On the other hand, `echo` can output multiple expressions while `print` can't.

Since we won't be pushing either feature very hard, for the most part, it really doesn't matter which one you use.

Most PHP developers end up using the first one they learned about. In this book, we'll use `print`, for the same reason.

We can add this to the footer as follows:

```
<footer>
    Copyright © Down Under<br><?= "$date ($timezone)" ?>
</footer>
```

An expression with double quotes (`" ... "`) is called a **string**. In PHP, you can use single or double quotes, but the double quotes are needed if you want to include variables or some special characters in the string.

A single-quoted string doesn't interpret variables, so it's actually safer (and, technically, less time-consuming). For most of the strings in this book, we'll use single-quoted strings, unless we need variables or special characters. When you use double-quoted strings with variable names, we say that the values are **interpolated** into the string.

If you reload the page now, you will see the date and time, together with the time zone (at the server).

PHP has no way of knowing the time server at the browser end. If it's really important, you can use some JavaScript to set the browser time zone in a cookie.

We'll look at using cookies later.

Next, we'll look at splitting up the page.

Including Files

Much of the page includes content which will be duplicated on other pages. This would include things like the banner and menu, as well as the footer. There's nothing technically wrong with that, but it will be harder to manage if you need to make a small change to this duplicate content. For that reason, we're going to move the content into separate files to be included in this and other pages.

PHP has four variations on a statement to include external content:

- `include` will include a file if it's available; if it's not, well, it will try to cope.
- `include_once` will also include a file if it's available; if it's already been included before, it will just ignore this one.
- `require` will include a file if it's available, but if it's not, the script will grind to a halt with an error.
- `require_once` will include a file if it's available unless it's already been included.

Generally, the `require_once` is probably appropriate: if the file isn't available, you have a serious problem. If the file has already been included, you probably don't want it again. However, we'll still casually refer to this as "include," which is what it's doing.

In our code, there will be five sections which we will extract into includable files:

- The head section comprising the `DOCTYPE` and the `head` element.
- The header section which contains the `header` element.
- The navigation block which is a collection of links in a `nav` element.
- The aside which will contain a random photograph in an `aside` element. It won't appear on all pages.
- The footer section which has the `footer` element, which we have just modified.

If you find the difference between "head" and "header" confusing, you can blame HTML itself. The `head` element contains information about the page, including the title and CSS links. The `header` is one of the newer structural elements added to HTML later along with other structural elements such as `nav`, `aside`, and `footer`. Basically, it contains content you'd expect to see at the top of a page.

We'll now split off these sections and reattach them to the page.

Preparing and Implementing Include

To modify our page, we will need to do the following:

- Cut the existing section of code from the page, and paste it in a new document.
- Save the new document in a suitable location.
- Replace the cut code with a `require_once` statement to include it again.

The actual name and location of an included file is entirely up to you, but the following is recommended:

- Have a special folder for your included files. This will simplify your file management.
- Always use the .php extension for your includable files.

PHP does not require this at all, which is why some people ignore this recommendation. However, if, somehow, a user locates and displays an included file without the `.php` extension, they will see the raw unprocessed PHP code since it won't have gone through the PHP processor. This may include code or data which you don't wish to share with the world at large. This is not good.

The project already has a folder called `includes`, which we can use for locating our included files. Open the file `index.php`, and locate the part of the head section between the comments:

```
<!--    begin head.inc.php -->
<?php
    $date = date('l, jS F Y g:i a');
    $timezone = date_default_timezone_get();
?>
<!DOCTYPE html>
    ...
<!--    end head.inc.php -->
```

Cut the content between the comments and paste it into a new file and save the file as `head.inc.php` inside the `includes` folder.

Replace the cut contents with the following code:

```
<!--    begin head.inc.php -->
<?php require_once 'includes/head.inc.php'; ?>
<!--    end head.inc.php -->
```

You can also delete the comments which are only there as markers.

The name `head.inc.php` is arbitrary. You can call it anything you like, though we recommend the `.php` extension as before. The name follows a simple pattern: the name of the block with `.inc` attached. It's only to make the purpose of the file easier to see.

