


Saturday, May 10. 2008

## Example Zend Framework Blog Application Tutorial - Part 7: Authorisation with Zend\_Acl and Revised Styling

You'd never think a guy could write so much about a blog application but to date after 6 parts we have covered a mass of detail from initial setup of our project's directory structure to Authentication of users. To date the feedback has been overwhelmingly positive to this series and I'm presently collecting comments regarding improvements for later inclusion.

Today's entry concerns authorisation. We previously covered how to authenticate an author to the blog, but we still have nothing ensuring only authenticated authors can access the new Administration Module. This is the domain of Zend\_Acl, an implementation of an Access Control List system which limits access to resources by the roles assigned to a user.

In the final section of this entry, we take a small detour into the world of CSS (which rarely works out for me  ) where I'll apply some small changes to our Layouts and add two new stylesheets. Once these are added, our infant blog application will look slightly more presentable than it's current nakedness.

### Step 1: Understanding Access Control Lists (ACL)

It can be a bit confusing to face off against ACL if you're new to the subject. In essence all ACL does is keep track of resources and roles.

As to what a resource is, it is anything to which access can be allowed or denied. For our blog application, I could decide that the Administration Module is itself one resource. From there I can restrict all access to that entire Module, including all it's Controller classes and Action methods (which are part of that single Resource). Or perhaps I could determine that only one Action method in the whole Module is a specific Resource, bearing in mind that Resources are nestable (i.e. a basket is a Resource, and each egg it holds are also discrete Resources). Since each Resource can be given differing access rules, you can globally prevent non-author users from accessing the Administration Module, but maybe allow some registered users access to specific Actions in that Module as an exception to the global rule.

A lot of the time managing global rules, and then applying exception rules, is how ACL works in practice.

Explaining a Role is even simpler. Any visitor to the application can be assigned a Role which ACL rules may use to define that user's access to Resources. Typically the first Role everyone will receive is "guest". From there you can escalate Roles to offer a visitors a greater degree of access to Resources. Any user can be given multiple Roles even. For example, if an author visits the blog they start with the role of "guest" but after authentication we might grant them the additional role of "author". If Roles dictate specific but limited responsibilities (perhaps there's an "author" and "editor" Roles) you might decide to start tracking roles more elaborately, in a database possibly.

Going a bit further, if our Administration Module is a Resource called "admin" then we can decide that the only Role with access to it will be the "author" Role. Since our user has been authenticated and granted the "author" Role (either post-authentication or permanently recorded on the database), they can access the Administration Module.


Finally is the concept of Privileges. Just because you can access a Resource, does not instantly mean you should have total uncontrolled access to it. You can limit control over a Resource using Privileges. Perhaps an Author can access the Admin Module (represented by an Admin Resource) but we want to deny Authors

the privilege of deleting entries from the database.

## Step 2: A Little Planning Goes A Long Way

Before we leap into the fray like a demented action hero, let's set out exactly what we're aiming for.

Since our blog is a relatively simple application, we really only need two Roles to start with. We'll call these `guest` and `author`. This may change in the future, perhaps we could allow for multiple Authors but one Editor capable of editing all posts. In that case we'd need to pick apart how that's implemented. But for now, two Roles is just fine.

As for Resources, the first is the public facing facade of our application where entries are displayed, logins performed, and comments made. The second is the Administration Module. Again, we could be more elaborate but let's not overcomplicate the application until we're forced to .

. This suggests we only have two Resources: the Default Module and the Administration Module. Remember that the Default Module comprises everything not assigned to a specific Module (like our Admin Module with its own separate tree of Controllers and Views).

The rules falling out from this quick analysis are simple.

1. Guests can access the Default Module
2. Authors can access the Default Module
3. Guests cannot access the Administration Module
4. Authors can access the Administration Module

## Step 3: Storing Rules in a Class

There is no specific backend storage for `Zend_Acl` which since it is a serialisable class can be simply serialised and stored in anything from a database to a file for later consumption. To keep things simple I'll just implement a class defining the relationship between Roles and Resources as described above. You'll note we are not using Privileges, since access to a Resource assumes the accessing Role has all possible Privileges by default.

