


Thursday, May 1. 2008


An Example Zend Framework Blog Application - Part 5: Creating Models with Zend_Db and adding an Administration Module

Today's entry adds a few more layers to our slowly growing blog application. First of all I decided to add an Entries Model and Authors Model to the mix, primarily to get ready for when we can add new entries to our blog. This leads to where we can create new Entries; we add an Administration Module to the application with it's own distinct Layout.

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

At the end of Part 5, we'll have Models in place and an ugly looking Administration Panel with exactly one reference to an administrative function. The ugliness is unavoidable for now 

. We'll have to start working on `/public/css/style.css` to make some custom changes to the default Blueprint CSS styling soon.

I kid you not, I'm installing the finished application on this subdomain at the end of the series so it better look good by then 

Step 1: Creating a Database Schema

The schema for our blog's database will be aimed at MySQL. We're only starting with two tables to hold entries and authors, which may appear a little suspicious. The suspicion of course is due to the lack of two elements: Authentication and Authorisation. We'll cover both in Part 6 in the near future.

Here's the tables I came up with for now - ensure you have a local database set up, perhaps using phpMyAdmin, so you can just throw in this SQL to create the tables.

```
CREATE TABLE `entries` (  
  `id` int(11) NOT NULL auto_increment,  
  `title` varchar(200) collate utf8_unicode_ci NOT NULL,  
  `date` timestamp NOT NULL default '0000-00-00 00:00:00',  
  `author` varchar(60) collate utf8_unicode_ci NOT NULL default 'anonymous',  
  `author_id` int(11) NOT NULL default '0',  
  `body` text collate utf8_unicode_ci NOT NULL,  
  `extended_body` text collate utf8_unicode_ci NOT NULL,  
  `comment_count` int(4) NOT NULL default '0',  
  `last_modified` timestamp NOT NULL default CURRENT_TIMESTAMP on update  
  CURRENT_TIMESTAMP,  
  PRIMARY KEY (`id`),  
  FULLTEXT KEY `title` (`title`),  
  FULLTEXT KEY `body` (`body`),  
  FULLTEXT KEY `extended_body` (`extended_body`)  
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
```

```
CREATE TABLE `authors` (  
  `id` int(11) NOT NULL auto_increment,  
  `realname` varchar(255) collate utf8_unicode_ci NOT NULL,
```

```
`username` varchar(20) collate utf8_unicode_ci NOT NULL,  
`password` varchar(64) collate utf8_unicode_ci NOT NULL,  
`email` varchar(128) collate utf8_unicode_ci NOT NULL,  
PRIMARY KEY (`id`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
```

We'll likely only use the bare minimum of fields to start with. There are others such as `comment_count` in `entries` we can't use yet since we don't have a commenting system in place, and likely more just not included above yet. Obviously the input page for entries can be anything from a simple set of form fields, to a three page monstrosity!

If you're curious about why we're using MyISAM for the entries table instead of InnoDB as for authors, the simple reason is that full-text indices are only available on MyISAM tables to enable search functionality using the `MATCH()` function in MySQL. We've added full-text indices for the title, body, and `extended_body` fields.

Step 2: Adding the Models

To interact with our newly created database tables, we're going to need some Models. Back in our previous excursion into explaining Model-View-Controller (MVC) we described that a Model is basically a representation of application state - i.e. data persisted between requests often in a database. The Model for a database in the Zend_Framework is usually captured as a Class inheriting from `Zend_Db_Table`.

The Model for our `entries` table will be maintained in an `Entries` class in `/application/models/Entries.php`:


```
<?php  
class Entries extends Zend_Db_Table  
{  
    protected $_name = 'entries';  
}
```

Similarly we will have `/application/models/Authors.php`:

```
<?php  
class Authors extends Zend_Db_Table  
{  
    protected $_name = 'authors';  
}
```

It really is that simple for now. `Zend_Db_Table` even assumes the primary key is "id" unless otherwise defined. If you really want to know what the fuss about Models is, it's all in the methods you add to this class. For now, it's a blank slate referencing the database's table name. But you can imagine a few business logic rules you could consider adding here (hint: processing of entry bodies before saving using `HTMLPurifier`). The main idea to keep in mind is that if you're doing something to data from a Model, and it's potentially useable in more than one Controller, then chances are you're better off adding it as a Model method. Why make your Controllers fat and bloated? The more bloat they contain, the harder it becomes to see the application's workflow by reading them!

Step 3: Adding Database Credentials and a Default Database Adapter for Zend_Db

We have our database schema in place along with Models to represent it. Time to do something so our Model can actually interact with the database then 


The first step is storing our database credentials in an editable file. Create a new file called `config.ini` in `/config` containing something along the lines of the following (edit for your personal credentials and database name):

```
[general]

;Database connection settings
db.adapter=PDO_MYSQL
db.host=localhost
db.username=root
db.password=passwd
db.dbname=zfblog
```

Note: The Subversion repository contains a template of the above file called `config.ini.example`. This way I can change my own `config.ini` file (which is on the subversion ignore list for its parent directory) without committing its changes to the repository all the time! You will need to manually create a copy if running from Subversion.

By now you probably know the drill, and you're wondering just how far I mangled `/application/Bootstrap.php` to integrate database setup...

Not too much really. In the below revised Bootstrap file I've shuffled a few things around, added new methods for setting up a Registry, Config file and Database connection. Hopefully it's self explanatory by now... To answer a question raised in the comments previously, I'm using a static class simply because the Bootstrap isn't really responsible for a whole lot other than initialisation and execution - most tutorials fall so far back in time they even use procedural style PHP! 

