Zend Framework Quick Start Improved
By Paul Albinson BSc (Hons), FdSc, MBCS

Introduction

The code I use for this tutorial is available on
https://github.com/paulalbinson/zf-quick-start-improved

The official Zend Framework (ZF) quick start tutorial is an excellent start to creating your first Zend Framework web application but there are some things that I found that can be simplified and improved on it. I will therefore clarify some of the tutorial, identify areas for improvements and how to overcome some problems I encountered with the aim of helping you understand and use ZF better.

Follow the ZF quick start guide with the following modifications and additions.

Before you start download the full Zend Framework zip package and extract add it to somewhere useful such as your website folder of your server e.g. htdocs or wwwroot. Next add the path to the bin folder within this to the Windows path as this allows us to use the Zend Framework tool (known as zftool) from anywhere on our system; I assume there is a similar system for other operating systems.

Create the Zend Framework project using the ZF Tool like it shows you in the guide.

Copy the following from the Zend Framework folder into the applications library folder:

- The Zend folder from the library folder
- The ZendX folder from extras > library

Zend contains all the important Zend Framework files and ZendX is extra classes and functions you may wish to use.

Doctype, encoding and charset

The guide suggests setting up the doctype for the website (e.g. HTML5) in the bootstrap, I prefer to add it in the application.ini config file as it is a setting and the config is a more logical place for settings in my opinion. We do this by adding:

```
resources.view.doctype = "HTML5"
```

And while we are there lets set the encoding and charset with the following:

```
resources.view.encoding = "UTF-8"
resources.view.charset = "UTF-8"
```

There we go 3 important settings for the project all set without having unnecessary code in the bootstrap.

Now add the code into the layout script from the quick start guide and we have a basic layout (we will improve on it later). You will see if you view the source that the HTML5 doctype (or whatever you chose to use) has been added as this is what we set in the application.ini config file and it is outputted in the layout with:

```
echo $this->doctype();
```

Dealing with errors

I noticed that the error code it refers to as what the ZF tool creates by default for ErrorController is not what is actually created. The code ZF tool creates is a lot more complex and prepares for logging of errors which I will explain how to use this properly later in this article.

Also I see that the code it says is what is the default contents of the view script (error.phtml) isn't actually what the tool creates so make sure you update it to what the code from the quick start guide.

Also there is no need to add layout code in the view script as layout is added automatically.

// When showing the view script add the code from later in the git as realised needed to improve error.phtml to take out unnecessary error code

Databases

The database creation query didn't seem to work but is solved by using outer double quotes and inner single quotes like:

```
"adapter=PDO_SQLITE&dbname=APPLICATION_PATH '/../data/db/guestbook.db'" production
```

Don't forget to create the folder for the SQLLITE databases to go in.

In the load.sqlite.php file when defining the APPLICATION_PATH it should be

```
realpath(dirname(__FILE__) . '/../../application'
```

not

```
realpath(dirname(__FILE__) . '/../application'
```

This is because we are in /data/db and need to go up to /application so need to go back 2 not 1.

Sign Guestbook Form

I recommend adjusting the string length on the comments field as 20 characters max is very short, I made mine minimum 10 and maximum 200.

I am not a fan of Zend Forms as they seem to make things overly complicated for complex forms for example only show questions based on other answers e.g. if they said yes on question 1 then show then questions 2 & 3. They do however make validation and filtering easier so it depends on your preference.

Expanding on the Quick Start example

Well that is the quick start guide complete with some improvements made and tweaks to make it work but there are a few other modifications that will help when expanding it into further things.

Index.php

In the index.php file in the public folder add the models path to the set_include_path to include the models folder so that the application can find the models folder easier. This shouldn't really be needed but solved an issue I was when I started using models more so it is well worth adding. Your set_include_path will then look like this:

```
set_include_path(implode(PATH_SEPARATOR, array(
   realpath(APPLICATION_PATH . '/../library'),
   realpath(APPLICATION_PATH . '/../application/models/'),
   get_include_path(),
)));
```

While we are in this file you will see it looks for a constant variable called APPLICATION_ENV and if it doesn't exist it gives it a value of production. The value is usually set in the virtual host for the site in the apache which enables you to define the value to development for when you are running the site on your development server and set it to production on your production server. The idea being you can do different things per environment you are in e.g. on development server where only your developers, testers etc. see the site we can output debug errors to the screen whereas we wouldn't want this happening on the live/production server. Unfortunately I can't seem to recreate this on IIS but you could instead set the value manually dependant on IP address. For example you could put the IP addresses of all development machines in an array and check for them and of they are in the array define development and if not set production; here is some code for this:

```
if (in_array($_SERVER["REMOTE_ADDR"], array("127.0.0.1", "192.168.1.2", "192.168.1.5"))):
   $applicationEnv = "development";
else:
   $applicationEnv = "production";
endif;

defined('APPLICATION_ENV')
      || define('APPLICATION_ENV', (getenv('APPLICATION_ENV') ? getenv('APPLICATION_ENV') :
$applicationEnv));
```

Obviously the original approach of setting it in the virtual host is a much better idea but if you are on IIS or just simply want to set the environment based on IP address then this is one such approach you could take.