Sometimes, you'll see a file with just the `.inc` extension. That may or may not work as intended. Certainly, you can include it with any extension you like, but you may not get the protection of running it through the PHP processor.

By default, `.inc` files are not treated as PHP files. However, the following line in either the `httpd.conf` file or the `.htaccess` files will fix that:

```
AddHandler application/x-httpd-php .inc
```

If you reload the page now, hopefully, you'll see nothing different. If you view the page source in the browser, you'll still see nothing different. That means it's working, and PHP has replaced the statement with the output, which is the included file.

Repeat the process for the following files:

- `header.inc.php`
- `nav.inc.php`
- `footer.inc.php`
- `aside.inc.php`

and test your page.

Apart from typing errors, the most common mistake you're likely to make is to forget to save your file before testing changes. Don't forget to save your file before testing changes.

When you've finished, the code should look like this:

```
<?php require_once 'includes/head.inc.php'; ?>
<?php require_once 'includes/header.inc.php'; ?>
<body>
<?php require_once 'includes/nav.inc.php'; ?>
    <main>
        <article id="index">
            ...
        </article>
<?php require_once 'includes/aside.inc.php'; ?>
    </main>
<?php require_once 'includes/footer.inc.php'; ?>
</body>
</html>
```

The remaining PHP pages already have the include statements, so you won't need to worry about them.

Headings and Titles

The problem with including content is that it's all the same. That includes the headings and titles for each of the pages.

If you're wondering about the difference between headings and titles:

- The `title` element appears in the `head` element and is information about the page. You won't see it on the page itself, but it will appear on the browser window title bar (if there is one), the tab name, and on the bookmark if you bookmark the page.
- The various headings appear in the `body` element. In particular, the `h1` element is the main heading of the page and, in this case, appears in the `header` section.

To make this work dynamically, we'll include variables in the `head.inc.php` and `header.inc.php` files to display the content. Then, we'll assign to those variables in the various pages.

Including Variables in the Include Files

We're going to use the following variables for the content:

- The page title will be in the `$pagetitle` variable.
- The page heading will be in the `$pageheading` variable. It will end up in an `h1` element.

For the `title` element, we'll need to make a change to the `head.inc.php` file. Currently, the content includes the following:

```
<title>Australia Down Under<!-- page title --></title>
```

As you see, the comment anticipates a dynamic heading. Modify it as follows:

```
<title>Australia Down Under
    <?php if(isset($pagetitle)) print " – $pagetitle"; ?>
    </title>
```

(The new content doesn't need to go an additional line—it just fits better on this page.)

- The `if()` conditional statement tests whether a condition is `true`. If it is, the following code will run.
- The `isset()` construct tests whether a variable has been set. Here, we're testing whether the variable `$pagetitle` has been set.
- If the variable has been set, we can use it in the `print` statement. Again, the `print` statement prints a double-quoted string, which includes the value of the variable.

For the page heading, we'll modify the `header.inc.php` file. Currently, the `h1` element is hard-coded:

```
<h1>Australia Down Under</h1>
```

We'll also use a conditional statement, but it will be used to replace the whole of the content:

```
<h1><?= isset($pageheading) ? $pageheading : 'Australia Down
Under'; ?></h1>
```

- The conditional operator (`test ? planA : planB`) results in one of two values, depending on the test at the beginning. It's also known as the **ternary** operator, because it has three parts.
- Here, the test is whether the variable `$pageheading` has been set. If so, we'll use its value (*plan A*); otherwise, we'll use the hard-coded value (*plan B*).

Now that we've modified the included files, we can set the values to be used.

Setting the Title and Heading

You won't yet see anything until you set these variables. You'll have to do that for every page, unless you're happy with the default values earlier.

In the `index.php` file, add the following code with the include:

```
<?php
        $pagetitle = 'Home';
        $pageheading = 'Australia Down Under';

        require_once 'includes/head.inc.php';
?>
...
```

We've reformatted the include block, since it's no longer a single statement.

For your convenience, this has been added to every other page in the project. However, it's not fatal if it's been left out, since the included files have allowed for missing variables.

