Zend Framework Workshop

Matthew Weier O'Phinney
Software Architect,
Zend Technologies

A loosely-coupled framework with a flexible architecture that lets you easily build modern web applications and web services.

Who am I, and what do I do?

- **I'm just another PHP developer**

Who am I, and what do I do?

- **Contributor to Zend Framework since pre-0.1.0; used ZF internally at Zend.**
- **Component lead on MVC since August 2006.**
- **Full-time Zend Framework developer since December 2007.**
- **Software Architect since April 2008.**

Goals for the Workshop

- **General understanding of Zend Framework**
- **Understanding of ZF MVC components**
  - Plugins and Helpers
  - Zend_Layout
  - Zend_Form
  - View Helpers
- **Understanding of Zend_Db_Table**
- **Basic understanding of authentication and ACLs**
- **Understanding of various utility classes (configuration, registry, i18n, search, etc.)**
- **How to get help and contribute**

What is Zend Framework?

It's just another PHP framework.

No, what is Zend Framework?

It's a glue library.

No, really, what is Zend Framework?

- **PHP 5 library for web development productivity**
- **Open source**
  - New BSD license is business-friendly
  - Free for development and distribution
  - CLA process assures that the code is free of legal issues
- **Class library – fully OOP**
- **Documentation – in many languages**
- **Quality & testing – fully unit tested**
  - >80% code coverage required

Zend Framework philosophy

- **Simplicity and Extensibility**
  - Easy solutions for the 80% most commonly-used functionality for web applications
  - Extensibility enables easy customization, to solve the remaining 20%
  - No complex XML configuration files
- **Good object-oriented and agile practices**
  - Use-at-will architecture, but also:
  - Full stack framework
  - Designed for extensibility
  - Frequent testing
  - Frequent interaction with user community

Zend Framework quality process

1. **Say what you're going to do**
   - Proposal process
2. **Do it**
   - Write object oriented components
   - Unit test your component
     - We encourage test-driven development (TDD)
   - Document how it works
3. **Verify it matches what you said**
   - Open-source development and community review
   - Frequent and thorough testing with PHPUnit
   - Code coverage reports with PHPUnit
   - Review by internal Zend team for compliance

Getting Zend Framework

- **http://framework.zend.com/download**
- **Choose the format you want: zip, tarball**
- **Unpack it**
- **Point your include path at standard/library/**

Birds' Eye View of Zend Framework

What's in Zend Framework?

- **MVC**
- **Database**
- **I18N**
- **Auth and ACLs**
- **Web Services**
- **Mail, Formats, Search**
- **Utility**

- **Zend_Controller**
  - Front controller
  - Router
  - Dispatcher
  - Action controller
  - Plugins and Helpers
  - Request and Response
- **Zend_View**
  - PHP-based views
  - Helpers
- **Zend_Layout**
  - Two Step Views
- **Zend_Form**

What's in Zend Framework?

- **MVC**
- **Database**
- **I18N**
- **Auth and ACLs**
- **Web Services**
- **Mail, Formats, Search**
- **Utility**

- **Zend_Db_Adapter**
  - Adapters for most database extensions provided by PHP
- **Zend_Db_Profiler**
- **Zend_Db_Select**
- **Zend_Db_Table**
  - Zend_Db_Table_Rowset
  - Zend_Db_Table_Row

What's in Zend Framework?

- **MVC**
- **Database**
- **I18N**
- **Auth and ACLs**
- **Web Services**
- **Mail, Formats, Search**
- **Utility**

- **Zend_Locale**
- **Zend_Date**
- **Zend_Measure**
- **Zend_Translate**
  - Adapters for PHP arrays, CSV, gettext, Qt, TMX, and Xliff

What's in Zend Framework?

- **MVC**
- **Database**
- **I18N**
- **Auth and ACLs**
- **Web Services**
- **Mail, Formats, Search**
- **Utility**

- **Zend_Auth**
  - Zend_Db_Table adapter
  - HTTP Digest
  - HTTP Basic
  - Write your own adapters
- **Zend_Session**
  - Persist identities
- **Zend_Acl**
  - Roles
  - Resources
  - Rights

What's in Zend Framework?

- **MVC**
- **Database**
- **I18N**
- **Auth and ACLs**
- **Web Services**
- **Mail, Formats, Search**
- **Utility**

- **Zend_Http_Client**
- **Zend_Rest_Client**
- **Zend_Service**
  - Many, many popular web APIs implemented
- **Zend_Feed**
  - RSS and Atom
- **Zend_Gdata**
  - Google access API to most Google services
- **Zend_XmlRpc**
  - Consume and serve XML-RPC services

What's in Zend Framework?

- **MVC**
- **Database**
- **I18N**
- **Auth and ACLs**
- **Web Services**
- **Mail, Formats, Search**
- **Utility**

- **Zend_Mail**
  - Read or send email
- **Zend_Mime**
  - Parse MIME encoded text
- **Zend_Pdf**
  - Read, edit, and create PDF documents
- **Zend_Search_Lucene**
  - Search Lucene indices
  - Apache Lucene compatibility

What's in Zend Framework?

- MVC
- Database
- I18N
- Auth and ACLs
- Web Services
- Mail, Formats, Search
- Utility

- Zend_Cache
- Zend_Config
- Zend_Console_Getopt
- Zend_Filter
- Zend_Filter_Input
- Zend_Loader
- Zend_Log
- Zend_Memory
- Zend_Registry
- Zend_Validate

What's in Zend Framework?

- **MVC**
- **Database**
- **I18N**
- **Auth and ACLs**
- **Web Services**
- **Mail, Formats, Search**
- **Utility**

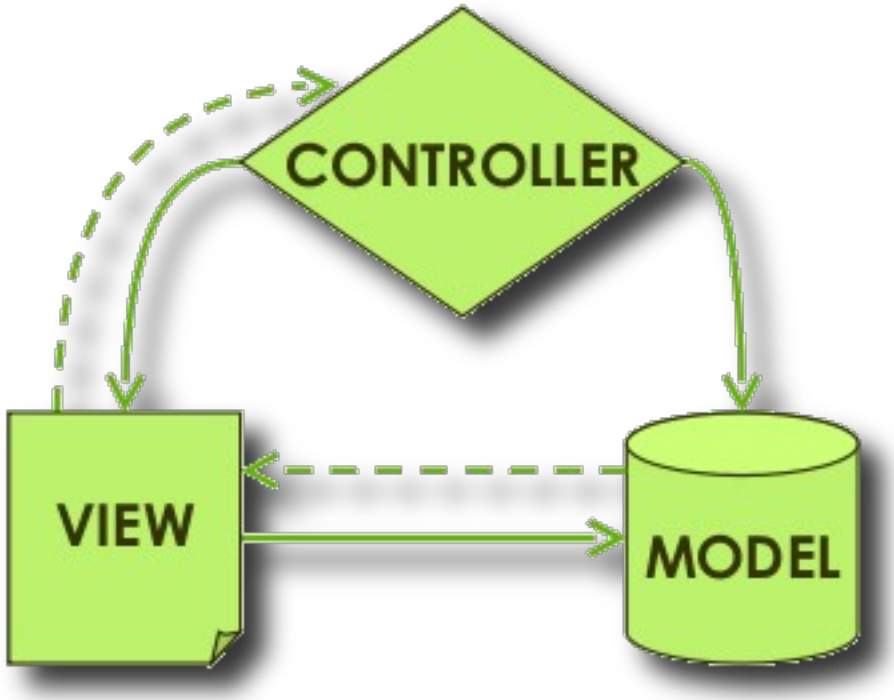
And much, MUCH more...

MVC Overview

What is MVC?

- **Model: data provider and data manipulation logic (typically)**
- **View: user interface**
- **Controller: request processor and application flow**

- **Design pattern originating with Smalltalk and dating to the 1970s**



Why MVC?

