

Tuesday, April 29. 2008

An Example Zend Framework Blog Application - Part 3: A Simple Hello World Tutorial

It's almost obligatory when introducing a new programming topic, that the author present the simplest possible example. Usually this means getting a programming language or framework to print "Hello World" to the screen. I'm going to be no different. So much for originality...

Previously: [Part 2: The MVC Application Architecture](#)

Step 1: Creating A New Application

Before we jump into programming with the Zend Framework there are a few setup tasks. We need to install the Zend Framework, get a virtual domain running, and have a basic collection of directories and empty files created to hold our source code.

You'll need to download the Zend Framework 1.5 (latest minor release) from <http://framework.zend.com> and put it somewhere on your include path. I try to minimise the number of include paths I end up having so I sometimes pick very odd places (like the PEAR directory which is usually already in the include_path) or another central library location like PEAR where I might have a dozen libraries all congregated.


A common solution found on blogs and articles is to put the framework into a "library" directory within your application directory tree. Ordinarily I don't recommend this unless your application needs a specific version of the Zend Framework - centralising a Zend Framework location for multiple applications to access can make maintenance a bit easier.

I've settled on using a virtual host mainly because it's easy to setup, and it helps avoid some of those annoying base path issues you get in your HTML when using mod_rewrite pretty urls - especially in development where, otherwise, the localhost document root becomes a mass of sub-directories upon sub-directories that end up causing base href issues so easily. Usually I open up the virtual-hosts.conf file from Apache's conf directory and add something like:

```
NameVirtualHost &lowast;:80
```

```
<VirtualHost &lowast;:80>
DocumentRoot "/path/to/htdocs"
ServerName localhost
</VirtualHost>
```

```
<VirtualHost &lowast;:80>
ServerName zfblog
DocumentRoot "/path/to/project/trunk/public"
</VirtualHost>
```


This should result in all HTTP queries for <http://zfblog/> reaching the selected document root (which is the public directory for the project containing index.php, located in our subversion's working copy "trunk" directory). We'll create this directory structure a little later. We usually also need to reference localhost to ensure it's recognised as a host correctly - otherwise any localhost addresses you already have will suddenly divert to zfblog's document root miseriously 

For those of you on Windows XP or Vista, you also need to add the new virtual domain to your hosts file (usually at C:\WINNT\system32\drivers\etc\hosts). This is tricky in Vista - you will probably need to alter the file's permissions using an Administrator account to grant the local user additional Modify privileges. Make sure to reverse the privileges after! Edit it to look this:

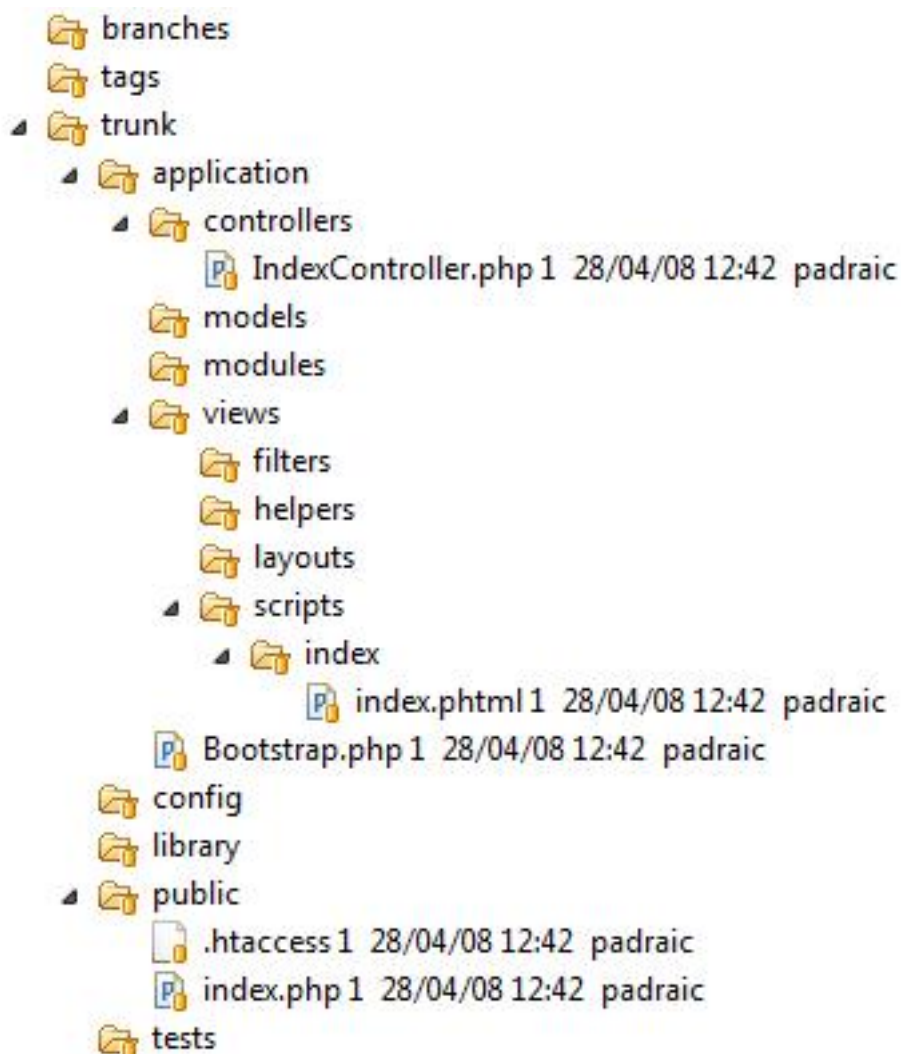
```
# Copyright (c) 1993-1999 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.

127.0.0.1 localhost
127.0.0.1 zfblog
```

