Routing

While there are other (mostly legacy) settings in the htaccess file, this is the main one:

RewriteRule !\.(js|ico|txt|gif|jpg|png|css|pdf|mp4|swf|fla|svg|woff|eot|ttf)$ index.php

What this is doing is telling Apache (web.config for IIS does the same thing) that every request that doesn’t end in the list given, is sent through index.php

The challenge with including a working index.php with the framework is that it can be very project specific. However QuickDRY does provide a WebView file which handles the rendering of pages and generation of PDFs. This allows a very generic index.php to be included which also works for command line scripts. You can include ‘index.php’ in your crons.

Example:

<?php

if (isset($\_HOST)) {

define('HTTP\_HOST', $\_HOST);

}

require\_once 'settings.php';

require\_once 'QuickDRYInstance/Defines.php';

require\_once 'QuickDRY/QuickDRY.php';

define('IS\_MOBILE',BrowserOS::IsMobile());

define('GUID', GUID());

require\_once 'QuickDRYInstance/ChangeLogHandler.php';

require\_once 'QuickDRYInstance/UserManager.php';

ExceptionHandler::Init();

$Web = new Web();

$Web->Init('signin', 'admin');

$Web->SetSecureMasterPages([MASTERPAGE\_DEFAULT]);

if (file\_exists($Web->SettingsFile)) {

require\_once $Web->SettingsFile;

} else {

if (file\_exists('../' . $Web->SettingsFile)) {

require\_once '../' . $Web->SettingsFile;

} else {

if (file\_exists('../httpdocs/' . $Web->SettingsFile)) {

require\_once '../httpdocs/' . $Web->SettingsFile;

} else {

Debug::Halt($Web->SettingsFile . ' does not exist');

}

}

}

if ($Web->Server->REQUEST\_URI) {

require\_once 'common/Menu.php';

require\_once 'common/MenuAccess.php';

$Web->InitMenu();

require\_once 'QuickDRY/web/WebView.php';

}

# Include Any Configuration Files

The convention I generally use for settings files is first to have

settings.php

This is an ancient hold over from I can’t even remember. Originally it contained the database passwords and whatnot and was excluded from the repository. As complexity of projects increased, it became necessary to break out other environment agnostic settings such as figuring out the HOST and what HOST the cookies should use. Also, naming the session. Things which will be configured the same way no matter what environment the code is running on.

This file then includes a second, environment specific, settings file. I generally use the convention

settings.host.tld.php

If your project is running on foo.bar.com then your settings file will be

settings.bar.com.php

or

settings.foo.bar.com.php

How you include the settings file may vary. You may be running different environments on separate sub domains. Regardless of how you ultimately figure out which settings file to include, it will contain ONLY defines for connection information to data sources and mail servers as appropriate. You can also configure error display settings here and other environment specific things.

Other things I may include here are an “IN\_PRODUCTION” flag so that things that aren’t production ready can be pushed to production while not being made available to end users. In complex projects there may be a large number of bugs being worked on and there may be a lot of changes that multiple bugs depended on.

As you’ll see later, the conventions used in QuickDRY makes it very easy to create things that can be worked on in DEV while the code sits idle in production until it’s fully vetted.

# Include Any Dependencies

There are two schools of thought on this. One is that PHP is a scripting language so you include things as needed. You’ll see no end of projects with “include” all over the place and the same file included 100 times with an “include\_once” or “require\_once” used to avoid problems.

The other is that I don’t care if it’s a scripting language, if you want maximum performance then you precompile. And in those cases, you include everything up front. The web server will cache all the code for the site on the first run and from then all, there is no more compiling. That is how .Net works. That’s how FastCGI works. But when you are selectively including things, it makes it harder for the cache to be used.

I generally use

modules.php

as the first file included. It in turn includes the QuickDRY framework, common\_modules.php and then any data source classes generated by the framework. Those generated files will be covered later.

common\_modules.php

includes any custom files in the common folder.

By having everything included at once, it’s cached, and ready to go for any crons or pages you create.

# Initialize the user object from the session if applicable and available

\*NOTE\* Most of this is wrapped up in the QuickDRY framework so it is unnecessary to do this yourself. This is here to explain what it’s doing and why

QuickDRY includes a UserClass class in QuickDRY/web

One of the key reasons for having a Session class to wrap the $\_SESSION global is so that things stored in the session are serialized. This allows you to store objects in the $\_SESSION which PHP doesn’t allow by default. You also don’t have to use isset everywhere to avoid errors. By default $Session will return null if no value is stored for the requested property.