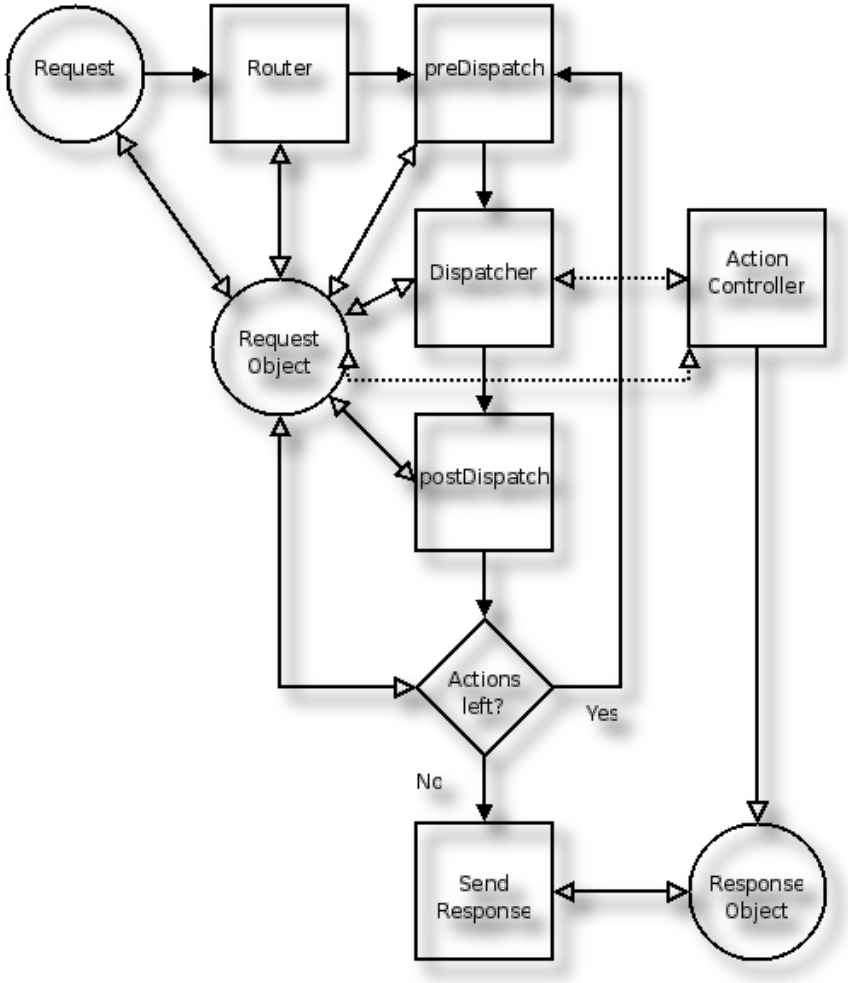
- **Simple solution for most web applications**
- **Flexible and extensible**
- **Supports advanced applications**
- **Best practice for application workflow**

Zend Framework MVC Features

- ***Front Controller*** **handles all incoming requests and emits final response**
- ***Router*** **maps URL and/or request to appropriate** ***Action Controller*** **and Action Method**
- ***Dispatcher*** **dispatches Action Controller**
- ***Action Controller*** **performs appropriate application logic**
- ***Request*** **object encapsulates request environment**
- ***Response*** **object aggregates response content and headers**

As an illustration...

- **In practice, it's not as hard as this looks...**



At its most basic...

- **Default URL format: http://framework.zend.com/manual/search**
  - 'manual' maps to the name of a **Controller *class***
  - 'search' maps to an **Action *method*** in that class

```
12 class ManualController extends Zend_Controller_Action
13 {
14     /**
15      * Search the manual
16      *
17      * @return void
18      */
19     public function searchAction()
20     {
21     }
```

Controllers are *nouns*, Actions are *verbs*

...add a view...

- **View scripts are just PHP, and are auto-rendered by default (more on that later):**

```
 4 <?= $this->dynamicHeader('subPageMainHeader', 'Search the Manual') ?>
 5
 6 <p class="first">
 7     Enter your search criteria in the sidebar.
 8 </p>
 9
10 <div class="dotted-line"></div>
```

...setup your front controller...

- **We call this “bootstrapping” your application:**

```
$front = Zend_Controller_Front::getInstance();
$front->addControllerDirectory('../application/controllers');
$front->dispatch();
```

And get results:

- **Some sample output:**

Add an ErrorController

- **The *ErrorController* is invoked when a controller or action is not found, or when an exception is thrown in your application logic.**

Sample ErrorController

```
class ErrorController extends Zend_Controller_Action
{
    public function errorAction()
    {
        $errors = $this->_getParam('error_handler');

        switch ($errors->type) {
            case Zend_Controller_Plugin_ErrorHandler::EXCEPTION_NO_CONTROLLER:
            case Zend_Controller_Plugin_ErrorHandler::EXCEPTION_NO_ACTION:
                // 404 error -- controller or action not found
                $this->getResponse()->setRawHeader('HTTP/1.1 404 Not Found');
                $this->view->code    = 404;
                $this->view->message = 'Page not found';
                break;
            default:
                // application error
                $this->getResponse()->setRawHeader('HTTP/1.1 500 Application Error');
                $this->view->code    = 500;
                $this->view->message = 'Application error';
                $this->view->info    = $errors->exception;
                break;
        }
    }
}
```

...and views for your ErrorController

- **You will often want the error details shown only in specific environments**

```
<h2><?= $this->message ?></h2>
<? if ((500 == $this->code)
      && (Bugapp_Plugin_Initialize::getConfig()->showExceptions)): ?>
<h4>Error details:</h4>
<dl>
    <dt>Exception Code:</dt>
    <dd><?= $this->info->getCode() ?></dd>

    <dt>Exception Message:</dt>
    <dd><?= $this->info->getMessage() ?></dd>

    <dt>Stack Trace:</dt>
    <dd><pre>
    <?= $this->info->getTraceAsString() ?>
    </pre></dd>
</dl>
<? endif ?>
```

Add a site layout:

- **A sample layout script:**

```
<?= $this->doctype() ?>
<html>
<head>
    <?= $this->headTitle() ?>
    <?= $this->headLink() ?>
    <?= $this->headStyle() ?>
    <?= $this->headScript() ?>
</head>
<body>
    <?= $this->render('_topnav.phtml') ?>
    <div id="content">
    <?= $this->layout()->content ?>
    </div>
    <?= $this->placeholder('nav') ?>
</body>
</html>
```

Add a site layout:

- **Tell your bootstrap about it:**

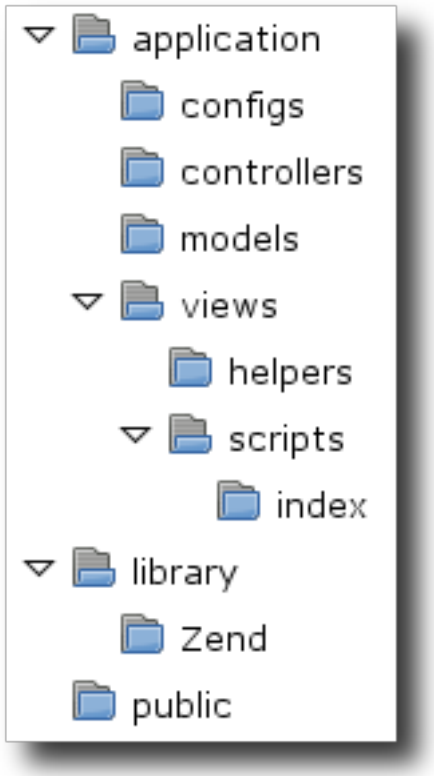
```
// Initialize layouts
Zend_Layout::startMvc($this->_root . '/application/views/layouts');
```

