Appendix B

File Storage, File Types, and File Addresses (Absolute and Relative)

Files and Folders

Everything you save to a disk drive is saved as a file of some kind. Since there may be thousands of files on a single disk, computer operating systems provide a file management system that uses folders and sub-folders to organize your files. Chapter 2 includes an introduction to the default organization of user files on Windows, macOS and Linux. You have a great deal of control over how you organize and manage your files and folders in a manner that works for you. This includes: creating, copying, moving, renaming, and deleting files and folders; setting permissions that determine the extent to which individual users can view, modify, or delete specific files and folders; searching for specific files or folders, or file content; and backing up your file system to reduce the risk of data loss. You will usually use your File Manager's graphical interface to conduct these operations, but as a developer you will need to learn how to conduct at least some of these operations from the command line, and also to indicate file names and file addresses in your code. This Appendix will explain more about file addresses, while Appendix C will introduce the command line.

File Storage Locations

Most computing devices usually provide multiple storage options for your files. These options usually include at least one non-removable "fixed" drive with a large storage capacity and fast access times. In addition your device may allow you to attach portable storage, such as USB drives and SD cards. Your device may also be directly connected to other computers via a Local Area Network (LAN), in which case you will probably have access to a centralized file server, designed for shared data storage and file access between users of the same network.

And of course if your device has an Internet connection, you will be able to copy files to and from Internet-based file servers. Internet-based storage is usually referred to as "cloud storage" and cloud storage providers make use of an Internet protocol known as **File Transfer Protocol** (**FTP**) to transfer and manage files and folders. You can use FTP directly if you have an account on an FTP server, for example with a Web hosting service. In fact an FTP server is included in your xampp installation, see Appendix F for more about using FTP to transfer files to and from remote servers.

Wherever a file is stored, its location can be identified by the file's name and its file path, or file address.

Naming Files

A file is identified by a file name followed by a period followed by a file type. While different operating systems may have slightly different rules for naming files, keep in mind that the file may at some point need to be copied to a different file system, and may also need to be referenced in your code, which may be running on a different file system than the system you are developing on. With that in mind, as a developer, it's a good idea to follow naming conventions that are likely to be acceptable to other operating systems, and to simplify file management:

- Don't start or end your filename with a non-alphanumeric character, space, period, hyphen, or (usually) underscore.
- Only use lowercase letters. Since most operating systems are case-sensitive this practice helps to avoid names being mis-spelled.
- Keep your filenames to a reasonable length, for example less than 30 characters. (At the same time ensure the name is long enough to reasonably identify the file's content and purpose.)
- Don't use spaces or (in most cases) underscores, instead use hyphens to separate English words in your file names. Spaces require additional steps to ensure a name will be considered as a whole, while underscores may be misread by users as a space (which is why Google for example recommends using hyphens over underscores when naming files). Hyphens are also preferred over the use of camel case in file names, to avoid the use of uppercase letters.
- When including dates in file names, consider using the convention **year-month-day** (4 digits, 2 digits, 2 digits), for example, **tax-report-2021-12-31.ods**, or **tax-report-20211231.ods**). Consistent use of this convention means that a directory listing will sort files with the same name but different dates in date order.

File addresses

As a developer you will frequently need to be able to reference files in your code, by providing their address, or **file path**. Throughout this book you have referenced files in this way, for example:

- You have used the **action** attribute of the HTML <**form**> tag to specify the name and location of the file that should process the form data when the form is submitted.
- You have used the **src** attribute of the HTML <**img**> tag to specify the name and location of an image that you want to display on your Web page.
- You have used the **href** attribute of the HTML <**a**> (anchor) tag to specify the name and location of a file that you want your Web page to provide a link to.
- You have used the **href** attribute of the HTML <**link**> tag to specify the name and location of an external CSS style sheet.
- You have included an argument in your **fopen**() function calls to indicate the name and location of files that you wish to open for read, write, or append operations.
- You have included an argument in your **include**() function calls to indicate the name and location of files when you wish to add the contents of these files to the content of the current file.