Static methods are also attractive because I do rely on not needing a discrete instance for other reasons - which we're not covering here since I'm not obsessing about Behaviour-Driven Development or TDD in this series.

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

    public static function run()
    {
        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');
self::$root = dirname(dirname(<u>_FILE_</u>));
}

public static function prepare()
{
    self::setupEnvironment();
    Zend_Loader::registerAutoload();
    self::setupRegistry();
    self::setupConfiguration();
    self::setupFrontController();
    self::setupView();
    self::setupDatabase();
}

public static function setupFrontController()
{
    self::$frontController = Zend_Controller_Front::getInstance();
    self::$frontController->throwExceptions(true);
    self::$frontController->returnResponse(true);
    self::$frontController->setControllerDirectory(
        self::$root . '/application/controllers'
    );
    self::$frontController->setParam('registry', self::$registry);
}

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($viewRenderer);
    Zend_Layout::startMvc(
        array(
            'layoutPath' => self::$root . '/application/views/layouts',
            'layout' => 'common'
        )
    );
}

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

public static function setupRegistry()
{
    self::$registry = new Zend_Registry(array(), ArrayObject::ARRAY_AS_PROPS);
    Zend_Registry::setInstance(self::$registry);
}


public static function setupConfiguration()
{

```

```

$config = new Zend_Config_Ini(
    self::$root . '/config/config.ini',
    'general'
);
self::$registry->configuration = $config;
}
public static function setupDatabase()
{
    $config = self::$registry->configuration;
    $db = Zend_Db::factory($config->db->adapter, $config->db->toArray());
    $db->query("SET NAMES 'utf8'");
    self::$registry->database = $db;
    Zend_Db_Table::setDefaultAdapter($db);
}
}
}

```


The `setupDatabase()` method takes the new `Zend_Config` instance storing our configuration data from `config.ini` (and which we're now keeping in the Registry for reference) and sets up a connection by passing the configuration data to `Zend_Db`'s `factory()` static method. The first act is to use the new connection to ensure we're once again being careful to stick with UTF-8 encoding. Have to be careful with my name after all - it has one of those weird European slash thingies over the a (we call it the "fada" in Irish Gaelic, the French call it an "acute", and HTML standards refer to it as "á" - outnumbered two to one ).

The connection object is stored to the Registry in case it's required later. It's usually needed as a last resort if we need it for something a Model isn't well suited for. The Registry is passed to the Front Controller in `setupFrontController()` as a user parameter. Finally, but not least, we make this new connection the default for `Zend_Db_Table` - available to all our Models.

It does look like a complicated web of steps, but honestly the code is pretty small for all this.

Step 4: Adding an Administration Module

We have the database, the Model, and the database connection.

Before we jump further, I'm going to make an assumption. Entry submissions will only be allowed from an Administration Module. Once we do have Authorisation implemented we'll seal off access to any such Administration functions, but for now since the blog remains on our private development PC we can leave it openly accessible - until Part 6 .

The Zend Framework allows for all Controllers and Views to be grouped into Modules relatively easily. Up to now we've been putting everything into their default locations without any thought of Modules, so it's high time we added one for Administration.

The first step is to introduce a new directory to our current directory structure called `admin` inside `/application`. You can also delete the previously suggested `modules` directory - we'll keep the directory tree a little flatter than I usually go with, although I can change this if readers prefer. Inside `admin`, add both a `controllers` and `views` directory. The `views` directory should in turn have `filters`, `helpers`, `layouts`

and scripts directories.

To allow our application divert requests to the new admin module, we also need to register its controllers directory with the Front Controller. This is a quick edit to our Bootstrap file at /application/Bootstrap.php in the setupFrontController() method:

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

    public static function run()
    {
        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');
        self::$root = dirname(dirname(<u>_FILE_</u>));
    }

    public static function prepare()
    {
        self::setupEnvironment();
        Zend_Loader::registerAutoload();
        self::setupRegistry();
        self::setupConfiguration();
        self::setupFrontController();
        self::setupView();
        self::setupDatabase();
    }

    public static function setupFrontController()
    {
        self::$frontController = Zend_Controller_Front::getInstance();
        self::$frontController->throwExceptions(true);
        self::$frontController->returnResponse(true);
        self::$frontController->setControllerDirectory(
            array(
                'default' => self::$root . '/application/controllers',
                'admin' => self::$root . '/application/admin/controllers'
            )
        );
        self::$frontController->setParam('registry', self::$registry);
    }
}
```

```

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($viewRenderer);
    Zend_Layout::startMvc(
        array(
            'layoutPath' => self::$root . '/application/views/layouts',
            'layout' => 'common'
        )
    );
}

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

public static function setupRegistry()
{
    self::$registry = new Zend_Registry(array(), ArrayObject::ARRAY_AS_PROPS);
    Zend_Registry::setInstance(self::$registry);
}

public static function setupConfiguration()
{
    $config = new Zend_Config_Ini(
        self::$root . '/config/config.ini',
        'general'
    );
    self::$registry->configuration = $config;
}

public static function setupDatabase()
{
    $config = self::$registry->configuration;
    $db = Zend_Db::factory($config->db->adapter, $config->db->toArray());
    $db->query("SET NAMES 'utf8'");
    self::$registry->database = $db;
    Zend_Db_Table::setDefaultAdapter($db);
}
}
}

```

Nothing huge has changed. The only odd part perhaps, is that our previous controller directory is now assigned as a "default". This is a special Module key and you should never omit it when setting up Modules since it is the ultimate fallback position for failed requests (after that it's to the ErrorController we'll add in the next Part of this tutorial series).