...and see what you get:

- **Sample output:**

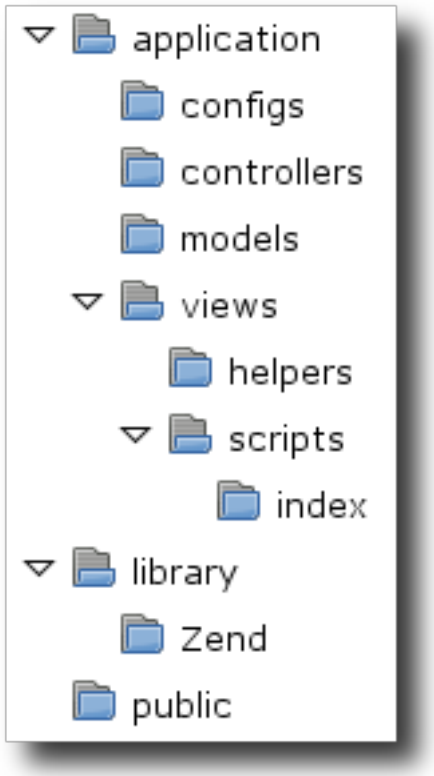
Some things you should know

- **ZF *suggests* a directory structure:**
  - *application/* contains your application code
  - *library/* contains Zend Framework and custom libraries
  - *public/* contains publicly available files, including your *bootstrap* file
- **...but we don't lock you in.**



Some things you should know

- **Your application directory:**
  - *controllers/* contains your Action Controller classes.
  - *models/* contains your Model classes.
  - *views/* contains your view scripts, with each controller's view scripts contained in a directory of the same name. It also contains View Helpers (more later).
- **...but we don't lock you in; all of this is configurable.**



Models

- **First rule of Zend Framework's MVC: There is no Zend_Model.**
- **Second rule: Model !== Database. It can be a web service, the file system, a messaging system, etc.**
- **ZF *will* address the question of the Model; until then, creation of models is up to the developer. Specific considerations:**
  - Input filtering
  - Attaching data source(s)
  - Retrieving result(s) from the model

What was that about extensibility?

- ***Zend Framework's MVC is extensible, right?***

  What does that mean?

MVC Extension Points

- **Dispatching happens in a loop; *more than one action* may be executed per request.**
- **Front Controller defines event hooks around each process; *Plugins* may trigger for each hook.**
- ***Action Helpers* may be loaded on demand and utilized by any Action Controller; you do not need to extend the Action Controller to add functionality.**
- ***View Helpers* may be loaded on demand and utilized by the View; you do not need to extend Zend_View to add functionality**

More than one action?

- **If the user does not have rights to an action, *forward* them to another**
- **Setup a stack of actions to perform (widgetized sites)**
- **Go to a different action on failure (or success)**

Plugin hook events?

- **Occur at *routeStartup()* and *Shutdown()* -- setup routing tables, or post process following routing.**
- **Occur at *dispatchLoopStartup()* and *Shutdown()* -- perform common initialization prior to dispatching any actions, and perform post processing when all are done.**
- **Occur at *preDispatch()* and *postDispatch()* of action controllers; check ACLs prior to executing an action, or determine what to do next afterwards.**

Plugin Examples

- **ErrorHandler – checks for exceptions, and forwards to your registered ErrorController to report errors.**
- **Zend_Layout – Two Step View pattern; wrap your content in a sitewide layout.**

Writing your own plugin:

- **Extend Zend_Controller_Plugin_Abstract**
- **Implement one or more event methods**

```
class Bugapp_Plugin_Auth extends Zend_Controller_Plugin_Abstract
{
    public function dispatchLoopStartup(Zend_Controller_Request_Abstract $request)
    {
        $view   = Zend_Layout::getMvcInstance()->getView();
        $auth   = Zend_Auth::getInstance();
        $values = array(
            'user_id'    => null,
            'user_name'  => null,
            'user_email' => null,
        );
        if ($auth->hasIdentity()) {
            $identity = $auth->getIdentity();
            $values = array(
                'user_id'    => $identity->id,
                'user_name'  => $identity->username,
                'user_email' => $identity->email,
            );
        }
        $view->assign($values);
    }
}
```

Register your plugin

- **Register plugin instances with the front controller:**

```
$this->_front->registerPlugin(new Bugapp_Plugin_Auth());
```

Action Helpers?

- **Helper *classes* that can interact with the action controller**
- **Push commonly used code into action helpers, and use on-demand**
- ***Helper Broker* has hooks for initialization, and pre/postDispatch events; automate tasks that require controller integration.**
- **When used correctly, no need to create a base Action Controller class with your common utility methods.**

Action Helper Examples

- **ViewRenderer – resolves view script path based on current controller and action, and automatically renders it after action completion.**
- **Layout – manipulate the layout object (perhaps disable it, or choose a different layout).**
- **Load forms, models, etc. based on current controller and action.**

Write an Action Helper:

- **Extend Zend_Controller_Action_Helper_Abstract**
- **Use direct() to provide a method that can be called as if it were a helper broker method**

```php
class Bugapp_Helper_GetForm extends Zend_Controller_Action_Helper_Abstract
{
    protected $_forms = array();

     * Load and return a form object
    public function getForm($form, $config = null)
    {
        $form = ucfirst($form);
        $class = 'Bugapp_Form_' . $form;
        if (!array_key_exists($class, $this->_forms)) {
            $this->_forms[$class] = new $class($config);
        }
        return $this->_forms[$class];
    }

     * Proxy to getForm()
    public function direct($form, $config = null)
    {
        return $this->getForm($form, $config);
    }
}
```

Register your Action Helper

- **Simplest and most flexible: register your action helper with the helper broker by passing a class prefix:**

```
Zend_Controller_Action_HelperBroker::addPrefix('Bugapp_Helper');
```

Use an action helper

- **Grab the helper via overloading, and call methods:**

```
$model = $this->_helper->getModel->getModel('user');
```

- **Or, if you implemented direct(), call it as a method:**

```
$user    = $this->_helper->getModel('user')->fetchUser($developer);
```

How does ZF do Views?

- **Zend_View renders view scripts written in PHP**
  - PHP is itself a templating language; let's leverage it
  - Why force developers to learn another domain-specific language?
- **ZF recommends using short tags and alternate logical/looping syntax for brevity and readability of view scripts**
  - But you can use long tags if you want
- **Provides basic functionality for assigning variables and conditionally escaping data**

Example View Script

```
<h2><?= $this->listType ?></h2>

<ul class="buglist">
<? if (0 == count($this->bugs)): ?>
    <li><b>No bugs found matching this criteria</b></li>
<? else: ?>
    <? foreach ($this->bugs as $bug): ?>
    <li><?= $this->bugLink($bug->id, $bug->summary) ?></li>
    <? endforeach ?>
<? endif ?>
</ul>
```

View Helpers?

- **Push that common code that clutters your views into helper classes**
- **Loaded on-demand**
- **Use to access models, format data, pull from the registry, etc.**

View Helper Examples

- **doctype(): specify as well as render the DocType for the current page.**
- **htmlList(): create an ordered or unordered list from an array of data.**
- **form*(): render form elements, fieldsets, and forms based on input provided to the helper.**
- **placeholder(): store data for later rendering.**
- **partial(): render another view script in its own variable scope, using the variables you pass it.**

View Helper Example Usage

```
<li><?= $this->bugLink($bug->id, $bug->summary) ?></li>
```

What are these “layouts”?

- **Zend_Layout is Zend Framework's solution for Two Step Views**
  - Wrap application content in a sitewide template, the ***layout***
  - Allow hinting from application views to sitewide layout
  - Layout scripts are just view scripts; all the functionality of Zend_View is available
  - By default, any response segment is available as a layout placeholder; the default, 'content', is always available.