Fixing error output

The view script (error.phtml) for the error action (in the error controller) has the following:

```
<?php if ('development' == $this->env): ?>
        The HTML & PHP to output the errors goes here
<?php endif; ?>
```

which is designed to only show details of the errors when the site is used in the development application environment but $this->env doesn't have a value so the details aren't shown. The idea is that the value should automatically be available to the view because the FrontController should know about it somehow but this hasn't been set. You can either add it in the Bootstrap after you have set up the front controller or instead just get the value from the constant variable as it is available to the script so is easy to use. The setting up of the variable in the front controller is a bit too advanced for this article and I will cover this in a future article. Therefore I will choose the option of looking at from the APPLICATION_ENV constant variable as it is available for use anywhere so let's use that. Therefore modify the if statement to the following:

```
<?php if (APPLICATION_ENV == 'development'): ?>
        The HTML & PHP to output the errors goes here
<?php endif; ?>
```

Logging errors

The error action (part of the error controller) contains some code for logging errors assuming logger is available which wasn't added in the ZF quick start guide but is really easy to do which I will now explain. First create a folder called logs in the root folder of the website i.e. on the same level as application. Next add the following to the production section of application.ini

```
resources.log.stream.writerName = "Stream"
resources.log.stream.writerParams.stream = APPLICATION_PATH "/../logs/application.log"
resources.log.stream.writerParams.mode = "a"
resources.log.stream.filterName = "Priority"
resources.log.stream.filterParams.priority = 4
```

This says we wish to use Zend Log to do the logging with a Zend_Log_Writer_Stream to write to the log. We specify where it will go, what mode to use (a is append existing log if it exists, if not it adds one) and also set is the name for the steam. Also included is the priority which is the level of severity of the error and this goes from 1 (Emergency: system is unusable) to 7 (debug) and you can choose which level and below to log e.g. if you choose 4 (warning) it will log errors of priority 4,3,2 & 1. Warn level seems reasonable for a production environment but you may wish to log the less important errors as well such as debug for debugging in your development environment; to do this add the following in the development part of application.ini you can add the priority you want there e.g.

```
resources.log.stream.filterParams.priority = 7
```

For more information on logging see:

- http://framework.zend.com/manual/en/zend.application.available-resources.html
- http://framework.zend.com/manual/en/zend.log.factory.html
- http://framework.zend.com/manual/1.11/en/zend.log.overview.html

HTML5

As we set the doctype to HTML5 we should also take a look at the layouts and scripts HTML5. In the code I added on GitHub I have used HTML5 and also improved the styling with CSS. I also adjusted the form code so it outputs nicer HTML which is more semantically correct.

ReCaptcha

The default Captcha of Zend Forms isn't very easy to read and doesn't look very professional. You can choose different captcha types should you wish and I chose to change it for ReCaptcha as you will see in my code. The use of this is very similar so I hope it is obvious how this works.

Starting to customise the bootstrap

The bootstrap is currently empty as everything we have needed to configure up to this point was set via the application.ini configuration file. To enable us to modify the way the application works such as different layouts per module, authentication etc we use the Bootstrap class.

By default when an application is launched it calls the run function in the bootstrap class which is done via the index.php file in the public folder and which all requests go through as dealt with via the .htaccess file (apache) or web.config (IIS). When you set up a project the run function doesn’t exist although the index.php file calls it but surprisingly it works; I assume ZF says if run function exists i.e. the user wishes to use custom code then use it and if not use these default settings. Whether you customise the way a ZF application loads using the bootstrap (usually via the run function but you can specify an alternative function) or not the application will look at the settings and setup the required functionality.

First as we don’t have the run function and this is the entry point to the application that is called by index.php let’s create it

```
public function run() {

}
```

Now that we have added code to the bootstrap and chose to use our own run function then the dispatching of a page (i.e. now show us the page) is not done automatically so we need to add code to do this. I will add these in separate functions and call it from the run function like so:

```
public function run(){
        $this->bootstrap('FrontController');  // Initialise the front controller
    $front = $this->getResource('FrontController'); // Get the front controller

    // Ensure that the bootstrap is a parameter in the front controller
    if ($front->getParam('bootstrap') === null) {
      $front->setParam('bootstrap', $this);
    }

    self::_dispatch();
  }

  protected function _dispatch() {
// Display an error if we can’t find the front controller
    if ($this->frontController == null):
      throw Exception('The front controller has not been initialized.');
    endif;

    $this->frontController->returnResponse(true);
    $response = $this->frontController->dispatch();

    $this->_sendResponse($response); // Call function to execute page/put on screen
  }

  // Specifies what to do with a response - It specifies it as a http response, adds a header, and
sends it.
```

```
  protected function _sendResponse(Zend_Controller_Response_Http $response) {
    $response->setHeader('Content-Type', 'text/html; charset=UTF-8', true);
    $response->sendResponse(); // Execute page/put on screen
  }
```

What this does is say get and setup the front controller, say we wish to return a response i.e. we want to display things to the screen (you may want an application to never output anything so could specify false instead). We also specify a header to say the content will be text/html and use the UTF-8 character set.