The Navigation Block

Currently, the navigation block is a hard-coded collection of anchors:

```
<ul>
    <li><a href="index.php">Home</a></li>
    <li><a href="about.php">About Oz</a></li>
    <li><a href="contact.php">Contact Us</a></li>
    <li><a href="blog.php">Oz Blog</a></li>
    <li><a href="gallery.php">Animals</a></li>
    <li><a href="admin.php">Administration</a></li>
</ul>
```

To highlight one of the pages, one of those anchors will be replaced with a span element, using CSS to give the appearance of an inactive link:

```
<ul>
    <li><a href="index.php">Home</a></li>
    <li><a href="about.php">About Oz</a></li>
    <li><span>Contact Us</span></li>    <!-- current (no
    anchor) -->
    <li><a href="blog.php">Oz Blog</a></li>
    <li><a href="gallery.php">Animals</a></li>
    <li><a href="admin.php">Administration</a></li>
</ul>
```

A span element is one of the miscellaneous elements available in HTML. The other is the div element. The difference between the two is that span is an *inline* element, while div is a *block* element, which affects where you can use them and how they appear by default.

By default, the span has no special appearance at all. However, if you identify a span, either with an id or class attribute or through its container, you can use CSS to make it look any way you want. In this case, we make it look like an inactive link.

What we'll do is use PHP to generate the collection dynamically. This way, we can decide whether each link should be a real anchor or a span element.

To do this, we'll follow this process:

- Create a collection of links.
- Determine the current page.
- Convert the collection into a collection of either link elements or, if the page is current, a span element.

We will begin by creating the array and converting it into a collection of links. Afterward, we will include the logic for the current page.

The Links Array

A PHP **array** is a collection of keys and values. In many languages, it's always a numbered collection, and the keys are all consecutive integers. Arrays with strings as keys are often called **associative arrays**. In PHP, keys can be numbers or strings, in any order, and you can even have a mix.

To create an array with string keys, we can use the following construction:

```
$array = [
    'key' => 'value',
    'key' => 'value'
];
```

You can, of course, write the statement on one line or as many lines as you need to make it readable.

- The square bracket (`[...]`) notation is used both to define an array and to specify one of the members of the array.
- The `=>` operator has nothing to do with either the `=` or the `>` operators. It's supposed to be a sort of arrow, suggesting that the key points to a value. When we need to give it a name, we'll call it a **thick arrow**; later, you see a **thin arrow** (`->`) which, of course, has a completely unrelated meaning.
- The array elements are separated by a comma (`,`). It's OK to have an extra comma at the end of the list, and it's often done. It has no effect on the collection.

For now, the links array will be derived from the existing unordered list. It will follow the pattern:

```
text => href
```

The associative array will have the link text for the key and the `href` for the value. We will call the array `$links`. Ultimately, the value will be the `href` for the anchors.

Add the following PHP block at the beginning of `nav.php`:

```
<?php
    $links = [
        'Home' => 'index.php',
        'About Oz' => 'about.php',
        'Contact Us' => 'contact.php',
        'Oz Blog' => 'blog.php',
        'Animals' => 'gallery.php',
        'Administration' => 'admin.php'
    ];
?>
<nav>
    ...
</nav>
```

Creating the List of Links

The next step will be to convert the links into an unordered list (`<ul>`). Each list item (except the current page) will have a link in the following form:

```
<li><a href="contact.php">Contact Us</a></li>
```

This will come directly from the contents of the array.

To do this, we will

- Create an empty array of list items
- Iterate through the links array
- Copy the array keys and values into strings for the list items
- Print the item string in the page

To create an empty array of list items is a matter of creating a new empty array:

```
<?php
    $links = [
        ...
    ];
    $ul = [];
?>
```

Before we iterate through the array, we'll prepare template strings for the anchors and span.

In PHP, there's a function called `sprintf()`, which you can pronounce as "s-print-f." It's supposed to print formatted data into a string, but it's easier to think of it as putting values into a template string. It looks like this:

```
sprintf($template, value, value, ...)
```

The template string itself is just a string, but it includes special placeholder codes which will be replaced by the values which follow. Typically, the code is `%s` which means a string.