In most (but not all) of these cases, you provided the file name only, for example when you created **circle.html** in Chapter 2, you referenced **circle.php** in a Web form as follows:

```
<form action="circle.php" method="post">
```

The reason that you could reference "circle.php" without indicating its exact folder location is because this file is (hopefully) in the same folder as circle.html. This is an example of a **relative** file address, where the address of a file is indicated **relative to the folder location of the current file**. By contrast **absolute** file addresses provide the complete path to a file. On the Internet absolute addresses are specified as a **Uniform Resource Locator**, or URL, for example, the absolute address for circle.php is:

```
http://localhost/webtech/coursework/chapter02/circle.html
```

You could have used this complete URL with your form tag. This raises the question: when should you use a relative address or an absolute address in your code? Before we consider this, let's first learn a little more about relative addresses.

Relative Addresses

Relative addresses refer to a file's folder location based on the location of the file that is providing the address. The simplest relative address consists of just a file name, which indicates that the file is in the same folder, but relative addresses can reference a file in any location within the same domain as the file making the reference. This is achieved by specifying a file path from the current file location; this file path can include the use of a "**parent folder**" indicator, which consists of 2 periods (..) to indicate the folder in which the current folder is located. The parent folder indicator can be used as many times as needed in a file path, combined with actual folder names, to provide a navigation to the file being referenced. The best way to understand this is to look at some examples; here are paths from the current file location to possible folder locations of a file named **my-image.jpg** to show how this works (note the use of the backslash / as the separator between folder names):

- **my-image.jpg**: the file is in the current folder.
- **images/my-image.jpg**: the file is in the **images** folder which is located inside the **current** folder.
- **media/images\my-image.jpg**: the file is located inside the **images** folder, which is inside the **media** folder, which is inside the **current** folder.
- **../my-image.jpg**: the file is located inside the **parent** folder of the current folder (so for example if the current folder is located inside a folder named **about**, this would mean that **my-image.jpg** is located inside the **about** folder).
- **../../my-image.jpg**: the file is located inside the **parent** folder of the **parent** folder of the current folder (so for example if the current folder is located inside a folder named **about**, and **about** is located inside a folder named **htdocs**, this would mean that **my-image.jpg** is located inside the **htdocs** folder).
- **../images/my-image.jpg**: the file is located in the **images** folder, which is inside the **same parent** folder as the **current** folder (so for example if the current folder is located inside a folder named **about**, this would mean that **my-image.jpg** is located inside the **about/images** folder).
- **../../media/images/my-image.jpg**: the file is located inside the **images** folder which is inside the **media** folder which is inside the **parent** folder of the **parent** folder of the **current** folder (so for example if the current folder is located inside a folder named **about**, and **about** is located inside a folder named **htdocs**, this would mean that **my-image.jpg** is located inside the **htdocs/media/images** folder).

This may look a bit bewildering at first, but it's quite straightforward once you see that, for every ../ in the path you move up one level through the folder system, and come down one level for every folder name in the path, from left to right. This simple addressing scheme, allows you to refer to any file under your domain (usually the root folder of a domain is htdocs) relative to the location of the file that includes the reference. Here are a few examples showing how relative addresses might appear in HTML tags and PHP function calls:

```
<a href="../../about/contact-info.html">Contact Info</a>

<a href="purchasing/order-form.html">Order Form</a>

<link rel="stylesheet" type="text/css" href="../../css/sam-
ple.css"

<form action="testarea/my-first.php" action="post">

<form action="../data/reports/process-report.php"
action="post">

<img src="../../../images/logo.jpg" />

include ("../../common-header.php");

fopen ("../data/wages/weekly-hours.txt");
```

Absolute Addresses

Note that relative addresses are **only** unique within a specific domain, so a file on your Web site might have the same name and relative address as a file on another Web site. There is no ambiguity in this case since relative addresses are always relative to a location within the same domain; if you use a relative address it cannot refer to a file located in a different domain.