Our Administration module is going to have two controllers. An Index Controller which will default to displaying a list of available actions (e.g. create new blog entry), and an Entry Controller for adding new Entries (and later editing and deleting them). We'll create a new Index Controller at

/application/admin/controllers/IndexController.php containing the following:

```
<?php
class Admin_IndexController extends Zend_Controller_Action
{
    public function indexAction()
    {
    }
}
```

You'll notice that all Controllers not part of the default Module must have their class name prefixed with the Module name followed by an underscore. This merely ensures we have no annoying name conflicts between Modules over common Controller names. The `indexAction` method is completely empty - on purpose. The controller has nothing to do here except look pretty for a few microseconds until control passes to the ViewRenderer Action Helper to have the correct View rendered.

The most obvious next step, therefore, is adding a template for our new module's index page. Create `index.phtml` at `/application/admin/views/scripts/index/index.phtml` and we'll just add, for now, a simple listing of available functions. All one of them.

The first one we need of course, is a URL for adding a new Blog entry. We'll assume we're going to use an Entry Controller within the Admin Module:

```
<h2>Administration</h2>
<ul>
<li><a href="/admin/entry/add">New Entry</a></li>
</ul>
```

Note: Please do not forget the leading forward slash in relative URLs! That little character ensures all relative URLs remain relative to the base URL and not just get appended to the current URL (which is all prettied up).

Try opening up our budding Administration Panel using the URL `http://zfblog/admin`.

There's one small niggle here, which demonstrates another interesting facet of using `Zend_Layout`. The index page for Administration has that text filled right column. We don't need that - it's destined to hold Blog related stuff for public consumption. Let's make our Admin module utilise a slightly different layout! We'll add a narrower left column this time for future Administration functions.

Step 5: Adding an Administration Module specific Layout

So, add a new template `admin.phtml` to `/application/admin/views/layouts/admin.phtml`. This will be an exact duplicate of `/application/views/layouts/common.phtml` with a few column changes.

```
<!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>
<link rel="stylesheet" href="/css/blueprint/screen.css" type="text/css" media="screen, projection"
>
<link rel="stylesheet" href="/css/style.css" type="text/css" media="screen, projection">
<link rel="stylesheet" href="/css/blueprint/print.css" type="text/css" media="print">
<!--[if IE]><link rel="stylesheet" href="/css/blueprint/ie.css" type="text/css" media="screen,
projection"><![endif]-->
</head>
<body>

<div class="container">
  <div class="block">
    <div id="zfHeader" class="column span-24">
      <h1>Lorem Ipsum</h1>
    </div>
  </div>
  <div class="block">
    <div id="zfExtraLeft" class="column span-5">
      <!-- We'll add the style to style.css later -->
      <div class="zfMenuLeft" style="margin-top: 3em;">
        Lorem ipsum dolor sit amet
        Lorem ipsum dolor sit amet
        Lorem ipsum dolor sit amet
        Lorem ipsum dolor sit amet
      </div>
    </div>

    <div id="zfContent" class="column span-18">

      <?php echo $this->layout()->content ?>
    </div>
  </div>
  <div class="block">
    <div id="zfFooter" class="span-24">
      <p>Copyright &copy; 2008 Pádraic Brady</p>
    </div>
  </div>
</div>
</body>
</html>

```

In order to get the Admin Module using this layout, we need to intercept the `Zend_Layout` object just before an application request is dispatched to any Controller. We can then alter the default configuration we have used in our Bootstrap file. Welcome to the world of the Front Controller Plugin!

A front controller plugin already exists called `Zend_Layout_Controller_Plugin_Layout` which contains a `postDispatch()` method which is where the Layout gets finally rendered. To switch Layouts, we can simply create a new class extending the existing plugin, and add a `preDispatch()` method to detect when the Admin Module has been requested and replace the `Zend_Layout` layout name and the path to the Module's `layouts` directory.

Front Controllers are very useful in this fashion for performing actions which have a dependency on the

Module, Controller or Actions being requested. In Part 6 of this series, we'll use another Front Controller Plugin to implement an Access Control List system for Authorisation using Zend_Acl which uses Module/Controller/Action names to check if the current user is authorised to access them.

To start, create a new directory tree within `/library` called `ZFBlog`. It will use the standard PEAR Convention directory structure for a class called `ZFBlog_Layout_Controller_Plugin_Layout`:

```
/library
  /ZFBlog
    /Layout
      /Controller
        /Plugin
          /Layout.php
```

I suggest keeping an eye on ZFBlog. As you start to develop Zend Framework applications you will likely end up with two distinct types of additional classes. Zend Framework specific extensions and subclasses, and application specific classes. A lot of the time, such quirky additions can be reused in other applications. I've used this Layout plugin more than once



The contents of the `Layout.php` file could be as simple as:

```
<?php
class ZFBlog_Layout_Controller_Plugin_Layout extends Zend_Layout_Controller_Plugin_Layout
{
    public function preDispatch(Zend_Controller_Request_Abstract $request)
    {
        switch ($request->getModuleName())
        {
            case 'admin':
                $this->_moduleChange('admin');
            }
        }
    protected function _moduleChange($moduleName)
    {
        $this->getLayout()->setLayoutPath(
            dirname(dirname(
                $this->getLayout()->getLayoutPath()
            ))
            . DIRECTORY_SEPARATOR . $moduleName . '/views/layouts'
        );
        $this->getLayout()->setLayout($moduleName);
    }
}
```

The above plugin is really simple. Before a request is dispatched we get the Module name from the Request object and check it against our switch statement. If a match is found, we can reset the Zend_Layout layout name and path for that request, or if nothing matches we leave things as they stand.