Start by creating a new file at `/library/ZFBlog/Acl.php`:

```
<?php
class ZFBlog_Acl extends Zend_Acl
{
    public function __construct(Zend_Auth $auth)
    {
        // Add Resources
        // Resource #1: Default Module
        $this->add(new Zend_Acl_Resource('default'));
        // Resource #2: Admin Module
        $this->add(new Zend_Acl_Resource('admin'));
        // Add Roles
        // Role #1: Guest
        $this->addRole(new Zend_Acl_Role('guest'));
        // Role #2: Author (inherits from Guest)
```

```

$this->addRole(new Zend_Acl_Role('author'), 'guest');
// Assign Access Rules
// Rule #1 & #2: Guests can access Default Module (Author inherits this)
$this->allow('guest', 'default');
// Rule #3 & #4: Authors can access Admin Module (Guests denied by default)
$this->allow('author', 'admin');
}
}

```

One confusing point I've seen asked is whether Resources explicitly refer to Modules, Controllers or Actions. They don't - the names used here are pure convention. A resource is a virtual item. Zend\_Acl doesn't know if a Resource is a Module, Controller or Action since that's a decision we make when we check the ACL rules later. What we'll do then is detect what Module a request for, and specifically carry out an ACL check for the Resource created here to refer to that Module, i.e. the connection between a real Module name and an ACL Virtual Resource is determined entirely by us at checking time.

Probably the biggest area of confusion is that its so common to consider Resources as Controllers, and Privileges as Actions, that people don't realise this is pure convention. Someone seeing `$this->allow('author', 'entry', array('create', 'edit', 'delete'))` would interpret that Authors are allowed access to the Entry Controller with privileges sufficient to create, edit or delete entries. That is only the case if, and when, your ACL logic determines this is how the rule is interpreted.

To prove a point, where do Modules fit? They don't. There is no Module parameter for `Zend_Acl::allow()`. You could append ACL interpretive logic where a Resource called "admin|entry" refers to the Entry Controller of an Admin Module which once again emphasises that a Resource is completely virtual. It is only what you interpret it to be and has no preconceived relationship with MVC. For all you care a Resource could be absolutely anything that is accessible only by passing through the ACL checkpoint.

## Step 4: Implementing ACL using a custom Front Controller Plugin

The problem with ACL is that it's one of those always-on checks. Every single request needs to be checked to ensure that the requesting user has a Role which allows them to access the Resource (i.e. the Module, Controller or Action) being requested.

By stringing together the requirements (interacts with request data, operates on all requests, occurs prior to Controller execution) we realise that the best way of accomplishing this is to add a Front Controller plugin implementing the `preDispatch()` method.

Create a new file at `/library/ZFBlog/Controller/Plugin/Acl.php`.

Once again we're using the PEAR Convention and we will continue to mirror the organisation of Zend Framework classes.

```

<?php
class ZFBlog_Controller_Plugin_Acl extends Zend_Controller_Plugin_Abstract
{
    protected $_auth = null;
    protected $_acl = null;
    public function __construct(Zend_Auth $auth, Zend_Acl $acl)
    {
        $this->_auth = $auth;
    }
}

```



allows them access to that Resource. If it does, we simply do nothing and let control pass back to the Front Controller. If access is denied we have two branches - authenticated users are just kicked back to the index page (part of our default Module) while unauthenticated users are redirected to the login page for Authors.

Note: Setting the Module name on the request object for forwarding is required when using Modules. That includes setting references to non-moduled controllers/actions with the Module name of "default".

Of course there are more complex scenarios again. What if the resource is not a Module, but a specific Action on a specific Controller within a Module? Obviously such a simple plugin as above would fall flat and need to be scraped off the floor. Again this is not dictated to you by the Zend Framework manual. It's usually typical to make use of Privileges in Resources and interpret these as Actions. And as explained earlier you can use a "admin|entry" convention for the Module/Controller pairing.

## Step 5: Initialising the Front Controller Plugin

Before our plugin is even used, we need to register it with the Front Controller. This is another change to our Bootstrap class! The new method `setupAcl` is at the bottom of the file.

```
<?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();
        self::setupAcl();
    }
    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);
}
public static function setupAcl()
{
    $auth = Zend_Auth::getInstance();

```

```

$acl = new ZFBlog_Acl($auth);
self::$frontController->setParam('auth', $auth);
self::$frontController->setParam('acl', $acl);
self::$frontController->registerPlugin(
    new ZFBlog_Controller_Plugin_Acl($auth, $acl)
);
}
}
}

```

In case they are needed for a more specific use case in a controller, I've added both the authentication and authorisation objects as Front Controller parameters.

Go boot up `http://zfblog/admin` for a test drive. If you have previously logged in, you can logout using `http://zfblog/author/logout` manually (it's not part of our View yet).

