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

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.

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.

Let's improve it further
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.

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