Zend_Layout Usage

- **Early in your application:**
  - Specify doctype
  - Initialize layout with layout path

```
// Setup View
$view = new Zend_View();
$view->doctype('XHTML1_TRANSITIONAL');

// Set view in ViewRenderer
$viewRenderer = Zend_Controller_Action_HelperBroker::getStaticHelper('ViewRenderer');
$viewRenderer->setView($view);

// Initialize layouts
Zend_Layout::startMvc($root . '/application/views/layouts');
```

Zend_Layout Usage

- **In your view script, hint to the layout:**

```
<? $this->headTitle($this->bug->summary) ?>
```

Zend_Layout Usage

- **In your layout view script, render placeholders and sitewide elements:**

```
<head>
    <?= $this->headTitle() ?>
```

What are placeholders?

- **Placeholders *persist* content between view scripts and view instances.**
- **Placeholders can *aggregate* content.**
- **Placeholders can *prefix* and *postfix* the content they hold, as well as provide *separator* content.**
- **Several targetted placeholders shipped for frequent tasks: headTitle(), inlineScript(), etc.**
- **Basic method for hinting from the application views to the layout view.**

Placeholder example

```
$view->placeholder('nav')->setPrefix('<div id="nav">')
                         ->setPostfix('</div>');
```

```
<?= $this->placeholder('nav') ?>
```

What are partials?

- **By default, all view variables persist for the lifetime of the view object. Sometimes this causes collisions.**
- ***Partials* provide a local variable space for the rendered view script.**
- **Only variables explicitly passed to a partial are available in the partial.**
- **Useful for looping... which has spawned the *PartialLoop* helper.**

Partial usage

```
<li><a href="/bug/list">Open bugs</a>
    <? if (null !== $this->user_name): ?>
    <ul>
        <li><a href="/bug/list/status/open/developer/<?= $this->user_name ?>">Your
        <li><a href="/bug/list/status/open/reporter/<?= $this->user_name ?>">Open
    </ul>
    <? endif ?>
</li>
```

```
<?= $this->partial('bug/_nav.phtml', array('user_name' => $identity->username)) ?>
```

What's that we're missing from MVC? Oh, right, Zend_Form

Zend_Form Basic Usage

Create elements: Username:

```
$username = new Zend_Form_Element_Text('username');
$username->addFilters(array('StringTrim', 'StringToLower'))
         ->addValidators(array(
             'Alnum',
             array('StringLength', false, array(3, 20))
         ))
         ->setRequired(true)
         ->setLabel('Username');
```

- **Multiple filters (filter chain!)**
- **Multiple validators (validator chain!)**
- **Required**
- **Don't forget the label!**

Zend_Form Basic Usage

Create elements: Password:

```
$password = new Zend_Form_Element_Text('password');
$password->addFilter('StringTrim')
         ->addValidator('StringLength', false, array(6))
         ->setRequired(true)
         ->setLabel('Password');
```

- Single filter
- Single validator
- Required
- Don't forget the label!

Zend_Form Basic Usage

Create elements: the Login button:

```
$login = new Zend_Form_Element_Submit('login');
$login->setRequired(false)
      ->setIgnore(true)
      ->setLabel('Login!');
```

- Need to display the button
- But we don't want to validate it or include it when pulling values

Zend_Form Basic Usage

Create the Form object:

```
// In a controller action:
$form = new Zend_Form();
$form->addElements(array($username, $password, $login));
if ($this->getRequest()->isPost()) {
    if ($form->isValid($this->getRequest($this->getPost()))) {
        // success
    }
}
$this->view->form = $form;
```

- Attach elements
- Check if valid – does all input filtering
- Pass it to the view

Zend_Form Basic Usage

Create the view script:

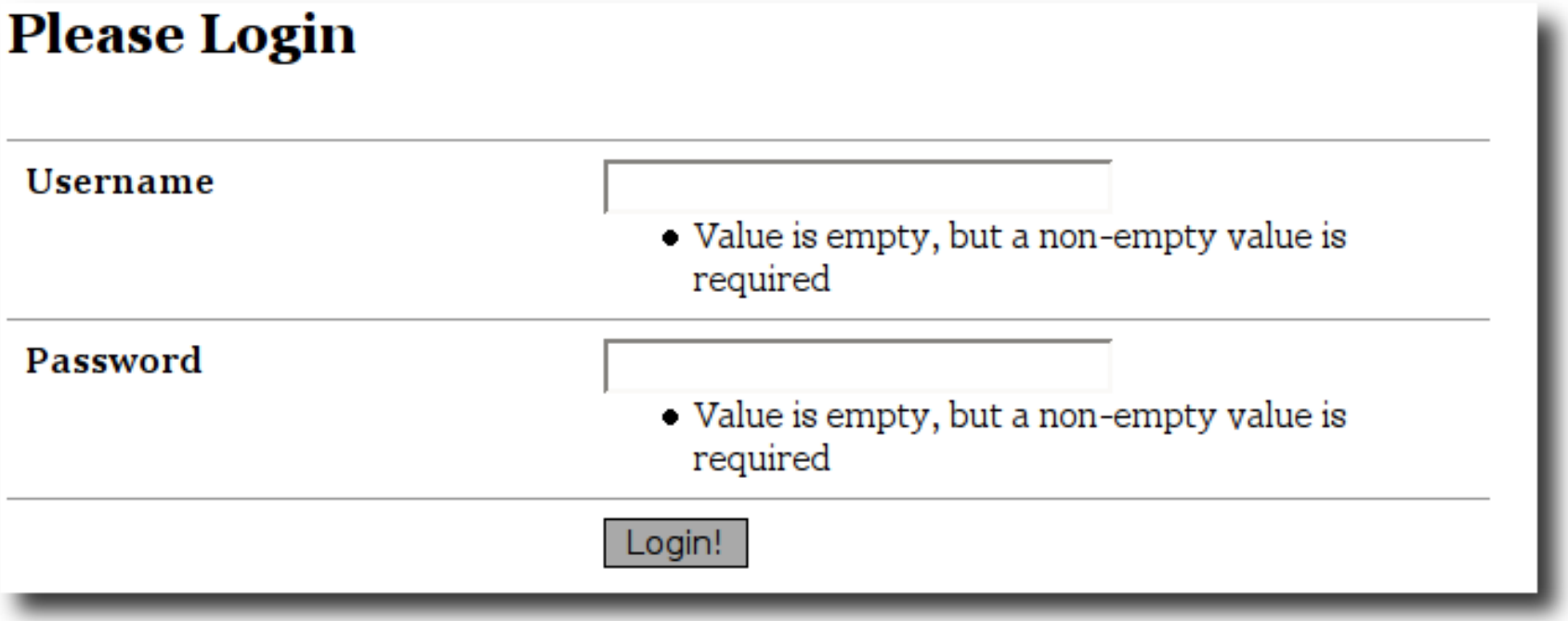
```
<h2>Please Login</h2>
<?= $this->form ?>
```

Zend_Form Basic Usage

First time viewing the form:

Zend_Form Basic Usage

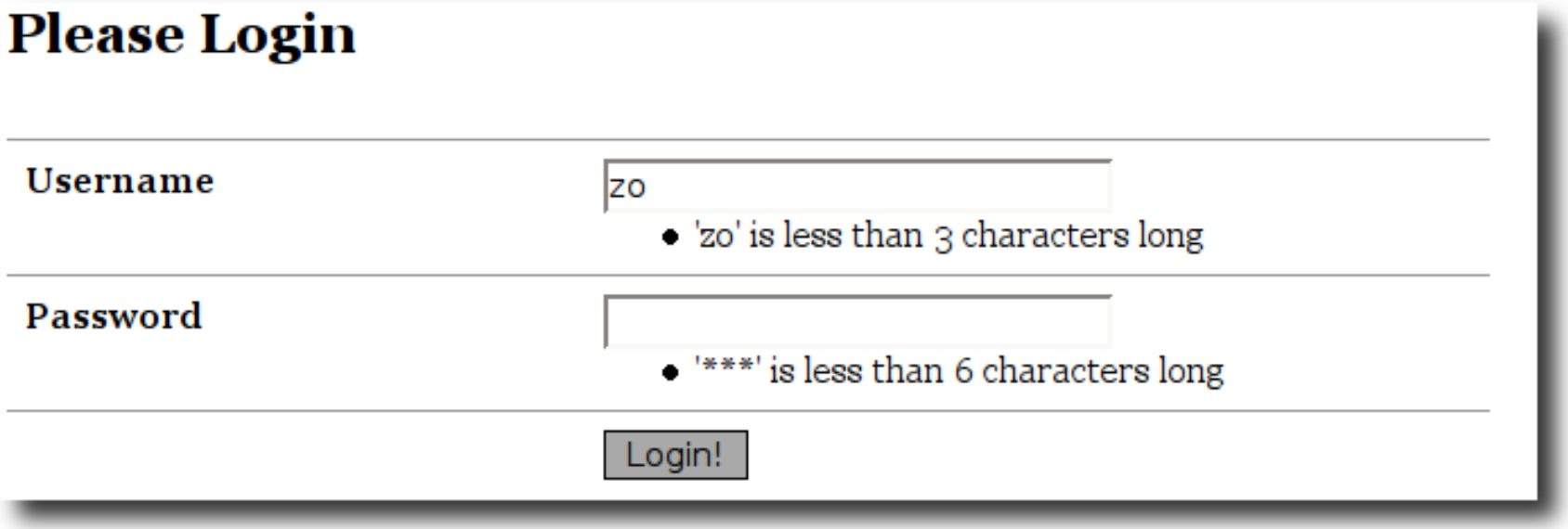
Results when submitting empty values:



- **Note: *required* flag has a correlation with the errors reported**

Zend_Form Basic Usage

Results when submitting invalid values:



- **Note: errors are reported!**

Zend_Form Features and Benefits

- **Internationalization: localize your forms for your customers!**
- **Partial and Full Set data validation**
- **Filter and Validation chains per element**
- **Fully customizable output**
- **Adheres to Zend_Validate_Interface**
  - Allows you to plug forms and/or elements in as validators for your models -- which means you could potentially replace Zend_Filter_Input classes in your models and thus make your models directly renderable!
- **Break forms into visual and/or logical groups**

Overview of Zend_Form's Architecture

Architecture Overview

- **Base classes**
  - forms
  - elements
  - display groups
  - sub forms
- **Plugins**
  - filters
  - validators
  - decorators
  - elements
- **Utilities**
  - plugin loaders
  - translators

Classes: Zend_Form

- **Model Forms**
  - Store and manipulate collections of elements and groups of elements
  - Validate attached elements and sub forms
  - Store and manipulate decorators for rending the form
- **Class: Zend_Form**

Classes: Zend_Form_Element

- **Store and manipulate element metadata**
- **Store and manipulate validator chains**
- **Store and manipulate filter chains**
- **Store and manipulate decorators for rendering element**
- **Base class: Zend_Form_Element**

- **Element types:**
  - Button
  - Checkbox
  - Hash (CSRF protection)
  - Hidden
  - Image
  - MultiCheckbox
  - Multiselect
  - Password
  - Radio
  - Reset
  - Select
  - Submit
  - Text
  - Textarea

Classes: Zend_Form_DisplayGroup

- **Group elements visually when rendering**
- **Collection of one or more elements**
- **Order display group in form, and elements within display group**
- **Class: Zend_Form_DisplayGroup**

Classes: Zend_Form_SubForm

- **Group elements logically**
  - For display purposes
  - For validation purposes
- **Potential uses**
  - Multi-page forms (each sub form used per page)
  - Dynamic forms (e.g., todo list, where each todo item is it's own mini-form)
- **Class: Zend_Form_SubForm**

Plugins

- **Utilizes Zend_Loader_PluginLoader for loading plugin classes**
- **Specify alternate class prefixes and paths to load:**
  - new plugins
  - alternate versions of standard plugins
- **Powerful and easy way to extend Zend_Form functionality**

Plugins: Filters

- **Normalize or filter input prior to validation**
- **Uses Zend_Filter classes by default**

- **Some available filters:**
  - Alnum
  - Alpha
  - Digits
  - HtmlEntities
  - StringToLower
  - StringToUpper
  - StringTrim
  - StripTags

Plugins: Validators

- **Validate input against one or more rules**
- **Uses Zend_Validate classes by default**

- **Some available validators:**
  - Alnum
  - Alpha
  - Date
  - EmailAddress
  - InArray
  - Int
  - Regex
  - StringLength

Plugins: Decorators

- **Render elements and forms by decorating them**
- **Uses pseudo-Decorator pattern**
- **More later...**

- **Some available decorators:**
  - Callback
  - Description
  - Errors
  - Fieldset
  - Form
  - HtmlTag
  - Label
  - ViewHelper
  - ViewScript

Plugins: Elements

- **Elements are loaded as plugins in Zend_Form**
- **You can create your own versions of standard elements, and still utilize Zend_Form's element factory methods**

- **Some standard elements:**
  - Button
  - Checkbox
  - Password
  - Radio
  - Select
  - Submit
  - Textarea
  - Text

Utilities

- **Plugin Loaders**
  - Load plugins
  - Register class prefixes and paths
- **Translators**
  - Zend_Translate and its adapters
  - Translate error messages and other translatable items

Zend_Form In-Depth

In-Depth: Plugins

- **As noted, uses Zend_Loader_PluginLoader for loading plugins**
- **Resources considered plugins:**
  - Filters (elements only)
  - Validators (elements only)
  - Decorators (all types)
  - Elements (forms only)
- **Generally, specify a class prefix, path, and plugin type**
- **Allows specifying both new plugins as well as local replacements of standard plugins**

Example: Validator plugin

```
// Specify additional or alternate validators for an element:
$element->addPrefixPath('My_Validate', 'My/Validate/', 'validate');

class My_Validate_Password { /* ... */ }

// Load My_Validate_Password:
$element->addValidator('Password');
```

Example: Decorator plugin

```php
// Specify additional or alternate decorators for the form:
$form->addPrefixPath('My_Decorator', 'My/Decorator/', 'decorator');

class My_Decorator_FormElements { /* ... */ }

// Load FormElements decorator:
$form->addDecorator('FormElements');

// Outputs "My_Decorator_Form_Elements"
var_dump(get_class($form->getDecorator('FormElements')));
```

- ***Replaces* standard "FormElements" decorator**

In-Depth: Decorators

- **Used to render elements, forms and groups**
- **Similar to *Decorator* pattern, but decorates string content using element and form metadata**
- **Each decorator decorates the content passed to it**
  - Initial content is always an empty string
  - Return value *replaces* previous value
  - Decorator internally can append, prepend, or replace provided
  - Typically Stack decorators from inside -> out to create output

In-Depth: Decorators

Example Decorator usage:

```
$element->addDecorators(array(
    'ViewHelper',
    'Errors',
    'Description',
    array('HtmlTag', array('tag' => 'dd')),
    array('Label', array('tag' => 'dt')),
));
```

- ViewHelper to render element
- Element error messages (appends)
- Element hint (appends)
- Wrap in <dd>
- Element label in <dt> tag (prepends)

In-Depth: Decorators

- **Each decorator has awareness of the element/form/etc.**
  - **Allows inspecting item to get metadata**
  - **Agnostic of class being decorated; retrieve with getElement() accessor regardless of class**
  - **Useful for building decorators that render only one aspect of an item**
    - Label
    - Description
    - Errors

In-Depth: Decorators

Some Standard Decorators

- **Callback**
  - Delegate to a specified callback
- **Description**
  - render from getDescription()
- **Errors**
  - render from getMessages()
- **Fieldset**
  - render content in a fieldset, with optional legend
- **FormElements**
  - Iterate through all elements, groups, and sub forms to generate content
- **Form**
  - Wrap content in an HTML form
- **HtmlTag**
  - Wrap content in HTML tags (or emit start or end tags

In-Depth: Decorators

Some Standard Decorators

- **Image**
  - Render a form image
- **Label**
  - render from getLabel()
- **ViewHelper**
  - Render using a view helper (typically pulled from element's 'helper' attribute)
- **ViewScript**
  - Render using a specified view script

In-Depth: Elements

Overview: What makes up an element?