This has now given us the basics for more customisation via the bootstrap.

Base URLs

It is good practice to use the baseUrl function to define where the root of the project is, this is typically / i.e. http://www.example.com/css is /css. However what if you later moved it into a sub folder to for example host multiple ZF sites on the same domain e.g. http://demos.example.com. In this situation the base would then be /folder-name not / like usual. By using baseUrl you can easily move the site around or create an alias of it and all you need to do is update the baseUrl and the sites works in the new location. So how do we do this?

Apparently the base URL should automatically find any folder defined as RewriteBase in an apache .htaccess file but I found this isn't the case; therefore if this happens or you need to manually set the baseUrl in the bootstrap and you can do so like this.

```
    if ($_SERVER["SERVER_NAME"] == "demos.example.com"):
        $front->setBaseUrl("/zf-quick-start-improved");  // Set the base URL
    endif;
```

What this does is say if we are on the domain demos.example.com then get the front controller and set the baseUrl (part of the front controller). You could define as many of these as you want dependant on the various locations you show the site and therefore set their related folders. Of course if it was only in one place you wouldn't need the if part; also when it isn't in a sub folder you don't need it at all as / is the default base URL. For this example I would put it in the run of the bootstrap under where we got the front controller (after all baseUrl is part of the front controller).

Now we have set the baseUrl (if required) we need to make use of it.

// Add updated bootstrap code here

It is also worth adding a base tag to the head of each layout with the following PHP code:

```
$baseUrl = $this->baseUrl();
if (!empty($baseUrl)): echo '<base href="'.$baseUrl.'">'; endif;
```

// Add layout file code here

I know this seems a lot of extra work "my site will probably always be in the root of the folder so why should I do the extra work?" and this was my initial thoughts too. I thought it seems unnecessary when I can just put the required URLs in the code. However I then decided to move all my demos into one sub domain http://demos.paulalbinson.info so all my demos in one place rather than having may sub domains (plus my domain provider at the time restricted the total sub domains

I could create). Getting the sites into a folder was a little tricky but I found the solution to be the Apache mod_alias module and the site worked in the folder while still being a standalone project thus meaning very little change to the project code to achieve this. I will explain more about how I did this is in another article. So I had the site working in the folder but because it was now in a sub folder all the links were wrong and links to resources (CSS, JavaScript etc.) were pointing to the wrong location. This meant that all the links needed updating whereas if I had used baseUrl like shown above all I would have had to have done is say if in demos site then the baseUrl is / zf-quick-start-improved (zf-quick-start-improved is the folder I put the project in, which in reality was an alias).

Outputting URLs

Now we have set base URLs and ensured the links to required files always includes the correct base URL we now need to deal with links within the pages, images etc.

Probably the easiest way to do this would be to use the baseUrl function around the link like so:

```
<a href="<?php echo $this->baseUrl('somewhere/else'); ?>" title="Somewhere else">Somewhere
else</a>
```

Or

```
   <img src="<?php echo $this->baseUrl('img/something.png'); ?>" alt="Image of something"
title="Image of something">
```

Etc

That is the basic approach and works well especially for things like images etc (it would put the required / or /folder-name/ etc. in for us dependant on where it is).

Of course as we set the base tag we can use relative URLs as another approach but I prefer fixed URLs as they are more reliable as it doesn't matter where you are they work and there is no chance of clashes with the same named file elsewhere.

Another approach for URLs which is mentioned in the official Zend Framework QuickStart guide is the URL function. Unfortunately the ZF QuickStart guide doesn't explain it that well and I can't seem to find proper documentation for it. I did however find an example online (http://blog.ericlamb.net/2010/04/zend-framework-url-view-helper/) which explains it well. In summary you can link to specific actions, controllers etc. by specifying the module, controller and index and it outputs the correct URL to for use in a link complete with the base URL included. Therefore it doesn't matter where the site is (in a sub folder or root) or even if it moves locations, as long as the base URL is correct it will deal with finding the correct link.

Also if what you are linking to is a route you can just refer to its name and any variables and values and it will work out the correct link like so:

```
        echo $this->url(array('id' => 1), 'admin-users-edit', true) ;
```

This would output the route admin-users-edit with the id set as 1.

The advantage of referring to routes by name is that should you change the routes path all the links referring to will be automatically updated (hooray, no need to manually update them).

Some things to remember:

- When you are linking to a controller not a route then set the route name variable to null
- When linking to a route you don't need to specify the module, controller or action (the route has this stuff)
- If you don't want the variables in the current pages URL added to the links in the page then set the reset variable to true

All this may seem unnecessarily complex but as I explained before it is good practice and allows for flexibility of where the site is located.

All done

That's it for now; we have covered the basics to create a Zend Framework MVC website.