## Step 6: And Now For Something Completely Different...

With ACL implemented I find myself at a loose end. So I spent some time putting together two new stylesheets to add some colour and life to this application. You'll need two new files:

```

/public/css/style.css
/public/css/ie.css

```

The first may already exist from an earlier Part of this series.

Edit `style.css` to contain the following CSS. I won't explain it - this, thankfully, is not a CSS blog 

```

body {
    margin: ;
}
#content {
    min-height: 50em;
}
#header {
    background: #303030;
}
#header h1 {
    float: left;
    width: 235px;
    height: 80px;
    margin: ;
}
#header a {
    color: #AAA;
    text-decoration: none;
}
#footer {
    clear: both;
    width: 100%;
}

```

```

margin: ;
padding: 15px ;
border-top: 1px solid #000;
background: #303030;
text-align: center;
color: #AAA;
}
fieldset {
float: left;
clear: both;
width: 100%;
margin: 1.5em ;
padding: ;
border: 1px solid #BFBAB0;
background-color: #F2EFE9;
}
fieldset.submit {
float: none;
width: auto;
border-style: none;
padding-left: 13.5em;
background-color: transparent;
}
legend {
margin-left: 1em;
padding: ;
color: #000;
font-weight: bold;
}
fieldset ol {
padding: 1em 1em 1em;
list-style: none;
}
fieldset li {
float: left;
clear: left;
width: 100%;
padding-bottom: 1em;
}
label {
float: left;
width: 10em;
margin-right: 1em;
text-align: right;
}
label strong {
display: block;
color: #C00;
font-size: 85%;
font-weight: normal;
text-transform: uppercase;
}
label em {

```

```

display: block;
color: #060;
font-size: 85%;
font-style: normal;
text-transform: uppercase;
}

```

Now edit `ie.css`.

```

#content {
  height:auto !important;
  height:50em;
}
legend {
  position: relative;
  top: 7px;
}
fieldset {
  margin-top: 2em;
  margin-bottom: ;
  position: relative;
}
fieldset ol {
  padding: ;
}
fieldset.submit {
  margin-bottom: 1.5em;
  padding-left: 13.2em;
}

```

Now repeat after me: "Paddy is not a designer!". I can't say this is the most spectacular CSS ever written when it's probably more in the opposite direction. But it works. I'm pretty sure it does anyway. Fingers crossed! There might be some IE7 issues because my conditional doesn't specify a version so check back with subversion in a while if so.

We're not done yet. We need to make two more edits to each of our previous Layout templates. Here's the amended excerpt for each. It is identical for both `/application/views/layouts/common.phtml` and `/application/admin/views/layouts/admin.phtml`. Now since it's identical we know we have some duplication in our layouts, which is undesirable, but we'll fix that another day.

```

<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">
  <link rel="stylesheet" href="/css/ie.css" type="text/css" media="screen, projection">

```

```

<![endif]-->
</head>
<body>

<div class="container">
  <div class="block">
    <div id="header" class="column span-24">
      <h1><a href="/">Lorem Ipsum</a></h1>
    </div>
  </div>

// .....
  <div class="block">
    <div id="footer" class="span-24">
      <p>Copyright &copy; 2008 PÃdraic Brady</p>
    </div>
  </div>
</div>
</body>
</html>

```

Just make matters worse, I added some small changes to our previous AuthorController and it's related login template.

```

/application/controllers/AuthorController.php
<?php
class AuthorController extends Zend_Controller_Action
{
  public function loginAction()
  {
    $form = new ZFBlog_Form_AuthorLogin;
    if (!$this->getRequest()->isPost()) {
      $this->view->loginForm = $form;
      return;
    } elseif (!$form->isValid($_POST)) {
      $this->view->failedValidation = true;
      $this->view->loginForm = $form;
      return;
    }
    $values = $form->getValues();
    // Setup DbTable adapter
    $adapter = new Zend_Auth_Adapter_DbTable(
      Zend_Db_Table::getDefaultAdapter() // set earlier in Bootstrap
    );
    $adapter->setTableName('authors');
    $adapter->setIdentityColumn('username');
    $adapter->setCredentialColumn('password');
    $adapter->setIdentity($values['name']);
    $adapter->setCredential(
      hash('SHA256', $values['password'])
    );
    // authentication attempt