In our case, we'll use the following template strings:

```
<?php
    $links = [
        ...
    ];
    $a = '<li><a href="/%s">%s</a></li>';
    $span = '<li><span>%s</span></li>';

    $ul = [];
?>
```

Notice that the `$a` string has two placeholders, while the `$span` string has one. We'll need to remember that when we apply them since `sprintf()` requires the number of placeholders to match the number of values.

To iterate through an array, PHP has the `foreach(...)` statement:

```
foreach($array as $key => $value) {
    ...
}
```

You can use `foreach` on other types of collections, such as data which we'll get later from the database. You don't always want the key, and, in some non-array collections, there won't actually be a key. If you want just the values, you can use

```
foreach($collection as $value) {
    ...
}
```

In this case, our `foreach()` looks like this:

```
<?php
    ...
    $ul = [];

    foreach($links as $text => $href) {

    }
?>
```

Within the iteration, we can take the data, run it through the template string, and add the result to the `$ul` array:

```
<?php
    ...
    $ul = [];

    foreach($links as $text => $href) {
        $ul[] = sprintf($a, $href, $text);
    }
?>
```

A statement starting with `$ul[] =` means to add the following to the `$ul` array. The technical term is to push the value on to the array. There's a PHP function called `array_push()` to do that, but there's no point.

When using a statement like

```
$ul[] = something;
```

the biggest mistake you're likely to make is to forget the square brackets and write something like

```
$ul = something;
```

That will end up replacing the variable with a simple string, and you'll have lost your array.

Note that the `$href` variable appears before the `$text` variable: that's the order they are to be used in the `$a` string.

We haven't used the `$span` string yet, but first we'll use the newly generated navigation block.

Displaying the Links

The actual HTML for the navigation block appears as

```
<nav>
    <ul>
        <li> ... </li>
        <li> ... </li>
        <li> ... </li>
    </ul>
</nav>
```

We'll replace the contents of the unordered list with our new array:

```
<nav>
    <ul>
        <?= $ul ?>
    </ul>
</nav>
```

This won't work yet. PHP really hates printing arrays, and you would get a message to that effect if you tried. To print the array, you'll have to join the array items into a single string. You can do that with the implode() function:

```
<?php
    ...
    foreach($links as $text => $href) {
        $ul[] = sprintf($a, $href, $text);
    }
    $ul = implode($ul);
?>
```

The `implode()` function takes an optional *first* value which is a string to put between the joined values. In this case, we've left it out, meaning that they'll be joined with nothing in between. The odd thing here is that normally if a function has an optional value, it comes *after* the other values; here, it comes *before*.

`implode()`? Normally, such a function would be called `join()`, and PHP does have `join()` as an alias of `implode()`. In the past, there was a function called `split()`, which would split a string into an array. However, its implementation proved to be problematic, so it was removed in favor of `explode()` which did a better job.

The `implode()` function is just the opposite of `explode()`.

You can now test the page. Once again, it's all working if you see nothing new.

Highlighting the Current Page

So far, you've gone through a lot of effort just to get something you already had. Now we're going to get PHP to dynamically highlight the current page.

First, we need to know something about PHP's so-called superglobals. These are built-in arrays of special data. There are nine of them:

- `$_GET`, `$_POST`, `$_COOKIE`, and `$_REQUEST` all contain data sent from the browser to the server. We'll be using these when uploading data and storing current values.
- `$_FILES` contains data about files which are uploaded to the server. We'll also use this when uploading files.
- `$_SERVER` has information about the web server and what's happening with the server. This will be very helpful when we need to know about the current page.

- `$_ENV` has information about the running server environment. We won't use this much.
- `$_SESSION` is used to store information between pages. It's particularly useful when managing user logins.
- `$GLOBALS` is a collection of variables.

If you think calling these arrays "superglobals" is a little strange, it's because it's not a very informative name. When you start writing functions, you'll learn more about global variables and why these are "superglobal." For now, just think of them as "magic" arrays.

In PHP, all variable names are case sensitive, and the superglobal names are all defined in upper case. Don't try your luck with variables such as `$_Post` or `$_post`, as they definitely won't work. This also applies to the keys in the arrays.