Restart your browser and with a little luck the browser will at least give us a directory listing (once you add the document root "public" directory as shown below).

The least exciting part is the creation of a tree of directories to contain our source code. Wil Sinclair over at Zend is working on a Zend_Build component and this will one day hopefully eliminate too much of the repetition in creating these directories and default files. Until then let the monotony ensue (or keep a skeleton setup handy for copy and pasting ).

Here's a snapshot of what the Hello World example looks like in the directory view from Eclipse:



Copy this and create the same directory structure in a new directory called something like "zfblog". Putting it all in "trunk" is optional - I've using subversion here which is why that directory exists. If you haven't used Subversion before I highly recommend it - it's simple to learn and use.

Let's examine each of these directories briefly.

The "application" directory is where we place all the components of an application implementing the Model-View-Controller. Inside "application", "controllers", "models" and "views" represent respective locations of controller, model and view source code and templates. Inside "views" there is a subdivision between "filters", "helpers", "layouts" and "scripts". For now, remember we put all templates rendered by the View inside "scripts". In the screenshot, we see an index directory which will contain an "index.phtml" file, which is the view template for indexAction of an IndexController class. The others we'll meet later.

The "modules" directory is also for controllers, models, and views, but in this case they are categorised into multiple divisions of an application. If you think about it, an application can be broken into several parts, for example, the main application and an administration part. The administration section could be partitioned into the Admin Module so it's not intermixed with the main application.

A Bootstrap.php file exists in the "application" directory which represents a class called "Bootstrap". Its purpose is to initialise the Zend Framework, adjust environment settings (for example, timezone and error_reporting level), and otherwise make application specific tweaks and additions before a HTTP request is processed. Most tutorials take an alternative view and put Bootstrap code into index.php. I strongly suggest you avoid this and use an actual class to organise the Bootstrap code - it makes it a lot easier to read if nothing else!

The "config" directory simply holds the configuration data for the application, for example, database connection details.

The "library" directory can hold a copy of the Zend Framework. Generally I avoid this since it's fairly easy to just add the Zend Framework to the PHP include_path, but it's still very useful if you want to change Zend Framework versions in use more flexibly. It's also simple to manage if you use it to add an svn::external reference to the Zend Framework subversion repository.

The "public" directory holds all public files accessible by a request to the web server. This includes an index.php file, which handles all application requests by calling on the Bootstrap class, as well as any CSS, HTML, Javascript files, etc.

A note on deployment practice: generally when deploying you'd want to move the "public" directory to your web server to be internet accessible, but keep the non-public directories somewhere below the web root. Since the public index.php ends up only making a simple reference to the Bootstrap file served by including it, this means most of the application files will not be accessible by internet users. It also means that the index.php file will end up deciding where the Bootstrap file is located - so it's the one public file that always needs to be edited by hand for deployment to change that location as needed.