```

```

    $auth = Zend_Auth::getInstance();
    $result = $auth->authenticate($adapter);
    // authentication succeeded
    if ($result->isValid()) {
        $auth->getStorage()
            ->write($adapter->getResultRowObject(null, 'password'));
        $this->view->passedAuthentication = true;
        $this->_forward('index', 'index', 'admin');
    } else { // or not! Back to the login page!
        $this->view->failedAuthentication = true;
        $this->view->loginForm = $form;
    }
}
}
public function logoutAction()
{
    Zend_Auth::getInstance()->clearIdentity();
    $this->_helper->redirector('index', 'index');
}
}
}

```

The main change here was to add additional View variables to flag some conditions to the view template as used below. I tend to use flags a lot with Views since they are simple booleans and delegate the translation of these flags into some meaning to the View itself. It's a really obvious gimmick I know but there are still lots of people who assign actual text without a second thought.

```

/application/views/scripts/author/login.phtml
<h2>Authentication</h2>
<p>Enter your author name and password below.</p>
<?php if($this->failedValidation): ?>
    <p class="error">Some problems were detected with the submitted form.</p>
<?php elseif($this->failedAuthentication): ?>
    <p class="error">The credentials supplied could not be authenticated. Please try again.</p>
<?php endif; ?>
<?php echo $this->loginForm->render() ?>


```

Lastly I made another tiny change to the template for our Administration Modules index page:

```

/application/admin/views/scripts/index/index.phtml
<h2>Administration</h2>
<?php if($this->passedAuthentication): ?>
    <p class="success">You have been successfully authenticated.</p>
<?php endif; ?>
<ul>
<li><a href="/admin/entry/add">New Entry</a></li>
</ul>

```

Upon authentication, as noted in our `AuthorController::loginAction()` method, users are forwarded to the Admin Module index page. This addition merely informs the user they were authenticated which explain why they've ended up there 

## Step 7: A Small Administration Template Update

Hindsight is great, but unfortunately you can't make use of it on a live blog, so I'll make a few quick changes now. What we'll do is create a simple Administration Menu in the left column of our Administration Module's templates.

Open up the file `/application/admin/views/scripts/index/index.phtml` and edit as follows:

```
<h2>Administration</h2>
<?php if($this->passedAuthentication): ?>
    <p class="success">You have been successfully authenticated.</p>
<?php endif; ?>
<p>Welcome, <?php echo Zend_Auth::getInstance()->getIdentity()->realname ?>.</p>
```

The change merely replaces the early menu block with a simple greeting using the Author's real name as stored from previous authentication.

Open up the file `/application/admin/views/layouts/admin.phtml` and edit as follows:

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

    <div class="container">
        <div class="block">
            <div id="header" class="column span-24">
                <h1><a href="/">Lorem Ipsum</a></h1>
            </div>
        </div>
        <div class="block">
            <div id="left" class="column span-5">
                <div class="menu">
                    <ul>
                        <li class="menu-header">Account</li>
                        <li><a href="/author/logout">Logout</a></li>
```

```

        <li class="menu-header">Entries</li>
        <li><a href="/admin/entry/add">New Entry</a></li>
    </ul>
</div>
</div>

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

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

```

If editing the menu in looks odd, don't worry yet. Those who really don't like it can add it as a template partial or pass an array of menu data into a custom View Helper (I'll be doing something similar in the near future to tidy up layouts). Finally, edit our stylesheet at `/public/css/style.css` to include the following styles for the now modified column.

