## Introduction

In my last post I introduced the idea of a front-end controller, which acts as the gate-keeper for the web site. In this post, I’ll describe how we can use the apache **rewrite** module to enforce this and reject any request to any other script.

## Mod Rewrite

Apache has a module called rewrite that can inspect incoming request URIs and rewrite them. To use this, first we have to make sure apache is configured to load it. This is configured in the apache configuration file (***httpd.conf***). It is turned on by default in a standard wamp configuration, so it should be on in your home configuration, but it is turned off at **CPIT**. With a standard wamp, you can check by clicking the wamp control panel, choosing ***apache***, then ***apache modules*** and scrolling down the list of modules. You should see a tick against ***rewrite\_module***.

You can check if it is on in any configuration by running phpinfo. See my script info.php or create one as follows:

|  |
| --- |
| <?php  phpinfo();  ?> |

This will show a list of configured modules under “apache2handler, loaded modules”. You should see **mod\_rewrite** towards the end. If it is off, and it will be at CPIT, turn it on by editing the **httpd.conf** file and restarting apache. To enable the module, remove the comment marker (#) in the following row:

#LoadModule rewrite\_module modules/mod\_rewrite.so

At CPIT, you’ll also to edit the path modules/mod\_rewrite.so. Take a look at the other entries. It’ll need to start with a drive letter like Q:. This need only be done once (unless you reinitialise the webdev directory).

## Rewriting URIs

Rewriting can be specified in either the apache configuration or in **.htaccess** files. We’ll use .htaccess because that’s where it’s normally done and, indeed, it’s the only place it can be done in many web hosting companies. The file .htaccess is a text file with no file name and an extension of .htaccess; it is not htaccess.txt!

When a request comes into apache, it begins a complex process of mapping a URI to the script that will be run or the file that will be sent to the user. As it does this, it walks through the directories on the web site level by level. If it finds a .htaccess file, in a directory it will process it, and rules in the file can modify the request URI. Note that if the URI is changed, apache will repeat the whole process of walking the directories using the modified URI. If you’re not careful, an endless loop can be set up here. Apache will throw an error if it detects an infinite loop.

## The .htaccess file

The first thing we need to do in the .htaccess file is to turn on the rewrite engine. We then specify a list of rewrite rules. Apache works down the list, applying each matching rewrite rule.

The basic matching technique uses a regular expression to match the URI. We can extend the logic by preceding the rewrite rule with a number of rewrite conditions. Here’s a basic htaccess file.

|  |
| --- |
| RewriteEngine on  # 1) These files can be served directly by apache  RewriteRule \.(gif|jpe?g|png|bmp|js|css|pdf|html|ico)$ - [NC,L]  # 2) We begin by dealing with recursion.  # ... if we are already in the right form, we're done!  RewriteRule ^website.php/ - [L]  # 3) Kick out any other php files [alternative - just say not found]  # (This will force all access through the front end controller)  RewriteRule \.php$ - [F,NC,L]  # 4) With that out of the way, we'll add our rule:  RewriteRule ^(.\*) website.php/$1 [L,NE]  # All done :-) |

After turning on the rewrite engine, we begin (1) by giving a list of file extensions that can be served directly by apache. The NC flag says “ignore case” and the L flag (last) tells apache it’s done with processing the htaccess file if it gets a match here. The hyphen says leave the URI unchanged.

In the next stage (2) we say that were happy with any request that starts with website.php, the name of our front-end controller. Then (3) we see if the request is for a file ending in .php. If so, we send the user an http forbidden code (the F flag). We’ll tweak this a bit later (why?), but what we have here is enough to get the basic idea over.

Finally, (4) we say that for anything else – ^(.\*) – we put website.php/ on the front of the URI.

## Discussion

We can allow for multiple portals by white-listing them in section two. We’ll add a services portal later to deal with web services. The rewrite module is very powerful and there’s a lot more we can do. For example, we can control ***hot-linking***, automatically force **https**, redirect users with or to a **www** prefix, and much more. Many people get confused by .htaccess, and treat it as *voodoo magic*. To add to the confusion, there are many tutorials out on the internet, written by people who clearly don’t get it. My advice is to stick to the official source:

<http://httpd.apache.org/docs/current/mod/mod_rewrite.html>