The "tests" directory usually holds any application specific unit, integration and acceptance tests. As I noted previously I'm deliberately taking a test-light approach to stay on topic (I have more than one testing/development article on Zend Devzone if interested in applying TDD or BDD). Main tests I expect to have are for application specific classes: any Model logic for example.

At the end of this step you should have the directory structure in place containing some empty files as follows (create them now, and we'll fill them in later):

```
/application/Bootstrap.php
```

```
/application/controller/IndexController.php
/application/views/scripts/index/index.phtml
/public/index.php
/public/.htaccess
```

Step 2: Implementing Our Bootstrap File

Bootstrapping is when we setup the initial environment, configuration and Zend Framework initialisation for our application. It's not a difficult file to write, but it can grow ever larger and complex as your application gains features. To manage this I strongly suggest implementing it as a class with bitesize methods. Breaking it up does wonders for your sanity. For our Hello World skeleton application, `/application/Bootstrap.php` contains:

```
<?php
require_once 'Zend/Loader.php';
class Bootstrap
{
    public static $frontController = null;

    public static function run()
    {
        self::setupEnvironment();
        self::prepare();
        $response = self::$frontController->dispatch();
        self::sendResponse($response);
    }

    public static function setupEnvironment()
    {
        error_reporting(E_ALL|E_STRICT);
        ini_set('display_errors', true);
        date_default_timezone_set('Europe/London');
    }

    public static function prepare()
    {
        self::setupFrontController();
        self::setupView();
    }

    public static function setupFrontController()
    {
        Zend_Loader::registerAutoload();
        self::$frontController = Zend_Controller_Front::getInstance();
        self::$frontController->throwExceptions(true);
        self::$frontController->returnResponse(true);
        self::$frontController->setControllerDirectory(
            dirname(__FILE__) . '/controllers'
        );
    }

    public static function setupView()
```

```

{
    $view = new Zend_View;
    $view->setEncoding('UTF-8');
    $viewRenderer = new Zend_Controller_Action_Helper_ViewRenderer($view);
    Zend_Controller_Action_HelperBroker::addHelper($viewRenderer);
}

public static function sendResponse(Zend_Controller_Response_Http $response)
{
    $response->setHeader('Content-Type', 'text/html; charset=UTF-8', true);
    $response->sendResponse();
}
}

```

Note: Due to some Serendipity bug in processing Geshi output, underline tags replace some underline characters such as for the PHP FILE constant which should have two underscores preceding and following "FILE". Sorry about that - S9Y has it's moments.

The Bootstrap class has two main methods here: run() and prepare(). run() executes a full application request, whereas prepare() only sets up (but doesn't execute) the Front Controller. For now let's focus on run() and all the steps our Bootstrap takes when it's called. The prepare() method is something I use when applying TDD or BDD to developing controllers, since I then delegate controller execution to the testing framework.

The first thing is to setup the local environment. So here we've set a timezone (as required by PHP5), enabled PHP's display_errors option, and set a fairly stiff error reporting level. If putting these basics into a class seems counter intuitive, you can simply extract them to the file head. Because I usually test extensively, I usually don't want these running during my tests and interfering with test results.

Secondly we prepare the Front Controller. Zend_Controller_Front is an implementation of the Front Controller Design Pattern, which acts as a single point of entry into a Model-View-Controller architected application. I kid you not - every request which is not for an existing public file will go through it. In our case, since the Bootstrap is included into, and called by public/index.php, "index.php" is our single point of entry.

Here, the prepare() stage involves setting up the Front Controller:

- get an instance of Zend_Controller_Front
- set it to throw all Exceptions (might want to disable this in production)
- set it to return a Response object when dispatched
- tell it where to find the default controllers

The last step of prepare() makes some changes to the View. The reason I've done this is to show how to make an extremely common change - adding support for UTF-8 encoded output (such as my name!). Don't worry too much about the details now - the ViewRenderer Action Helper and the Helper Broker lets one add or modify Action Helpers and is well covered by Matthew Weier O'Phinney over on Zend Devzone - <http://devzone.zend.com/article/3350-Action-Helpers-in-Zend-Framework>.