In particular, the value of `$_SERVER['SCRIPT_FILENAME']` gives us the path of the currently running script.

The value includes the full path, relative to the machine root. That's going to be a problem later, when we want the path relative to the site root. For now, we'll need to extract only the file name, without the rest of the path, using the `basename()` function.

At the beginning of the code, add

```
<?php
    $current = basename($_SERVER['SCRIPT_FILENAME']);
    ...
?>
```

To use this value, we will need to incorporate a test in the code which develops the link: if the `$href` matches the current page, we will use a span element instead of the anchor. Otherwise, we will use the anchor element as before.

For the test, we can use the conditional operator. The test is something like this:

```
href is current ? use span : use anchor
```

In our case, the code is

```
$ul[] = $href == $current
    ? sprintf($span, $text)
    : sprintf($a, $href, $text);
```

- The test `$href == $current` (double equals) compares whether the two variables have the same value. It's a *very* common mistake to use `$href = $current` (single equals). In PHP, that would assign a new value to the `$href` variable.
- Note that we've written the code on three lines to fit better. That's OK with PHP. We've also indented the subsequent lines to make it clear that they're a continuation.

We can add the following code:

```
<?php
    $current=basename($_SERVER['SCRIPT_FILENAME']);

    ...
    foreach($links as $text => $href) {
        $ul[] = $href == $current
            ? sprintf($span, $text)
            : sprintf($a, $href, $text);
    }

    $ul = implode($ul);
?>
```

Comments

No, not comments about this chapter, but how to use **comments** in PHP. There are many times when you need to add notes to your PHP code. These notes should be for the benefit of the human reader only and ignored by the PHP processor.

In PHP, there are three ways to write comments. They can be mixed with ordinary PHP code. For example:

```
<?php
    /*  This is a comment block. */
    //  Single line comment
    #   Another single line comment
?>
```

The comment block starts with `/*` and ends with `*/`. The block can span multiple lines. Note that comment blocks can't be nested.

The single-line comments start with either `//` or `#` and end at the end of the line. From PHP's perspective, it doesn't matter which type you use. We're going to take advantage of that and use the different type for two different purposes.

Some of the things you would use comments for include

- Explain something in your code that's not immediately obvious. You'll need some discernment in how you define what's "obvious."
- Act as headings for sections of code.

- Disable sections of code for testing or troubleshooting. Because we can, we'll use the # for troubleshooting comments; there's no technical reason, but it helps to distinguish what's meant to be temporary.

You must resist the urge to over-comment or explain what should be obvious. Over-commenting just clutters up your code and makes it harder to work with and, frankly, is an insult to whomever is supposed to read your code.

Of course, deciding what is or isn't obvious is a matter of judgment and experience.

We'll use all of these comments as we write PHP code, but for now, we'll just add one comment to illustrate the point. In the PHP block at the beginning of your `index.php` page, add the following comment:

```
<?php
    //  Page Title and Heading
        $pagetitle = 'Home';
        $pageheading = 'Australia Down Under';

    require_once 'includes/head.inc.php';
?>
```

Note that the two statements setting variables have been indented to make it clear that the heading comment applies to them. Using the tab key helps to keep everything neat and tidy.

In the rest of the book, we'll be using comments extensively to help document the code.

If you're familiar with other coding languages, you'll probably find that they all have some form of comments. Of course, they're slightly different. Here are the comment styles you're likely to encounter in your web development:

Context	Comment Style
PHP	`/* block comment */// line comment# line comment`
HTML	`<!-- block comment -->`
CSS	`/* block comment */`
JavaScript	`/* block comment */// line comment`
SQL	`/* block comment */-- line comment`
ini Files	`; line comment`
config Files	`# line comment`

It's very easy to get lost in the comment styles when you're working with multiple languages, as with typical web development.

Summary

This chapter focused on the basics of writing PHP and introduced a few of the basic concepts.

PHP was designed to be mixed in with HTML. In order to reduce wasting time and energy, only files with the `.php` extension will be sent to the PHP processor.

PHP code is written in PHP blocks which are contained between PHP tags. This is even if there is no other HTML in the file. Code which is not inside a PHP block will be passed through, effectively becoming output of the script.