*/\** ***@var*** *$CurrentUser UserClass \*/*$CurrentUser = **null**;  
**if**($Session->**user**) {  
 $CurrentUser = $Session->**user**;  
}

Retrieving the CurrentUser is as simple as grabbing it from the Session object. The Session object will be talked about later.

# Translate the URL path to a file path

$qs = $\_SERVER[**'QUERY\_STRING'**];  
$ru = $\_SERVER[**'REQUEST\_URI'**];  
  
*define*(**'JSON\_REQUEST'**, *stristr*($ru,**'.json'**) !== **false**);  
  
  
$page = *str\_replace*(**'?'** . $qs, **''**, $ru);  
$page = *str\_replace*(**'/'** . $qs, **'/'**, $page);  
  
**if**(*strstr*($page,**'/'**) === **false**)  
 $page .= **'/'**;  
  
**if**($page[*strlen*($page) - 1] == **'/'**) {  
 $page = *substr*($page, 0,*strlen*($page) - 1);  
}  
  
$full\_path = $page != **'/'** ? $page : **'/main'**;  
$t = *explode*(**'/'**, $full\_path);  
$cur\_page = $t[*sizeof*($t)-1];  
  
  
  
**if**(!$cur\_page) {  
 **if**($CurrentUser && !$CurrentUser->Is(***ROLE\_ID\_RESPONDENT***)) {  
 $cur\_page = **'admin'**;  
 $full\_path = **'/admin'**;  
 } **else** {  
 $cur\_page = **'main'**;  
 $full\_path = **'/main'**;  
 }  
}  
  
*define*(**'CURRENT\_PAGE'**, $full\_path);  
  
*define*(**'CURRENT\_PAGE\_NAME'**, $cur\_page);  
  
$page\_alt = **'pages'** . ***CURRENT\_PAGE*** . **'/'** . ***CURRENT\_PAGE\_NAME*** . **'.php'**;  
$code\_alt = **'pages'** . ***CURRENT\_PAGE*** . **'/'** . ***CURRENT\_PAGE\_NAME*** . **'.code.php'**;  
  
$page = **'pages'** . ***CURRENT\_PAGE*** . **'.php'**;  
$code = **'pages'** . ***CURRENT\_PAGE*** . **'.code.php'**;  
  
$controller = *file\_exists*($code) ? $code : $code\_alt;  
$view = *file\_exists*($page) ? $page : $page\_alt;

<http://foo.bar.com/test>

will map to

pages/test/test.php (view)

and

pages/test/test.code.php (controller)

OR

pages/test.php (view)

and

pages/test.code.php (controller)

It is not necessary to put the files in the folder of the same parent name. But it is preferable to keep the project tidy. Depending on the complexity of the page there may be other files that need to be included that are only relevant to that particular page. The page may be broken into multiple sections to simplify setting up the view.

pages/test/test.php

pages/test/test.code.php

pages/test/classes/foo.php

pages/test/sections/bar.php

For example.

Then test.code.php will include the classes and test.php will include the sections.

Note that in this example routing code some business logic is included to select a default page for <http://foo.bar.com/> depending on the role of the user.  
  
This is why index.php is not a simple addition to the framework. One of the challenges of writing a framework is separating the features that change from the features that don’t so the user does not have to wade through a lot of unnecessary code to get the frameset set up for their project.

One school of thought is that the actual php files that define a page should not align neatly and predictably with the URL of the page.

We’ll get into more of how the code and view are defined to avoid security holes later.

# Include the actual file that the URL is requesting

Once the actual file is found that the user is requesting based on the URL, it’s time to include it

*ob\_start*();  
  
Metrics::*Start*(**'Controller'**);  
**if**(*file\_exists*($controller)) {  
 **require\_once** $controller;  
  
 **if**(**isset**($PageModel)) {  
 $PageModel->Init();  
 $\_MASTERPAGE = $PageModel->**MasterPage** ? $PageModel->**MasterPage** : **null**;  
  
 **switch**($\_MASTERPAGE) {  
 **case 'user'**:  
 **case 'respondent'**:  
 **case 'default'**:  
 **default**:  
 **require\_once 'common/menu.php'**;  
 **require\_once 'common/MenuAccess.php'**;  
 **require\_once 'QuickDRY/controls/navigation.code.php'**;  
 **break**;  
 }  
  
 **switch**($\_SERVER[**'REQUEST\_METHOD'**]) {  
 **case 'GET'**:  
 $PageModel->Get();  
 **break**;  
 **case 'POST'**:  
 $PageModel->Post();  
 **break**;  
 }  
  