- **Metadata**
- **Filters**
- **Validators**
- **Decorators**

In-Depth: Elements

Metadata

- **Stored as “properties” of the element, via overloading**
- **Anything that can better qualify an element**
- **Typically for rendering**
- **Examples:**
  - CSS class(es)
  - Javascript events (onClick, etc)
  - Explicit XHTML id
  - Javascript attribute hints (e.g., for Dojo)

In-Depth: Elements

Filters

- **For normalizing input prior to validation**
- **Objects implementing Zend_Filter_Interface**
- **Attaching to elements:**
  - Instantiate and attach
  - use addFilter(), addFilters(), or setFilters() as factories, using just the “short name” of the filter (class name minus prefix)
- **Use Zend_Form::setElementFilters() to set common filters for all elements en masse**

In-Depth: Elements

Validators

- **For validating input, to ensure it meets acceptance criteria**
- **Objects implementing Zend_Validate_Interface**
- **Attaching to elements:**
  - Instantiate and attach
  - use addValidator(), addValidators(), or setValidators() as factories, using just the “short name” of the validator (class name minus common prefix)

In-Depth: Elements

Decorators

- **For rendering as a form element**
- **Attaching to elements:**
  - Instantiate and attach
  - use addDecorator(), addDecorators(), or setDecorators() as factories, using just the “short name” of the decorator (class name minus common prefix)

In-Depth: Elements

Decorators

- **Default decorators for most elements:**
  - ViewHelper
  - Errors
  - HtmlTag (<dd>)
  - Label (rendered in a <dt> tag)
- **Some differ:**
  - Submit, Reset, Button, Image
- **Can set all elements en masse using Zend_Form::setElementDecorators()**

In-Depth: Forms

Overview: What makes up a form?

- Metadata
- Elements
- Display Groups
- Sub Forms
- Ordering
- Decorators
- Validation Methods

In-Depth: Forms

Metadata

- **Stored as “attribs” in the form**
- **Used to further qualify a form**
  - action
  - method
  - CSS class
  - XHTML id
  - Javascript events
- **Use various *Attrib(s)() accessors to manipulate**

In-Depth: Forms

Elements

- **Can attach concrete instances, or create new ones**
- **createElement()**
  - Create and return a new element, using paths and defaults set in the form object
- **addElement() / addElements() / setElements()**
  - Create and attach one or more elements to the form

Display Groups

- **Used to group elements visually when rendering**
- **Use fieldsets by default**
- **Use addDisplayGroup() as a factory to:**
  - create a new display group
  - attach specified elements to a display group
  - provide options specifying how the display group is rendered
    - decorators
    - legend

Sub Forms

- **What are they used for?**
  - Way to group items logically
  - Items that are related semantically
  - Repeatable groups of items (e.g., todo list tasks)
  - Single page of a multi-view form
- **Extends Zend_Form**
  - Has all functionality of Zend_Form
- **Use addSubForm() to add sub forms**
  - Accepts only concrete instances

Sub Forms

- **Utilize Array Notation:**

```
<dt><label for="user-username" class="required">Username:</label></dt>
<dd><input name="user[username]" id="user-username" value="" type="text"></dd>

<dt><label for="user-password" class="required">Password:</label></dt>
<dd><input name="user[password]" id="user-password" value="" type="password"></dd>

<dt></dt>
<dd><input name="user[login]" id="user-login" value="Login!" type="submit"></dd>
```

- **Uses sub form's name as array name**
- **Each element is a key in the array**
- **XHTML ids are inflected to use valid formats**
- **Array submitted for sub form is passed to sub form for validation**

In-Depth: Forms

Ordering

- **Each attached item – elements, display groups, sub forms – has its own order attribute; form object uses these to determine order**
- **Ordering performed only when iterating over form, rendering, or validating**
- **Elements attached to display groups**
  - When rendering, element order honored by display group; *not* double-rendered (removed from overall order).
  - When validating, order is honored; display groups are ignored.

In-Depth: Forms

Decorators

- **Default decorators for Zend_Form:**
  - FormElements (iterates over elements, display groups, and sub forms)
  - HtmlTag (dl tag with class of 'zend_form')
  - Form (wrap content in XHTML form tag)

Decorators

- **Default decorators for Zend_Form_SubForm and Zend_Form_DisplayGroup:**
  - FormElements
  - HtmlTag (<dl> tag)
  - Fieldset (using legend for item, if available)
  - DtDdWrapper (to keep flow of parent form; empty <dt>, and remainder of content in <dd>)

Validation Methods

- **isValid():** validate entire form (except optional fields which are missing or empty)
- **isValidPartial():** validate only fields that were submitted
- **getErrors():** returns array of element => error codes
- **getMessages():** returns array of element => error messages array

Zend_Db

What does Zend_Db do?

- **Database *Abstraction* Layer**
  - Vs. PDO, which is a db *access* layer
- **Provides a variety of database adapters**
  - PDO adapters
  - Driver specific adapters
- **Abstraction of common functionality**
  - limits/offsets
  - inserts/updates
  - Transactions
- **Connects to database on-demand**
  - Instantiating the object does not connect to the database; only when the first data access occur does it connect

How do I connect to a database?

- **Basic Usage: Zend_Db::factory() with parameters:**

```
$db      = Zend_Db::factory(
    'pdo_sqlite',
    array(
        'dbname' => dirname(__FILE__) . '/../data/db/bugs.db'
    )
);
```

How do I connect to a database?

- **Advanced Usage: Zend_Db::factory() with Zend_Config object:**
- **Zend_Config definition:**

```
db.adapter = "pdo_sqlite"
db.params.dbname = "../data/db/bugs.db"
```

- **And now the connection:**

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

Zend_Db_Select

- **Query Abstraction**
- **Object Oriented, fluent interface for dynamically building queries:**

```
$select->from(array('b' => 'bug'))
       ->joinLeft(array('i' => 'issue_type'), 'i.id = b.type_id', array('issue_type' => 'type'))
       ->joinLeft(array('r' => 'resolution_type'), 'r.id = b.resolution_id', array('resolution'))
       ->joinLeft(array('p' => 'priority_type'), 'p.id = b.priority_id', array('priority'))
       ->where('date_deleted IS NULL')
       ->where('date_closed IS NOT NULL')
       ->where('reporter_id = ?', $reporterId)
       ->limit($limit, $offset)
       ->order('date_created DESC');
```

Zend_Db_Select

- **Support for:**
  - Joins
  - ORDER
  - HAVING
  - GROUP
- **All select objects are specific to the current adapter**
  - Abstracts the final query according to the needs of the underlying database

How do I debug Zend_Db?

Use Zend_Db_Profiler

- Stores what queries have been run on the current adapter
- Stores the parameters used for prepared statements
- Has timing information for each query run

Zend_Db_Table

Why does Zend_Db_Table get its own section?