An **absolute** file address, on the other hand, provides the complete URL, which is unique across the Internet, and so absolute addresses can refer to a file located in any domain worldwide. We could have written the form tag in circle.php using the absolute address as follows:

```
<form action="http://localhost/webtech/coursework/chapter02/
     circle.html" method="post">
```

(Looking at this you might first wonder: why use a URL and not the file path on the computer where the file is located? It is important to distinguish between the address of a file that is only accessible on a local file system, and the address of a file that is accessible on the Internet. A file address on your local file system needs only to be unique within that system, but a file that is referenced on a Web page must have an address that is unique across the Web, just as your home address must be unique. A URL, is composed of an Internet domain name followed by a file path within that domain, and this ensures that every file across the entire World Wide Web, has a unique address. Since we are coding Web-based applications we need to use complete URL's to indicate a file's absolute address.)

Absolute URL's are **always** necessary whenever a file is located under a **different** domain than the domain under which the application is running. So if you are including an <**href**> link to a page on someone else's Web site, you will need to provide the complete URL. There are also times when you might need to reference a file on a different domain from your <form>, <link>, or <img> tags, or your include() or fopen() statements, and in these and similar cases a complete URL is always necessary. Note that, just as with any links to different domains, any link to a file on another domain that is not under your control runs the risk of that file being changed, modified or deleted (an example would be if one of your <img> tags referenced the URL of a remote image, located on a different Web site).

When to Use Absolute or Relative Addresses

The choice between using an absolute or relative address becomes less clear when the file that is being referenced is located under the same domain name as the file making the reference. Complete URLs could have been used in all of the examples in the "**Relative Addresses**" section above. For example if these files were all located in various folders under **mikeokane.com**, the first two and the last examples could have been written as follows:

```
<a href = "https//www.mikeokane.com/about/contact-
info.html">Contact Info</a>
```

```
<a href = "https//www.mikeokane.com/instruct/examples/pur-
chasing/order-form.html">Order Form</a>
fopen ("https//www.mikeokane.com/instruct/data/wages/weekly-
hours.txt");
```

The choice between using relative and absolute addresses will depend on the nature of the Web site, and workplace policies. Two advantages of using relative addresses are:

Ease of coding: it's a **lot** faster to just code file names and relative file paths than to code an entire URL every time a file is referenced. It's also somewhat easier for another programmer reading your code to identify relative links and see where different files are located on the server.

Ease of transfer between domains: for example if you are developing your code locally, with URL's based on the localhost domain, you will have to edit all these when the application is moved to the live Web server, to reflect the true Web site domain. If you used relative addresses, this would not be necessary since the file paths would remain the same relative to one another.

Now here are two reasons to consider using absolute addresses:

SEO: The main advantage of using absolute addresses is associated with **Search Engine Optimization** (SEO) and will usually apply on larger, or busier, busy Web sites. This is more technical but the general issue is that, if a complete URL is not provided, Google may not be able to associate URL's to the same page that begin with **http://** and **htpps://**, or that may or may not include **www.** This can lead to penalties for duplicate content, and to unscanned pages.

Security: Another possible consideration is that using complete URLs helps to protect your Web site from being copied (especially in the case of larger sites). If all the local addresses are relative, it is quite easy to copy the entire site and run it under a different domain, this becomes a little harder when all the addresses include the domain, not a lot harder but that extra effort can act as a deterrent.

As long as you are working on a relatively small Web site, it is not a major effort to make one decision for now and change your mind later. If you are working for a software company you will probably be told which approach to follow.

Using File Addresses at the Command Line

You will also need a good understanding of file addresses, and relative and absolute addresses, in order to work from your computer's **command line**, accessible through a **Terminal** window. **Appendix C** introduces the command line and you will find it useful to review that material now that you have read Appendix B.