If you are particularly astute at refactoring, you may feel explicitly mentioning "admin" in the class is a bad move - I'll leave it to readers to search for more flexible solutions handling multiple modules with different layouts.

Now, we have a new layout template and we have a plugin to utilise it when the Admin module is requested. Let's make sure this new class replaces the Zend_Layout_Controller_Plugin_Layout class referenced by default in Zend_Layout. Guess where that gets done?

The Bootstrap! 

Check out the revised setupView() method where we change the default plugin class. Note that since this project is utilising Autoloading, and we're storing the new plugin class in `/library` while also applying the PEAR Naming Convention, we require no other class inclusions and such. It just gets autoloaded when needed.

Note: The Zend Frameworks default autoloading function is only useful for classes and libraries strictly following the PEAR Conventions. I suggest moving non-PEAR Convention libraries and classes to a separate vendor directory parallel to `library`.

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

    public static function run()
    {
        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');
        self::$root = dirname(dirname(__FILE__));
    }

    public static function prepare()
    {
        self::setupEnvironment();
        Zend_Loader::registerAutoload();
        self::setupRegistry();
        self::setupConfiguration();
        self::setupFrontController();
    }
}
```

```

    self::setupView();
    self::setupDatabase();
}

public static function setupFrontController()
{
    self::$frontController = Zend_Controller_Front::getInstance();
    self::$frontController->throwExceptions(true);
    self::$frontController->returnResponse(true);
    self::$frontController->setControllerDirectory(
        array(
            'default' => self::$root . '/application/controllers',
            'admin' => self::$root . '/application/admin/controllers'
        )
    );
    self::$frontController->setParam('registry', self::$registry);
}

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($viewRenderer);
    Zend_Layout::startMvc(
        array(
            'layoutPath' => self::$root . '/application/views/layouts',
            'layout' => 'common',
            'pluginClass' => 'ZFBlog_Layout_Controller_Plugin_Layout'
        )
    );
}

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

public static function setupRegistry()
{
    self::$registry = new Zend_Registry(array(), ArrayObject::ARRAY_AS_PROPS);
    Zend_Registry::setInstance(self::$registry);
}

public static function setupConfiguration()
{
    $config = new Zend_Config_Ini(
        self::$root . '/config/config.ini',
        'general'
    );
    self::$registry->configuration = $config;
}


public static function setupDatabase()
{

```

```
$config = self::$registry->configuration;
$db = Zend_Db::factory($config->db->adapter, $config->db->toArray());
$db->query("SET NAMES 'utf8'");
self::$registry->database = $db;
Zend_Db_Table::setDefaultAdapter($db);
}
}
```

Go ahead and reload <http://zfblog/admin> in a browser. New module, new layout!

Conclusion

In this fifth installment in our series we've visited Models and Modules. By now I'd say people are probably wondering where all the good stuff is  . Patience, Padawan. There is a lot to take in so far, and it only gets worse!

In Part 6, I'll be adding Authentication and Authorisation using Zend_Auth and Zend_Acl. By this point we'll have an Author, and we can spend Part 7 looking into how to save and retrieve blog entries using our Models, Zend_Form, and our pre-prepared Administration Module.


Questions are welcome in the comments - don't hold back!

Continue to: [Part 6: Introduction to Zend_Form and Authentication with Zend_Auth](#)

Note: The source code for this entry is available to browse, or checkout with subversion, from <http://svn.astrumfutura.org/zfblog/tags/Part5>. The full source code for the entire application (as it exists thus far) from <http://svn.astrumfutura.org/zfblog>.

Posted by Pádraic Brady in PHP General, PHP Security, Zend Framework at 15:45

It took a little while to get through all of that! I am following step by step along with this and it has been answering a lot of basic questions I have had with ZF. Such as: What is the best way to add a different layout for an "admin" area? What is a good solid structure for an application? (bootstrap helped here).

Thanks for all the hard work, let me know if you need any help getting that CSS rolling 

Anonymous on May 1 2008, 01:00


I second David Arnold's comment. This is the first example I've been able to find that shows how to use Zend_Layout with a modular directory structure. So, thanks a bunch!

Looking forward to see you tackle authentication and authorization in the next post. Anonymous on May 1 2008, 02:26

Pádraic,


First and foremost, I'd like to thank you for your time and effort to put this tutorial together. There's a lot of folks out there that need a walk through like this, and your outstanding commentary is most appreciated.

I like how you are bringing this along. As one that's been relying on the moduleDirectory structure, I'd like to see that approach. But I'm certainly interested in seeing how you tie everything together, so either approach is fine with me.

As one that has followed your blog for a while, I have always appreciated your insight on ZF. Keep it coming 

Anonymous on May 1


2008, 20:49

Did I mention already in the previous steps that it really helps to figure out how all the small elements of ZF work together? 

I have a question though regarding:

"Note: Please do not forget the leading forward slash in relative URLs! That little character ensures all relative URLs remain relative to the base URL and not just get appended to the current URL (which is all prettied up)."

So using a leading forward slash in ZF-Urls always refers to the base url and not to the document root? No need to use `getBaseUrl()` in all my view scripts? Anonymous on May 1 2008, 21:06

@Jay: I might still switch, but probably not. I expect there to only be the one Admin Module. Thanks for the appreciative words! It's good to know it's slotting into the other tutorials/articles out there in a meaningful way 


@Balu: The forward slash is purely HTML related. In this case, using a virtual host means the base url is identical to the document root - a lot of the time I see folk using subdirectories of htdocs which causes quite a lot of those base href issues a simple virtual host avoids.

In case I over-answered ;), you're right - `getBaseUrl()` won't be required; just use a starting forward slash on all relative urls. Anonymous on May 1 2008, 22:20

When I access (in my case) `localhost/zfblog/admin` I have "404 Not Found" page.

I have download de source-code with the SVN link. And `localhost/zfblog` works perfectly.

Regards from Brasil Anonymous on May 1 2008, 23:06

Using the following ini structure you can simplify (as if it is complicated now 

the `Zend_Db::factory()` call:


```
[general]
;Database connection settings
db.adapter=PDO_MYSQL
db.params.host=localhost
db.params.username=root
db.params.password=passwd
db.params.dbname=zfblog
```

```
$db = Zend_Db::factory($config->db);
```

Is there an option to automatically prefix table names in `Zend_Db` (e.g. to have multiple installations use the same database)?

Anonymous on May 1 2008, 23:19

@PÃdraic: Don't forget all those poor fellows who use shared hosting services. They usually don't have a way to set up their own virtual host.

For those you need to have some base url mechanism. 

Anonymous on May 1 2008, 23:24

I haven't tested the application outside a virtual host setting, so I can't be absolutely sure of why a 404 occurs. A 404 indicates you're not hitting the application at all (I haven't added an ErrorController so there's nothing in the app to throw a 404).

I recommend using a Virtual Host for Apache as described in an earlier Part and seeing if that works. Perhaps you even are using a VH, and you should drop the localhost part of the URL? Final stop is that without a VH, and with the default directory structure - it would actually end up as `http://localhost/public/admin` (I think) assuming the public directory is located in the localhost directory root.

This is what happens when you depart from the tutorial steps when setting up



. Th VH will save you a lot of trouble! Anonymous on

May 1 2008, 23:28

I'm having a problem with the RewriteRules, because I've a different url. Instead of `http://zfblog/`, I put it all in `http://localhost/zfblog/`, and `http://localhost/zfblog/admin/` doesn't work for me. What do I have to change in `.htaccess` to match my url??

Thanks, great tutorial serie. Anonymous on May 1 2008, 23:55

This is going to sound really dumb - but automatic prefixing can be done by giving your models `$name` property



. Just change

"authors" to "zfblog_authors", and edit the schema.

If you need that configured, add a `Zend_Db_Table` subclass to extend all Models from, and use its `init()` method to apply a configured prefix to the value of `$_name`



There is nothing odd or unusual about extending ZF classes - I do it all the time and it works really well for those specialised cases where you need access to the class internals. Anonymous on May 1 2008, 23:55

Two queries so far on this



. That extra `/admin/` partial in the url brings out all the non Virtual Host users.

I'll make a note back in Part 3 about how to work around the lack of a Virtual Host - do you also wish me to add in the automated database table prefixing as an example? Make more shared host users happier probably! Anonymous on May 1 2008, 23:58

I'm using in mysql `httpd.conf`(Apache 2):

```
Alias /zfblog /usr/local/Zend/apache2/htdocs/zfblog/public
```

How can I set a virtual host?

I don't have a file `virtual-hosts.conf` in my "conf" directory. Anonymous on May 2 2008, 00:51

I'd love to see that




. At the moment I'm wildly guessing that you'd extend `Zend_Db_Table` to fetch the prefix from the registry (aka config) and add it to `$_name`...

However the database prefix is probably not that important as the base url might be. Anonymous on May 2 2008, 02:35

Perhaps I should've read all the comments before making my wild guess




Anonymous on May 2 2008, 02:38

You can input directly into http.conf 

. The VH file is in a subdirectory of conf in Apache 2.2 at least. Anonymous on May 2 2008, 02:55

I set the virtual host, but when I access http://zfblog/admin I have the same 404 page..... very strange Anonymous on May 2 2008, 07:57


What you get from http://localhost? This really is very odd. Is there any chance you're getting webservers mixed up? I once did the same thing and got mixed up between having 2 Apache webservers, and IIS on the same machine. I started up one, had my files set for another, and spent something ridiculous trying to figure out why I kept getting 404s 

. Anonymous on May 2 2008, 15:38


Thanks for this great tutorial, this is something that ZF really needs to kick start developers.

Everything is so easy to understand to everybody that tried building an application with ZF, to everybody else I would really recommend to take some time and go through the ZF manual, starting with the Zend_Controller chapter.

I just started building a personal blog with ZF and this series came in the right time to help me clarify some things (kudos for the layout controller plugin).

I started to write this comment to tell you to use DIRECTORY_SEPARATOR constant instead of "/" when referencing directories, but it's not needed, I just created the config directory in the wrong location 

Anonymous on May 2 2008, 21:19

Glad to have you along for the ride 


Anonymous on May 2 2008, 22:09

From http://localhost/ I have the default apache page "It works!".

From http://zfblog/ I have the "Lorem Ipsum" page.

And from http://zfblog/admin/ I have the 404 error page. Anonymous on May 2 2008, 22:24

Is mod_rewrite enabled in http.conf? It's often distributed in Apache as disabled by default. Anonymous on May 2 2008, 22:54

Thanks 

I think it's worth mentioning:

In Bootstrap.php you have

```
$db->query(&quot;SET NAMES 'utf8'&quot;);
```

which forces the application to make a mysql connection and we are not utilizing the lazy connections that ZF provides.

Unfortunately ZF doesn't have an option to pass this kind of "parameter" to the database factory.

I am working on a fix for MYSQL_PDO and MYSQLI Db adapters to send this query while creating the connection if the appropriate 'charset' is passed along with other database options.

Is it really necessary to register the database connection with the registry? I was using a little bit different approach. My Models had an init() method with something like this in them:

```
$this-&gt;db = Zend_Db_Table_Abstract::getDefaultAdapter();
```

And one more question, the code that you provide, is it free for use and customize for our own projects?
Anonymous on May 3 2008, 01:18

Now It's OK!

I set AllowOverride to All in my Apache configuration.

Thanks! Great Tutorial Anonymous on May 3 2008, 01:42

As to the first - I'm not a Zend_Db person, but perhaps a Zend_Db_Table subclass with a static tracking var for when the query "SET NAMES" is called by an init() method? This really should be standard - hope you can get something through!

I use the Registry as a route of last resort for DB access outside of a Model. Rarely used (if ever) if I can help it.

All code is released under the New BSD License. All text is released under a Creative Commons 3.0 License. The 3.0 will apply to more download+read format I'm working on (check svn - the first 2 articles are already formatted as Docbook XML). Anonymous on May 3 2008, 02:39

I have something going on I don't get. When I visit <http://meh/> or <http://meh/index> everything works as expected (I've even implemented authentication and a few other things and they all work).

However, when I go to <http://meh/index.php> I get an exception thrown with the message: invalid controller(error).

Any ideas on what might cause that Mr. Maugrim The Reaper?

BTW, let me echo the praise for your work here (and with PHP in general, for that matter) especially with tying things together via this bootstrap class and module implementation. I'm one very impressed and appreciative Ohioan. Anonymous on May 3 2008, 14:11

The last exception occurs because I haven't added an ErrorController yet. Next Part as an aside to Authentication/Authorisation.

The error itself I'm not sure of - it works for me. Then again, as comments have suggested I do tend to take my setup for granted so ensure you have the usual suspects covered: .htaccess as described, Allow Override set to All in http.conf for Apache's document root or other (if not a child of that path), make sure no other rules messing with the .htaccess file (e.g. other rules buried in http.conf).

Unfortunately, the ZF's most annoying flaw when first starting is that controller exception since it gives you exactly no useful information in diagnosing the problem. Anonymous on May 3 2008, 14:48

I was having problems with the 404 error when pointing to [zfblog/admin](#) as well.

Changing my .htaccess file from:

```
RewriteEngine on
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule .* index.php
```

To:

RewriteEngine on
RewriteRule !\.(js|ico|gif|jpg|png|css)\$ index.php

Fixed the error for me. Anonymous on May 3 2008, 22:30

Hi Anson,


Add directories to the rewrite condition. Try adding:

```
RewriteCond %{REQUEST_FILENAME} !-d
```


and see if that fixes the problem. I dislike the use of

```
RewriteRule !\.(js|ico|gif|jpg|png|css)$ index.php
```


because you constantly have to maintain it if you want to add new files such as a .zip or .tgz file to your application. Anonymous on May 4 2008, 01:00

I'm looking into a few problems like this - seem to be a lot of disparate configs out there 

. I'll more than likely end up with either a single solution, or make note of varying solutions that work depending on the configuration issues. Anonymous on May 4 2008, 07:04

Im excited for the part 6. when can we expect it? 


Anonymous on May 5 2008, 15:03

It will be done...when it's done 

. It's a bank holiday here in Ireland so likely tomorrow. Anonymous on May 5 2008, 19:01

Thanks Pádraic this is a great series. You are covering everything very clearly and I especially like that you repeated the whole bootstrap when you made changes to it. It allows us to see the changes in context.

Thanks,
Nigel Anonymous on May 6 2008, 06:02

Pádraic, thanks for taking the time to post this tutorial. I hope that a publisher contacts you asking if you'd like to write a book on the ZF 

I'm having a problem with the admin controller and the stack trace isn't very helpful.

```
Fatal error: Uncaught exception 'Zend_Controller_Dispatcher_Exception' with message 'Invalid controller specified (error)' in /home/username/public/zf.blog/library/Zend/Controller/Dispatcher/Standard.php:249
```

The main index and database setup are both fine. The big difference in my development environment is that I'm working from my user web directory rather than root. I had to include a RewriteBase value in my Rewrite rule.


```
RewriteBase /~username/zf.blog/public
```

Ideas? Anonymous on May 7 2008, 00:46

@gu Add a RewriteBase to your htaccess.

```
# Apache mod_rewrite rules
RewriteEngine on
RewriteBase /zfblog
RewriteRule !\.(\.js|ico|gif|jpg|png|css)$ index.php Anonymous on May 7 2008, 01:46
```


Whoops. The RewriteBase should be /zfblog/public Anonymous on May 7 2008, 01:58

We learn from our mistakes, or at least I hope we do 

I turned on throwExceptions for the frontController which revealed the culprit. I hadn't put my admin index view template into a scripts directory.

Mea Culpa Anonymous on May 7 2008, 02:09

Next part has a small section towards the end to add a quick ErrorController - probably should have done it earlier to avoid those anonymous controller errors about it being missing.

Me and publishers are having a holiday for now - I was discussing a book on a topic with one but recent events in my life simply turned my focus elsewhere. Maybe another day 

RewriteBase needs adding to Part 3...hmm Anonymous on May 7 2008, 03:31

I look forward to the next part. Since you asked for them, here are a few more comments.

1. Show how to set and use a global variable for use in all views, \$webroot. This variable would precede all links and references in templates, including javascripts, style sheets, images, and links.
2. I had seen useful logic to switch configuration settings between development and production environments. I think the example was in one of the ZF screencasts. Anonymous on May 7 2008, 06:39

Good idea on turning the throw exceptions on, I did that and got a slightly more useful error:

```
Uncaught exception 'Zend_Controller_Action_Exception' with message 'Action "index" does not exist and was not trapped in __call()'
in /Applications/MAMP/bin/php5/lib/php/Zend/Controller/Action.php:477 Stack trace: #0
/Applications/MAMP/bin/php5/lib/php/Zend/Controller/Action.php(504): Zend_Controller_Action->__call('indexAction', Array) #1
/Applications/MAMP/bin/php5/lib/php/Zend/Controller/Dispatcher/Standard.php(293): Zend_Controller_Action->dispatch('indexAction')
#2 /Applications/MAMP/bin/php5/lib/php/Zend/Controller/Front.php(914):
Zend_Controller_Dispatcher_Standard->dispatch(Object(Zend_Controller_Request_Http), Object(Zend_Controller_Response_Http))
#3 /Applications/MAMP/htdocs/zfblog/application/Bootstrap.php(17): Zend_Controller_Front->dispatch() #4
/Applications/MAMP/htdocs/zfblog/public/index.php(12): Bootstrap::run() #5 {main} thrown in
/Applications/MAMP/bin/php5/lib/php/Zend/Controller/Action.php on line 477
```

perhaps you can make some sense of it?

It would seem to be a problem with my directory structure, but it's all triple checked against the setup described.

Any help would be appreciated. Anonymous on May 8 2008, 03:28

What is it about committing something irreversibly to an internet discussion that makes you instantly see what you're silly problem was?

I capitalized indexController.

(-_-)

thanks anyway, and really awesome tutorial! Anonymous on May 8 2008, 03:34

Your embarrassment is noted, Padawan :p Anonymous on May 8 2008, 04:46

This is definitely a better way to init Zend_DB.

The tutorial version does not work if you use Postgres for your database, as it passes "adapter" as a parameter in \$config->db->toArray(). This then throws an exception. I think it is the actual pdo_pgsql that complains and not Zend. However as no one has mentioned it here I assume it works fine when using mysql. I guess the Mysql backend ignores invalid parameters where as Postgres throws an exception. Anonymous on May 10 2008, 17:19

Great tutorial. I'm just now catching up, but had to stop and give props. It's very easy to understand, keep it up! Anonymous on May 13 2008, 05:02

This is really good idea to write this tutorial, thank you for your work PÃ¡draic!

I wonder why did you save Zend_Registry object in Zend_Controller_Front parameters?

```
self::$frontController->setParam('registry', self::$registry);
```

 Anonymous on May 13 2008, 23:44

You're welcome



It's a personal preference. If I need a Registry I like to avoid using the singleton API getInstance() in favour of having the object accessible by a getter method on (non-static) on the Controller. Anonymous on May 14 2008, 00:03

FYI:

The line having the following codes must be corrected:

```
. DIRECTORY_SEPARATOR . $moduleName . '/views/layouts'
```

... should have been:

```
. DIRECTORY_SEPARATOR . '/../' . $moduleName . '/views/layouts'
```

... to go up one level in reference to the original path. Anonymous on Nov 8 2008, 15:46

Hello,

Regarding the message I posted prior to this, please disregard it, as I realized that my folder structure is a little different, as it was the one created by Zend Studio for Eclipse when I used its wizard when I created the project, where a default folder is created, like:

```
> zfBlog
> application
> admin
> controllers
> views
> default
> controllers
> views
```

.
.
.
Sorry for that Anonymous on Nov 8 2008, 16:29

Thanks, By using this method, now we can connect more than 1 DB.

Once again Thanks Anonymous on Dec 17 2008, 20:47

Padraic, you can use this too on your config file for the UTF8 stuff on MySQL:

```
db.params.driver_options.1002 = "SET NAMES utf8"
```

;1002 is the constant for PDO::MYSQL_ATTR_INIT_COMMAND. Anonymous on Dec 18 2008, 20:02

Wish I could have done that when writing this



. Don't worry - the revised versions are coming shortly after Christmas. Anonymous on

Dec 19 2008, 23:46

Thank you so much for this blog especially for "Step 5: Adding an Administration Module specific Layout". I wanted to do this for ages but I didn't know how I could do that



now I know. Anonymous on Dec 23 2008, 22:21

Thanks Padraic for putting the time and effort in helping me understand wiring everything together.

One question: If I have multiple databases configured in my config.ini, how would the setupDatabase function look in the Bootstrap? Would I just create multiple db variables and register them all with different registry names?

For example:

```
$db1 = Zend_Db::factory($config->db1->adapter, $config->db1->toArray());  
$db1->query("SET NAMES 'utf8'");  
self::$registry->database1 = $db1;
```