- For most DB use cases, Zend_Db_Table will be the best choice for database access
- Extensible architecture, allowing for custom business logic in many places
- OOP architecture makes all operations with rows and rowsets normalized

Zend_Db_Table Features

- **Table and Row Data Gateway components**
- **Similar to ActiveRecord, but lower level**
- **Provides ability to create, update, and delete rows**
- **Functionality for finding rows by ID or arbitrary criteria (using Zend_Db_Select)**
- **Rows are returned as Rowsets**
  - Iterable
  - Countable
  - Contain Row objects
- **Define table relations to other Zend_Db_Table classes**

Zend_Db_Table_Row

- **Maps fields to object properties (via overloading)**
- **Allows saving row after changes**

```
if ($link->relation_type != $linkType) {
    $link->relation_type = $linkType;
    $link->save();
}
```

Zend_Db_Table_Row

- **Poll parent or dependent row/rowsets (as defined in table relations):**

```
$user  = $comment->findParentRow('Model_Table_User');
```

```
$comments = $bug->findDependentRowset('Model_Table_Comment');
```

Extending Zend_Db_Table

Why?

- **Provide custom row or table classes with custom business logic**
- **Override methods in table class to ensure global behavior**

Fetching Row/sets via arbitrary criteria

- **Pass Select objects to fetch*() methods:**

```
$select->where('bug_id = ?', $originalBug)
       ->where('related_id = ?', $linkedBug);

$links = $table->fetchAll($select);
```

Joins are tricky

- **Rows cannot contain columns from other tables and still be writable**
- **setIntegrityCheck(false) allows you to retrieve these rows in read-only mode**

```
$select   = $bugTable->select()->setIntegrityCheck(false);
$select->from(array('b' => 'bug'))
       ->joinLeft(array('i' => 'issue_type'), 'i.id = b.type_id', array('issue_type' => 'type'))
       ->joinLeft(array('r' => 'resolution_type'), 'r.id = b.resolution_id', array('resolution'))
       ->joinLeft(array('p' => 'priority_type'), 'p.id = b.priority_id', array('priority'))
       ->where('date_deleted IS NULL'); // never fetch deleted bugs
```

Table Relations

- **Define for each table**
- ***Dependent tables* define how they relate to the parent**
- ***Parent tables* define what table classes are child tables**

Dependent table

- **A dependent table looks like this:**

```
class Model_Table_Comment extends Zend_Db_Table
{
    protected $_name    = 'comment';
    protected $_primary = 'id';

    protected $_referenceMap = array(
        'User' => array(
            'columns'        => 'user_id',
            'refTableClass' => 'Model_Table_User',
            'refColumns'     => 'id',
        ),
        'Bug' => array(
            'columns'        => 'bug_id',
            'refTableClass' => 'Model_Table_Bug',
            'refColumns'     => 'id',
        ),
    );
}
```

Parent table

- **The related parent table looks like this:**

```
class Model_Table_Bug extends Zend_Db_Table
{
    protected $_name    = 'bug';
    protected $_primary = 'id';

    protected $_dependentTables = array(
        'Model_Table_BugRelation',
        'Model_Table_Comment',
    );
```

Fetching the parent row

- **Given a row in the dependent table, fetch the parent row:**

```
$user  = $comment->findParentRow('Model_Table_User');
```

Fetching the child rows

- **Given a row in the parent table, fetch a rowset from the child table:**

```
$comments = $bug->findDependentRowset('Model_Table_Comment');
```

More on table relations

- **The above examples showed 1-many relations**
- **Many-to-Many are also possible via pivot tables**
- **Use joins when possible, as they are more performant**

Authentication

What is authentication?

- **Simply, is someone who they say they are?**

Authentication terminology

- **Credentials:**
  any information uniquely identifying someone

- **Identity:**
  information and metadata defining a person in the system

Okay, so how do I authenticate users?

- **Databases**
- **LDAP**
- **Web Services**
- **OpenId**
- **InfoCard**
- **HTTP Auth/Digest**
- **Filesystem**
- **...?**

What should I use?

- **Whatever suits your site's requirements**

Zend_Auth

- **Provides authentication adapters to utilize to verify credentials and retrieve identity**
- **Provides an interface for creating your own adapters**
- **Provides identity persistence**

Authentication adapters

- **Define a method, authenticate(), for verifying *credentials***
- **Define a method, getIdentity(), for retrieving an authenticated user's identity**
- **Current shipped adapters:**
  - DbTable
  - Ldap
  - OpenId
  - InfoCard
  - HTTP simple and digest

Zend_Auth Object

- **Authenticate a user based on the provided adapter**
- **Check for identity and retrieve identity**
- **Persist identity**
  - Via Zend_Session_Namespace

Example: DbTable Adapter

- **Verify credentials against stored records in a database table**
- **Provide constructor with:**
  - Table
  - Column specifying identity
  - Column specifying credentials
  - Optionally, credential “treatment” (such as MD5, extra conditions, etc.)

Example: DbTable Adapter

- **Example:**

```
$adapter = new Zend_Auth_Adapter_DbTable(
    Zend_Db_Table_Abstract::getDefaultAdapter(),
    'user',
    'username',
    'password',
    '? AND (date_banned IS NULL)'
);
$adapter->setIdentity($values['username']);
$adapter->setCredential(md5($values['password']));
```

Example: Identity Persistence

- **Check for identity:**

```
if ($auth->hasIdentity()) {
```

- **Retrieve identity:**

```
$identity = $auth->getIdentity();

$values = array(
    'user_id'    => $identity->id,
    'user_name'  => $identity->username,
    'user_email' => $identity->email,
);
```

- **Clear identity (logout!):**

```
Zend_Auth::getInstance()->clearIdentity();
```

Access Control Lists (ACLs)

What are Access Control Lists?

- **Check to see if a *role* has *rights* to a *resource*.**

ACL terminology

- **A *resource* is an object to which access is controlled**
- **A *role* is an object which may request access to a *resource***
- **A *right* is a specific access a *role* requests of a *resource*.**
- **Confused?**

ACL in plain language

- **We want *anonymous* users to be able to *list* and *view* any *bug***
- **A *bug* is a *resource***
- **“list” and “view” are *rights* on that *resource***
- **The *role* is *anonymous***

ACL in plain language

- **We want *registered users* to be able to *submit* new bugs and *comment* on existing *bugs*, in addition to all *rights* of *anonymous* users**
- **A *bug* is a *resource***
- **“submit” and “comment” are *rights* on that *resource***
- **The *role* is *registered user***

Roles

- **Implement Zend_Acl_Role_Interface, which has one method, getRoleId():**

```
class Bugapp_Acl_Role_Developer implements Zend_Acl_Role_Interface
{
    public function getRoleId()
    {
        return 'developer';
    }
}
```

Resources

- **Implement Zend_Acl_Resource_Interface, which has one method, getResourceId():**

```
class Bugapp_Acl_Resource_Bug implements Zend_Acl_Resource_Interface
{
    public function getResourceId()
    {
        return 'bug';
    }
}
```

Building ACLs

- **Instantiate the ACL object:**

```
$acl = new Zend_Acl();
```

- **Add one or more resources with add():**

```
$acl->add(new Bugapp_Acl_Resource_Bug);
```

- **Add one or more roles with addRole():**

```
$acl->addRole(new Bugapp_Acl_Role_Guest)
    ->addRole(new Bugapp_Acl_Role_User, 'guest')
    ->addRole(new Bugapp_Acl_Role_Developer, 'user')
    ->addRole(new Bugapp_Acl_Role_Manager, 'developer');
```

Building ACLs

- **Specify rules (rights):**

```
$acl->deny()
    ->allow('guest', 'bug', array('view', 'list', 'index'))
    ->allow('user', 'bug', array('comment', 'add', 'add-process'))
    ->allow('developer', 'bug', array('resolve'))
    ->allow('developer', 'bug', array('close', 'delete'));
```