```


div.menu {
    padding: 1em 1em;
    margin-top: 3em;
}
div.menu ul {
    list-style: none;
}
div.menu ul li.menu-header {
    padding-top: 1em;
    font-weight: bold;
    font-size: 110%;
}
div.menu ul > li:first-child.menu-header {
    padding-top: ;
}
div.menu a {
    text-decoration: none;
    font-weight: bold;
}

```

Try out the application once more. It should now look a bit more presentable.

## Conclusion

Implementing an ACL solution with Zend\_Acl hasn't proven difficult, in part due to the simplicity of our immediate needs. I would reiterate that remembering that Resources are not Controllers/Action, but merely virtual values which can be mapped or interpreted as such, is quite important. Using a 1:1 mapping is simply a common convention.

Since we're on the cusp of entering the realm to create new blog entries, an update to styling is welcome. It's not the final styling obviously, but looking at the minimalistic monotone output in my browser was becoming tiring 

In the next edition of this series we'll visit the task of creating, editing and displaying new blog entries.


Posted by Pádraic Brady in PHP General, PHP Security, Zend Framework at 00:48

Pádraic,

Again, I wish to thank you very much for your hard work on this series. I've been working with ZF since its initial conception, and your blog entries have always been on the top of my reading priority. This series has been extremely needed, due to the lack of direction currently, and the extensive possibilities that ZF provides.

Acl is so flexible, yet a hard concept to grasp. I'd love to see some real-world examples of Assertions in the Acl, but I'm not sure that's in the realm of this series.


I've never refreshed someone's blog to see if a new post has been made ever... Consider yourself a first! This series is a valuable asset to the ZF Community. Keep up the good work! Anonymous on May 9 2008, 21:07

Quick fix! In your update to the AuthorController I wasn't sure if you meant to display the full code for the file but it is definitely shorthanded there 


Great job with Acl.

-d Anonymous on May 9 2008, 23:10

I am trying my best to understand everything and from what I can tell we have not specified that the admin area is off limits to guests correct? Or did I miss something?

I really like the way you did the forms, I can tell that error/success reporting using CSS is definitely one of your strong points 

Anonymous on May 9 2008, 23:26

It's Friday and I hope you're enjoying a pint somewhere 

Anonymous on May 10 2008, 00:10

@David Arnold:

Good catch, Serendipity was cutting off part of the entry there.

We did specify Guest would be denied access to the Admin Module; it's one of the 4 analysed rules.

My CSS ability using Blueprint default classes is legendary!

@Chad: Just one pint? 

Anonymous on May 10 2008, 01:05

Pádraic, many thanks for an excellent tutorial so far. I did play around with ZF when version 1 was released, but not since then.

I am trying to use Postgres instead of MySQL but am getting errors when trying to authenticate.

```
Fatal error: Uncaught exception 'Zend_Db_Adapter_Exception' with message 'SQLSTATE[08006] [7] invalid connection option
"adapter"' in /home/ben/www/zend_library/Zend/Db/Adapter/Pdo/Abstract.php:131 Stack trace: #0
/home/ben/www/zend_library/Zend/Db/Adapter/Abstract.php(743): Zend_Db_Adapter_Pdo_Abstract->_connect() #1
/home/ben/www/zend_library/Zend/Db/Adapter/Abstract.php(813): Zend_Db_Adapter_Abstract->quote('5e884898da28047...', NULL)
#2 /home/ben/www/zend_library/Zend/Auth/Adapter/DbTable.php(354): Zend_Db_Adapter_Abstract->quoteInto("password" = '?',
'5e884898da28047...') #3 /home/ben/www/zend_library/Zend/Auth/Adapter/DbTable.php(285):
Zend_Auth_Adapter_DbTable->_authenticateCreateSelect() #4 /home/ben/www/zend_library/Zend/Auth.php(118):
Zend_Auth_Adapter_DbTable->authenticate() #5 /home/ben/www/application/controllers/AuthorController.php(35):
Zend_Auth->authenticate(Object(Zend_Auth_Adapter_DbTable)) #6 /home/ben/www/zend_library/Zend/Controller/Action.php(502):
AuthorController->loginAction() #7 /home/ben/www/ze in /home/ben/www/zend_library/Zend/Db/Adapter/Pdo/Abstract.php on line 131
```

I have changed the adapter in config.ini to PDO\_PGSQL although I am not to sure what to do about the schema, but I did try changing the table\_name in loginAction() to schema.table\_name but this didn't work either. I am a little confused about the error. Does anyone have any suggestions ? Sorry I know it is a little off the tutorial. Anonymous on May 10 2008, 03:26

I'd suggest skipping the shorter connection code I use, and using the fuller API per the manual. Looks like the Postgres adapter is having trouble parsing the config data for some reason. Anonymous on May 10 2008, 03:32

How odd, I am getting no errors and I am definitely logged out but when I go to /admin is takes me there. Hrm. Anonymous on May 10 2008, 03:53

It is a strange error I have found if I do

```
$db = Zend_Db::factory($config->db->adapter,array(
'host' => 'localhost',
'username' => 'ben',
'password' => 'xxx',
'dbname' => 'ben' ));
```

everything works.

the config.ini file contains

[general]