Now, what about /public/index.php?

<?php

// Update after deployment for location of non-public files

\$root = dirname(dirname(<u>_FILE_</u>));

// We're assuming the Zend Framework is already on the include_path

```

set_include_path(
    $root . '/application' . PATH_SEPARATOR
    . $root . '/library' . PATH_SEPARATOR
    . get_include_path()
);
require_once 'Bootstrap.php';
Bootstrap::run();

```

Simplicity itself! The index is our Front Controller's single point of entry into a Zend Framework application, and all it needs to do is determine a valid `include_path` for the application and then delegate all other tasks to the Bootstrap class's `run()` method.

The final step is how to make all requests to "http://zfblog/" pass through `index.php`. This is solved for Apache web servers using its Rewrite Module, which simply takes an incoming request URL, and rewrites it so it points to `index.php`. If you've never used Apache Rewrite before, you probably need to enable this module in `http.conf` before this works. Rewrite rules can be defined in Apache's `http.conf` file, or simply via a `.htaccess` file in "public" containing the rules:

```

RewriteEngine on
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule .&lowast; index.php

```

The `.htaccess` file here first turns on Rewriting, and then specifies rules. The rules here map all requests to `index.php`, unless they relate to a file which exists on the server (i.e. it won't rewrite URLs to CSS, Javascript or image files).

Step 4: Implementing The Index Controller

Before we continue, a quick word on controller/view wiring. Some time back it was decided to make automated rendering the default mode of operation for the Zend Framework. This means Controller will rarely need manual intervention to render a View - instead an Action Helper called the ViewRenderer (which should sound familiar since it used it to add UTF-8 encoding in our Bootstrap!) is called upon. This helper locates a View matching the name of the current Controller and Action and automatically renders it. There are ways to disable this automated rendering as documented in the manual - which is often useful when you want to bypass `Zend_View` (e.g. maybe you're outputting finalised JSON or XML and no further templating or processing is needed).

If a URL to a Zend Framework application does not contain a controller or action reference, then the Zend Framework uses the default "index" value. Since we only intend requesting to the root URL of "http://zfblog/", we'll need an Index Controller containing an Index Action. Let's create one!

Add a file called "IndexController.php" in `application/controllers` containing:

```

<?php
class IndexController extends Zend_Controller_Action
{

    public function indexAction()
    {
        $this->view->title = 'Hello, World!';
    }
}

```

```
}
```

All controllers must extend `Zend_Controller_Action`, since it contains all later referenced internal methods and properties common to all Controllers. If you need to tweak the basic Controller to add new methods or properties you may do so using a subclass of `Zend_Controller_Action` and then making all application Controllers extend this subclass instead. Be very careful however in subclassing this way - it's useful for adding new properties to the class, but additional methods are often better added as discrete Action Helpers.

Our new Action method ignores rendering; it's done automatically as discussed earlier. All we need to do is set View data we need. You can add almost anything to a View, since it just becomes a property in a View template. So feel free to add Objects and Arrays. If using Smarty or PHPTAL, this might be different.

Above we've told the View that templates might refer to a variable "title" and given it a value of "Hello, World!". Once the Action method is done, the ViewRenderer will kick in and try to render a template located at `"/path/to/application/views/scripts/index/index.phtml"`.

And what use is a Controller, without a View to display?

Step 5: Implementing The View For `IndexController::indexAction()`

By default, the Zend Framework or more specifically the ViewRenderer Action Helper, will attempt to render a View template for all Controller Actions using a simple convention. For example, the template for `IndexController::indexAction()` should be located at `"/index/index.phtml"` within the "scripts" subdirectory of "views".