Checking ACLs

- **Check if the resource exists in the ACLs:**

```
if ($acl->has('bug')) {
```

- **Check if the *role* has *rights* on the *resource*:**

```
if (!$acl->isAllowed($role, 'bug', $action)) {
```

Putting some things together

- **Create a plugin to check for identity and build ACLs:**

```
public function dispatchLoopStartup(Zend_Controller_Request_Abstract $request)
{
    $view   = Zend_Layout::getMvcInstance()->getView();
    $auth   = Zend_Auth::getInstance();
    $values = array(
        'user_id'    => null,
        'user_name'  => null,
        'user_email' => null,
    );
    if ($auth->hasIdentity()) {
        $identity = $auth->getIdentity();
        $values = array(
            'user_id'    => $identity->id,
            'user_name'  => $identity->username,
            'user_email' => $identity->email,
        );
        $role = empty($identity->role) ? 'user' : $identity->role;
    } else {
        $role = 'guest';
    }

    $view->assign($values);
    Zend_Registry::set('acl', $this->getAcl());
    Zend_Registry::set('role', $role);
}
```

Putting some things together

- **getAcl() implementation:**

```
public function getAcl()
{
    if (null === $this->_acl) {

        $acl = new Zend_Acl();

        $acl->add(new Bugapp_Acl_Resource_Bug);

        $acl->addRole(new Bugapp_Acl_Role_Guest)
            ->addRole(new Bugapp_Acl_Role_User, 'guest')
            ->addRole(new Bugapp_Acl_Role_Developer, 'user')
            ->addRole(new Bugapp_Acl_Role_Manager, 'developer');

        $acl->deny()
            ->allow('guest', 'bug', array('view', 'list', 'index'))
            ->allow('user', 'bug', array('comment', 'add', 'add-process'))
            ->allow('developer', 'bug', array('resolve'))
            ->allow('developer', 'bug', array('close', 'delete'));

        $this->_acl = $acl;
    }
    return $this->_acl;
}
```

Putting some things together

- **Create a view helper to retrieve a role:**

```
class Zend_View_Helper_GetRole
{
    public function getRole()
    {
        return Zend_Registry::get('role');
    }
}
```

Putting some things together

- **Create a view helper to check an ACL:**

```
class Zend_View_Helper_CheckAcl
{
    public $view;

    public function setView(Zend_View_Interface $view)
    {
        $this->view = $view;
    }

    public function checkAcl($resource, $right)
    {
        $acl  = Zend_Registry::get('acl');
        $role = $this->view->getRole();
        if (!$acl->has($resource)) {
            return true;
        }
        return $acl->isAllowed($role, $resource, $right);
    }
}
```

Putting some things together

- **Using the checkAcl() view helper:**

```
<?  if ($this->checkAcl('bug', 'comment'): ?>
<h3>Submit a comment:</h3>
<?= $this->commentForm ?>
<? endif ?>
```

Other components you'll use often

Zend_Config

- **For static configuration**
- **Typically defines the application environment**
  - Database adapter credentials
  - Caching options
  - Paths
- **Standard OOP interface to config files**
  - INI files
  - XML files
  - PHP arrays

Zend_Config

- **Allows pulling by section**

```
[development]
showExceptions = 1
phpSettings.display_errors = 1
db.adapter = "pdo_sqlite"
db.params.dbname = "../data/db/bugs.db"

[test : development]

[production : development]
showExceptions = 0
phpSettings.display_errors = 0
```

```
$config = new Zend_Config_Ini('../application/config/site.ini', 'development');
```

Zend_Config

- **Allows inheritance by section**

```
[development]
showExceptions = 1
phpSettings.display_errors = 1
db.adapter = "pdo_sqlite"
db.params.dbname = "../data/db/bugs.db"

[test : development]

[production : development]
showExceptions = 0
phpSettings.display_errors = 0
```

Things to know about Zend_Config

- **By default, read-only access**
- **Each instance can be marked mutable**

```
$config = new Zend_Config_Ini(
    '../application/config/site.ini',
    'development',
    array('allowModifications' => true)
);
```

- **Mutable config objects may be written to:**

```
$config->root = dirname(__FILE__) . '/../';
```

- **...but changes will not be saved.**

Zend_Registry

- **Static storage of objects and arbitrary key/value pairs for global access**
- **Used by some components internally**
  - Always uses class name as key
  - Zend_Translate
  - Zend_Auth

Internationalization

- **i18n = Internationalization**
  - Process of making an application locale aware
- **L10n = Localization**
  - Translation into regional language
  - Region-specific transformations of selected objects:
    - Weights
    - Distances
    - Temperatures
    - etc.

Lost in Translation

- **Translate strings or message identifiers to the target language**
- **Utilize one or more translation adapters:**
  - PHP Arrays
  - XML
  - Gettext
  - QT
  - More...

Translation aware components

- **Zend_View_Helper_Translate: <?= $this->translate('something') ?>**
- **Zend_Form**
  - Error messages
  - Labels
  - Descriptions
  - Any metadata which a decorator decides is translatable
- **Register the Zend_Translate object with Zend_Registry as the 'Zend_Translate' key to automate the process:**

```
Zend_Registry::set('Zend_Translate', $translate);
```

Search

Search is really, really hard to get right

- **Fulltext searches are often slow, and limit table options**
- **Difficult to score hits**
- **Difficult to build multi-field queries**
- **Difficult to build queries in general**

Enter... Zend_Search_Lucene

- **Binary compatible with Apache Lucene**
- **Tokenizes documents and stores them in binary indices**
- **Each document can consist of one or more arbitrary fields, allowing for document-specific searches**
- **Can build queries programmatically**
- **...or have Z_S_L parse a query string for you**
- **Indexes may be updated**

Adding a document to an index

```
$doc = new Zend_Search_Lucene_Document();
$doc->addField(Zend_Search_Lucene_Field::Keyword('listing_id', $id));
$doc->addField(Zend_Search_Lucene_Field::Text('title', $listing->title));
$doc->addField(Zend_Search_Lucene_Field::Text('year_built', $listing->year_built));
$doc->addField(Zend_Search_Lucene_Field::Text('asking_price', $listing->asking_price));
$doc->addField(Zend_Search_Lucene_Field::Text('number_bedrooms', $listing->number_bedrooms));
$doc->addField(Zend_Search_Lucene_Field::Text('number_baths', $listing->number_baths));
$doc->addField(Zend_Search_Lucene_Field::Text('square_feet', $listing->square_feet));
$doc->addField(Zend_Search_Lucene_Field::Text('acres', $listing->acres));
$doc->addField(Zend_Search_Lucene_Field::Text('short_description', $listing->short_description
$doc->addField(Zend_Search_Lucene_Field::Text('long_description', $listing->long_description))
```

Querying the index

```
$query  = Zend_Search_Lucene_Search_QueryParser::parse($search);
$hits   = $index->find($query);
foreach ($hits as $hit) {
    echo $hit->listing_id, ': ', $hit->title, "\n";
}
```

Topics we didn't cover...

- **Consuming and creating web services**
- **Caching with Zend_Cache**
- **Serving alternate content types from MVC applications**
- **Testing models and applications**
- **And even more**

Where do I go for help?

- **Manual: http://framework.zend.com/manual**
- **Issue Tracker: http://framework.zend.com/issues**
- **Mailing Lists: http://framework.zend.com/archives**
- **IRC: #zftalk on freenode.net**

Where can I learn more?

Blogs:

- **Rob Allen: http://akrabat.com/**
- **Padraic Brady: http://blog.astrumfutura.com**
- **My own: http://weierophinney.net/matthew/**
- **DevZone: http://devzone.zend.com/tag/Zend%20Framework**

How can I contribute?

- **http://framework.zend.com/community/contribute**
- **Sign a CLA**
- **Submit Bugs**
- **Propose new components**
- **Fix Bugs**
- **Write or translate documentation**

Thank you!