```
$db2 = Zend_Db::factory($config->db2->adapter, $config->db2->toArray());  
$db2->query("SET NAMES 'utf8'");  
self::$registry->database2 = $db2;
```

etc.? Assuming I modify the prefix in my config.ini for each database.

Thanks again for sharing your expertise! Anonymous on Jan 14 2009, 05:46

Nice tutorial. As for the suggested refactoring in the Layout plugin, how about simply using the (sanitized; maybe with a Zend_Filter) return value of \$request->getModuleName() to build the path to the layout template? This would keep the plugin truly independent from the concrete application. Anonymous on Jan 16 2009, 05:44

Hello.

I just want to point some things. I'm working with zf for a while and i have some suggestions.

Its better idea to put all the blog code inside a blog module, even the administration functions of the blog. This is because if you want to add your blog to a more complex app then you need to maintain all the blog code together.

You can create a simple admin controller in the default module and create an admin layout. There is a simple way to group all admin functions logically. You can create an action in the admin controller of the default module to show the admin panel menu. You can store the admin panel menu links in an xml file. Call the admin panel menu action from the layout.
Then /admin will be the admin panel url.

It is not a good idea to select layouts in front controller plugins because the layout selector code is executed in every request. A better idea is to change the layout in the init() method of the admin controllers of every module. In a project with more than 7 modules like mine currently, this small tip is critical.

Personally i create an admin controller for every module that need to export functionalities for the admin panel and then register the admin controller actions in the admin panel menu xml file.

With the same idea i develop a generic user module that is in charge of the login process, the user profile, etc. The module export admin functions to add login domain, login adapters, acl management, user management, etc. Then i use this module in all my projects and my friends use it too.

I believe you can continue this tutorial showing how to abstract functionalities from the blog app for later use, like the user module.

Following this idea i develop modules for user, analytics, content publishing, comments, search, messaging. I use this modules very often and my friends and partners use them to. Anonymous on Jan 22 2009, 04:57

Hello i'm just following this great tutorial to start with Zend Framework, and i want to say thank you!!!

Just a question: i tried to change the layout switch like this and it work very well:

this is my application/modules/admin/controllers/IndexController.php

[geshi lang=php]

Anonymous on Feb 4 2009, 09:07

Thanks for the post.

For me too <http://zfblog/admin> was not working.

I modified .htaccess as per your post and now it's working. Anonymous on Feb 12 2009, 17:43

Hi,

Thanks for the nice zend tutorial/example. Anonymous on Feb 12 2009, 17:45

dude your tutorial is good and nice to read Anonymous on Feb 16 2009, 23:18

I'm getting this error: <http://www.copypastecode.com/codes/view/3884>

My index.php file:

<http://www.copypastecode.com/codes/view/3886>

My Bootstrap file:

<http://www.copypastecode.com/codes/view/3887>

I have absolutely no clue what the problem is. My Zend folder is in `zfblog/library/zendframework/Zend`
My bootstrap file is in `zfblog/application/Bootstrap.php`

my library folder is in zfblog/library

Can anyone help? Anonymous on Feb 23 2009, 03:30

HI,

I was just curious instead of extending Zend_Layout_Controller_Plugin_Layout perhaps we could have used a simple route on the ini file:

```
routes.adminIndex.route = admin
routes.adminIndex.defaults.module = admin
routes.adminIndex.defaults.controller = index
routes.adminIndex.defaults.action = index
```

adding this to the front controllers router instance, you just say that once this link is given, you should forward the request to module 'admin', 'index' controller and 'index' action.

Just as an info. I have done this way instead of extending and it works pretty well. Anonymous on Mar 4 2009, 21:20

hi, thanks for the better tutorial . I have setup and working with this and all worked fine for me till I add the Zend_Config_Ini file to parse the ini file. I am getting following error.

```
Fatal error: Uncaught exception 'Zend_Config_Exception' with message
'parse_ini_file(C:\wamp\www\test\zfBlog\application..\config/config.ini) [function.parse-ini-file]: failed to open stream: No such file or
directory' in C:\wamp\library\Zend\Config\Ini.php:117 Stack trace: #0 C:\wamp\www\test\zfBlog\application\Bootstrap.php(72):
Zend_Config_Ini->__construct('C:\wamp\www\tes...', 'general') #1 C:\wamp\www\test\zfBlog\application\Bootstrap.php(29):
Bootstrap::setupConfiguration() #2 C:\wamp\www\test\zfBlog\application\Bootstrap.php(12): Bootstrap::prepare() #3
C:\wamp\www\test\zfBlog\public\index.php(9): Bootstrap::run() #4 {main} thrown in C:\wamp\library\Zend\Config\Ini.php on line 117
```

I found no problem in path even. It is correctly included and all other classes are instanciating fine. can you please help me to shoot this trouble.

this is my ini file.

```
[general]
; Database connection settings
db.adapter=PDO_MYSQL
db.host=localhost
db.username=root
db.password=test
db.dbname=zfblog Anonymous on Mar 5 2009, 05:45
```

hi,,thanks for the very good tutorial. I am following the above tutorial last 3 days and going well. Every think work for me fine until i faced the following error while I am using the Zend_Config_Ini class to parse the config.ini file. The error is as following.

```
Fatal error: Uncaught exception 'Zend_Config_Exception' with message
'parse_ini_file(C:\wamp\www\test\zfBlog\application..\config/config.ini) [function.parse-ini-file]: failed to open stream: No such file or
directory' in C:\wamp\library\Zend\Config\Ini.php:117 Stack trace: #0 C:\wamp\www\test\zfBlog\application\Bootstrap.php(72):
Zend_Config_Ini->__construct('C:\wamp\www\tes...', 'general') #1 C:\wamp\www\test\zfBlog\application\Bootstrap.php(29):
Bootstrap::setupConfiguration() #2 C:\wamp\www\test\zfBlog\application\Bootstrap.php(12): Bootstrap::prepare() #3
C:\wamp\www\test\zfBlog\public\index.php(9): Bootstrap::run() #4 {main} thrown in C:\wamp\library\Zend\Config\Ini.php on line 117
```

my config file looks like below.

[general]

; Database connection settings

db.adapter=PDO_MYSQL

db.host=localhost

db.username=root

db.password=test

db.dbname=zfblog

please help me to solve this problem.

Thank you!!! Anonymous on Mar 6 2009, 00:53

I had similar problems with windows and not finding files even if a path was specified correctly. I moved over to Ubuntu Linux and have had no problems since. Anonymous on Jul 12 2009, 17:17