The `Zend_View` component is responsible for rendering all View template, often with a little assistance (e.g. ViewRenderer or `Zend_Layout`) from the Controller. A `Zend_View` template is HTML (or some other output type) interspersed with PHP and it doesn't use a separate tag language such as what the Smarty template engine, for example, uses. Here's something to put into your `index.phtml` file for our first Zend Framework application:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <meta name="language" content="en" />
  <title><?php echo $this->escape($this->title) ?></title>
</head>
<body>
  <p>Hello, World!</p>
</body>
</html>
```


As you can see, it's almost pure HTML with a single PHP inclusion for the title text (which we set from our `indexAction()` method earlier!). To ensure security against Cross Site Scripting (XSS) we should escape all output from within the application, and `Zend_View` uses the PHP `htmlspecialchars()` function from it's `Zend_View::escape()` method. Another noteworthy mention is that, since we setup our View to use UTF-8 encoding, this encoding is also passed into `htmlspecialchars()`.

If using `escape()` everywhere sounds bad - well, it is, and it isn't. It's a pain since it's something that would obviously benefit from automation - so not applying escaping is the exception by default. On the other hand it must be done at all times so if you ever see a variable being echoed that's not surrounded by the `escape()` method you should proceed with care.

Step 6: Did it work?

Go ahead and open up your browser. The base url of `http://zfblog/` should now do its thing and return a HTML document containing "Hello, World!".

Conclusion

Getting a simple example up and running has already provided a few insights. But don't let your guard down just yet. Like any battle plan, it will not survive first contact with the enemy (Maugrim's Marvellous Blog is so evil 

). As we proceed, there will more than likely be a few flaws to the simple approach, our Bootstrap will need to expand yet remain maintainable, and we haven't even looked at `Zend_Db` yet!


It's a start. If you're lucky, it may be many starts since you can reuse a similar basic skeleton to skip all those setup tasks for other Zend Framework projects.

Note: The source code for this entry is available to browse, or checkout with subversion, from <http://svn.astrumfutura.org/zfblog/tags/Part3>. The full source code for the entire application (as it exists thus far) from <http://svn.astrumfutura.org/zfblog>.

Continue to: [Part 4: Setting the Design Stage with Blueprint CSS Framework and Zend Layout](#)

Posted by Pádraic Brady in PHP General, PHP Security, Zend Framework at 00:42

Fantastic article thanks mate - it covers a lot, but, thanks to the tone of voice and the article's structure, I was able to digest it all.

I don't have any questions this time, which is surely a good sign! 

Just wanted to say thanks! Anonymous on Apr 29 2008, 00:09

your eclipse snapshot appears to be missing.

apart from that - very detailed and easy to read. I don't think I'll ever be a fan of MVC (it's just not for me), but it's good to read detailed tutorials and come away with ideas. Anonymous on Apr 29 2008, 00:33

Yeah - it's missing. Blooper on my end I'll fix soon 

. I think it got lost when editing from a raw copy! Anonymous on Apr 29 2008, 00:41

One quick question: why are you using a view variable for the page title, instead of the `headTitle()` view helper? Was it simply to simplify this particular tutorial, or did you have another reason? Anonymous on Apr 29 2008, 01:03

Nice article, thanks.

I guess in `/public/index` it should read something like:

```
""
```

```
$root = realpath(dirname(FILE));
```

""

instead of

""