 **if**($Request->**export**) {  
 **switch**(*strtolower*($Request->**export**)) {  
 **case 'xls'**:  
 **if**(*method\_exists*($PageModel, **'ExportToXLS'**)) {  
 $PageModel->ExportToXLS();  
 } **else** {  
 **exit**(**'ExportToXLS Not Implemented'**);  
 }  
 **exit**;  
 **case 'json'**:  
 **if**(*method\_exists*($PageModel, **'ToJSON'**)) {  
 $PageModel->ToJSON();  
 } **else** {  
 **exit**(**'ToJSON Not Implemented'**);  
 }  
 **exit**;  
 }  
 }  
 } **else** {  
 **require\_once 'common/menu.php'**;  
 **require\_once 'QuickDRY/controls/navigation.code.php'**;  
 }  
}  
Metrics::*Stop*(**'Controller'**);  
  
Metrics::*Start*(**'View'**);  
**if**(*file\_exists*($view)) {  
 **require\_once** $view;  
}  
Metrics::*Stop*(**'View'**);  
  
$\_PAGE\_HTML = *ob\_get\_clean*();

Note here that the controller is included first. While not required it is recommended that your control define a class that extends BasePage and be instantiated with the variable $PageModel

Also note that some business logic is included here to decide whether to include the navigation. It’s very likely customization will be required in your own project.

Two very important things are happening here. The first is that ob\_start is being used. This captures all the output of the controller and view rather than sending it directly to the browser. This allows for a lot of things but one of the primary ones is that it allows the controller to use redirects without any chance of “output already sent to browser” errors.

The secondary purpose is to allow the output to be easily shoved into a master page. The output is captured to $\_PAGE\_HTML which can then have a master template wrapped around it and it can also be rendered to a PDF or emailed out, or any number of things.  
  
PHP output really should never go directly to a browser until the document is fully rendered.

Which brings us to the last part

# Render it to the user

**if**($Session->**pdf**)  
{  
 Metrics::*Start*(**'render pdf'**);  
 **switch**($Session->**pdf\_lib**)  
 {  
 **case 'webkit'**:  
 **default**:  
 **require\_once 'QuickDRY/pdf\_output/webkit.php'**;  
 **break**;  
 }  
  
 Metrics::*Stop*(**'render pdf'**);  
 **exit**();  
}  
  
**require\_once isset**($\_MASTERPAGE) && $\_MASTERPAGE ? **'masterpages/'** . $\_MASTERPAGE . **'.php'** : **'masterpages/default.php'**;

There is a “magic” session variable named “pdf” that will render any page to a PDF rather than as a webpage. Depending on your project, you may want to exclude this or add security to avoid excessive server usage.

If that isn’t set, then by default a master page is used. The master page includes all the global JavaScript and style sheets that the site uses and it sets the overall layout of pages. Any page can use any compatible master page just by setting $\_MASTERPAGE in the .code.php file or using $this->MasterPage when using the PageModel technique.

It can also include the footer, and global controls like the “Notice” dialog that’s commonly used to notify users of various things. The included controls will be covered more later.

One of the common features in master pages I use is including css and javascript specific to the page being views.

**<?php if**(*file\_exists*(**'pages'**.***CURRENT\_PAGE***. **'/'** . ***CURRENT\_PAGE\_NAME*** . **'.css'**)) { **?>** <**link rel="stylesheet" type="text/css" href="/pages<?php echo *CURRENT\_PAGE***. **'/'** . ***CURRENT\_PAGE\_NAME*** . **'.css'**; **?>"** />  
**<?php** } **?>**

**<?php if**(*file\_exists*(**'pages'**.***CURRENT\_PAGE***. **'/'** . ***CURRENT\_PAGE\_NAME*** . **'.js'**)) { **?>** <**script type="text/javascript" src="/pages<?php echo *CURRENT\_PAGE***. **'/'** . ***CURRENT\_PAGE\_NAME*** . **'.js'**; **?>"**></**script**>  
**<?php** } **?>**

Notice that they reference /pages in the URL. In the htaccess file, css and js files are not put through the routing. If you want your page to have some custom JavaScript or custom css simply name the files the same name as the page.

pages/test/test.code.php

pages/test/test.php

pages/test/test.css

pages/test/test.js

will all be included automatically when the user goes to

<http://foo.bar.com/test>

.code.php, .js and .css are all optional files.

Master pages will be covered in more detail later.