```
;Database connection settings
db.adapter=Pdo_Pgsql
db.host=localhost
db.username=ben
db.password=xxx
db.dbname=ben
```

so I don't know whats going on. Anonymous on May 10 2008, 04:57

Why are you using front controller parameters and not the registry to save authentication and authorisation objects?

Have a nice weekend. Anonymous on May 10 2008, 05:14

I personally dislike too much reliance on any singleton class like a Registry. I know, pass the Registry in an all is well




. But still...it

smells a bit naughty but I just avoid it. Even knowing Zend\_Auth must be a singleton gives me a rash if I dwell on it too much...

Put simply, if it acts like a global or exhibits any global related characteristics I will go out of my way to avoid referencing it. I prefer straightup dependency injection via constructors, setters, service locators, or dependency injectors. I am one of those waiting for Zend\_Di to pass inspection... Anonymous on May 10 2008, 05:42

To be perfectly honest, I don't know either. Perhaps raise it on the mailing list? I haven't used Zend\_Db with anything other than MySQL since I normally use a full ORM or a extended Data Mapper to power my Models with other databases. Anonymous on May 10 2008, 05:46

I was imagining one-at-a-time, but hey, more power to the one-in-each-hand approach 

Looking forward to the next installment. I've added the start of an author management component. Listing authors was easy enough. Haven't figured out how to pass an author record from my /admin/author controller to the Form plugin. Anonymous on May 10 2008, 05:55

Maybe you should use FlashMessenger helper for outputting messages on failed and successful authentication.

Placing that piece of code in layout/view script could be reused later for other messages (entry successfully added, entry deleted, etc.) Anonymous on May 10 2008, 07:14

Hello, Ben Hewson!

Try the following:

=====  
the config.ini file contains


[general]

```
;Database connection settings
db.adapter=Pdo_Pgsql
db.params.host=localhost
db.params.username=ben
db.params.password=xxx
db.params.dbname=ben
```

=====

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

 Anonymous on May 10 2008, 07:23

View Helpers are another topic for another Part 

. I haven't touched them yet until I can introduce them in detail. Anonymous on May 10 2008, 07:33

Regarding: "Note: Setting the Module name on the request object for forwarding is required when using Modules. That includes setting references to non-moduled controllers/actions with the Module name of "default"."

I don't think you have to specify module when forwarding to controller/action within the default module. I've just tried without it, and it works.


BTW, would it be to much to ask for a websvn or ViewVC frontend for your subversion repository? Anonymous on May 10 2008, 10:18

I shouldn't work - your request for the Admin Module would have "admin" set as the module name when it should be "default". Anonymous on May 10 2008, 14:15

Great work!

Can't wait your next tutorials... Anonymous on May 11 2008, 23:59

It would be nice if you could show examples of how to ensure that only authors can edit their own entries (if there were multiple entries of course). There's a lot of information about this scattered in various places, but no full clear example like this. I think this were most people lose site of the ACL (you're example is a pretty trivial one which is well documented, I think). Anonymous on May 12 2008, 01:12

I'd tried that myself, but there is a nasty bug in PHP 5.2.0 (default on Debian installations) that makes the flashmessenger error 


Anonymous on May 13 2008, 05:25

This is not a Zend Framework tutorial. Its a work of art!

May I suggest, as a part of the final chapter (whenever that is), a little tweaking of the code / code layout / directory structure. I am sure you've had some new ideas after publishing each part.

Awesome work!!

Thanks Anonymous on May 13 2008, 12:17

It's a really great idea that I will implement before the series ends along with an Editor role. For the moment however, bear in mind it's not a trivial example - this is a real application and the ACL rules part of its minimal requirements 

. As I said before - I will replace

Serendipity with this application as my blogging platform. Anonymous on May 13 2008, 16:03

Thanks for the high praise, Pedro! I am currently compiling a list of fixes, improvements, edits, and an assortment of points where things could be better. At the end of the series a revised version will be published in multiple formats. Anonymous on May 13 2008, 16:22

PÃ¡draic:

This has been a wonderful, very educational series so far. I'm loving every minute of it.

One question: when logging in, the user is forwarded to "/admin/index/index" instead of being redirected, with the effect that the address bar still reads "/author/login" after landing on the admin page. Something similar happens when a non-authenticated guest tries to access the admin module: they are redirected to the login page as desired, but the address bar still shows "/admin". Is there a way to fix this? Anonymous on May 14 2008, 01:07

Instead of forwarding, you can use the Redirector Action Helper which has a similar use, e.g. `$this->_helper->redirector('index', 'index');` Anonymous on May 14 2008, 03:07

hi,  
I just reviewed your Acl Class and wondering why you pass the \$auth object in the constructor. I checked also the repository but even there is not made use of it. Is there a deeper motivation? Anonymous on Jan 12 2009, 04:43