```
$root = dirname(dirname(FILE));
```

"" Anonymous on Apr 29 2008, 03:34

The doubled `dirname()` usage should also work...in most cases. Of course if readers have their own preferred way of doing things there's lots of room to expand beyond this series! Anonymous on Apr 29 2008, 03:39

@Matthew: It's because the view helpers come into play heavily when applying layouts that need variable values from another template out of their scope. For now, this is the simplest possible example but once I need a layout View Helpers will get a lot of attention. Anonymous on Apr 29 2008, 03:42


With all those functions, surely it'd be easier to "de-static" the Bootstrap class?

Then again, maybe `self::` looks better than `$this->`?

Regards,

Rob... Anonymous on Apr 29 2008, 04:57

Or maybe I have a specific platform, that requires fine grained access to Bootstrap functionality as an abstract API to executing requests via the ZF?

Maybe it does look cuter though 

. Again, this is a Hello World example - such concerns are for another day, as the application grows and evolves. Anonymous on Apr 29 2008, 05:11


Good stuff, I like the bootstrapper example. BTW, I read the new PHP mag, great interview 

Anonymous on Apr 29 2008, 06:09

I an absolute newbie to ZF, so I'm really interested in where this is going.

For now using a bootstrap class and "" makes this tutorial look a lot more professional than all others I've read so far.

In those you usually have stuff like "" which always makes me shiver...

I'm really hoping to see some form handling in one of the next episodes 

Anonymous on Apr 29 2008, 07:24

Regarding the so-called serendipity bug, have you tried disabling the default `s9y`-formatting and just using `geshi` (and probably `nl2br`)? This article doesn't look like you need the `s9y` format plugin anyway. Anonymous on Apr 29 2008, 19:17

Hi,

Excellent article, can I use `index.php/bootstrap` to pass certain request to Zend Controller and others avoid.

For example `www.mydomain.com/controller/method ...` I handle via my own app controllers.

But for `www.mydomain.com/affiliate/index.php`, I want to avoid passing through the controller.

How to achieve this?

Many Thanks,

Santosh Anonymous on Apr 29 2008, 19:25

I've tried disabling it yes - but the formatting is still applied. I'm going to have to hunt down where it's applied and disable it manually I think. It's just such a small thing at the end of the day I haven't worked up the enthusiasm to go do it.

What can I say, programmers create frameworks because we're innately opposed to avoidable work



. Anonymous on Apr 29 2008,

19:26

If you're pointing to a specific file - and the URL ending does contain the exact file relative path with file extension - then in theory the .htaccess rule I use should be sufficient. The other possible option is assembling the avoidance rules as an extra Bootstrap method which hands off to a simple include() statement to bring in the file you want. Anonymous on Apr 29 2008, 19:29

is not possible to download a zip with the source code? would be very helpfull Anonymous on Apr 30 2008, 05:12

I'll look into the possibility tomorrow - fyi all code isn't explicitly licensed yet so be aware I'll be licensing it all under the New BSD License (feel free to rob it blind in other words). Anonymous on Apr 30 2008, 05:16

Don't subclass the Action controller! That's what Action Helpers are for. Anonymous on Apr 30 2008, 17:20

@mak: If you mean subclassing for a new controller you're plain wrong - how else do you get a new Controller? If you mean having a stock subclass which all controllers will extend, then I have done no such thing. I always use Action Helpers or Front Controller parameters instead. Anonymous on Apr 30 2008, 19:24

This is a very nice example for a newbie like me...

i have gone through one more example before reading this...

i like to mention that, if you are having an error "< undefined symbol", which is caused by `dirname(FILE)` in `index.php` and `Bootstrap.php`,

Then...

U can replace the function `dirname(FILE)` by `dirname(_FILE_)`

Thanks for this Example... Anonymous on Nov 17 2008, 22:09

You're welcome - unfortunately that annoying error is unavoidable right now given how serendipity works on my system. Anonymous on Nov 18 2008, 01:33

hi...

use `_FILE_`

double underscore before and after FILE in `dirname()` function to remove the "< undefined" error...

i posted same thing before, but it changed to `FILE` rather than `_FILE_` (here double underscore before and after FILE, you might viewing only 1 underscore, use 2 underscore) Anonymous on Nov 18 2008, 13:26

Hello,

Very nice tutorial. But I have a question.

I'm not sure I understand why you dont recommend putting the framework in "library" folder. If you have to separate projects arent they using the same folder? Isn't the index file that routes the request. I think it is handled like this in CodeIgniter. Is there a difference ZF and CI handles the `index.php`? Anonymous on Jan 20 2009, 00:42

Hi there,

Fantastic series of tutorials. I have a question regarding the integration of Zend_Amf with the bootstrap file described here.

The main problem I've encountered is that the Amf server echos a line of text before the index view, causing PHP to get a bit cranky when it comes to outputting the index template.

Do you have any tips on how to run an Amf server along with a standard HTML view?

Cheers,

Chris Anonymous on Feb 3 2009, 12:35

To prevent some warnings in the actual version of ZF replace the following lines in the Bootstrap-File

```
require_once 'Zend/Loader.php';  
Zend_Loader::registerAutoload();
```

with `require_once 'Zend/Loader/Autoloader.php';` and `Zend_Loader_Autoloader::getInstance();`; Anonymous on Jul 9 2009, 07:32

Thank you, Thank you, Thanks you

I read "Zend framework in action" and "Zend quick start tutorial" and many many Articles to know the details of MVC@ZF but this is the best article to starting with ZF

many many Thank again Anonymous on Sep 2 2009, 20:11