



Zend Framework's MVC Components

Matthew Weier O'Phinney
PHP Developer
Zend Technologies

Zend Framework provides rich and flexible MVC components built using the object-oriented features of PHP 5.



Topics Overview

- Zend Framework Overview
- What is MVC?
- Zend_Controller: The 'C' in MVC
- Zend_View: The 'V' in MVC
- Zend_... Where's the 'M'?
- Putting it Together
- Q & A

Zend Framework Overview

What is Zend Framework?

- **Component Library**
 - Loosely coupled components for general purpose actions
 - Use-at-will architecture
- **Application Framework**
 - Cohesive framework for building applications

Zend Framework Goals

Extreme Simplicity:

- **Simpler is easier to use**
- **Simpler is more stable and less prone to error**
- **Simpler is easier to maintain**

Zend Framework Goals

Showcase Current Trends:

- Web Services
- Ajax
- Search

Zend Framework Goals

Stability and Documentation

- All components must have > 80% test coverage
- All components must have end-user documentation and use-cases

Zend Framework Goals

Business Friendly

- Contributor License Agreement required in order to contribute code, patches, or documentation
- All code licensed under the new BSD license

What is MVC?

MVC Overview

- **Model**
The "stuff" you are using in the application -- data, web services, feeds, etc.
- **View**
The display returned to the user.
- **Controller**
Manages the request environment, and determines what happens.

MVC Interactions

- **Controller <-> View**
Controller and View can interact
- **Controller <-> Model**
Controller can pull data from the model for decisioning, or push data to the model
- **View <- Model**
View can access the model to retrieve data, but not write to it.

Front Controller

- **Handles all requests**
- **Delegates requests to 'Action Controllers' for handling**
- **Returns response**

Zend_Controller: The 'C' in MVC

Zend_Controller: Basics

Action Controllers:

- **Extend Zend_Controller_Action**
- **Class name ends in 'Controller'**
 - IndexController
 - BlogController
- **Underscores indicate directory separators**
 - Foo_AdminController => Foo/AdminController.php
 - Note: rule is different with modules
- **CamelCasing allowed**
 - FooBarController
 - Separate CamelCased words in URLs with '-' or '.':
 - foo-bar
 - foo.bar

Zend_Controller: Basics

Controller Actions:

- **Method the action controller should perform**
- **Public methods ending in 'Action'**
 - barAction()
 - indexAction()
- **CamelCasing allowed**
 - fooBarAction()
 - Separate camelCased words on the URL with '.', '-', or '_':
 - foo-bar
 - foo.bar
 - foo_bar

Zend_Controller: Basics

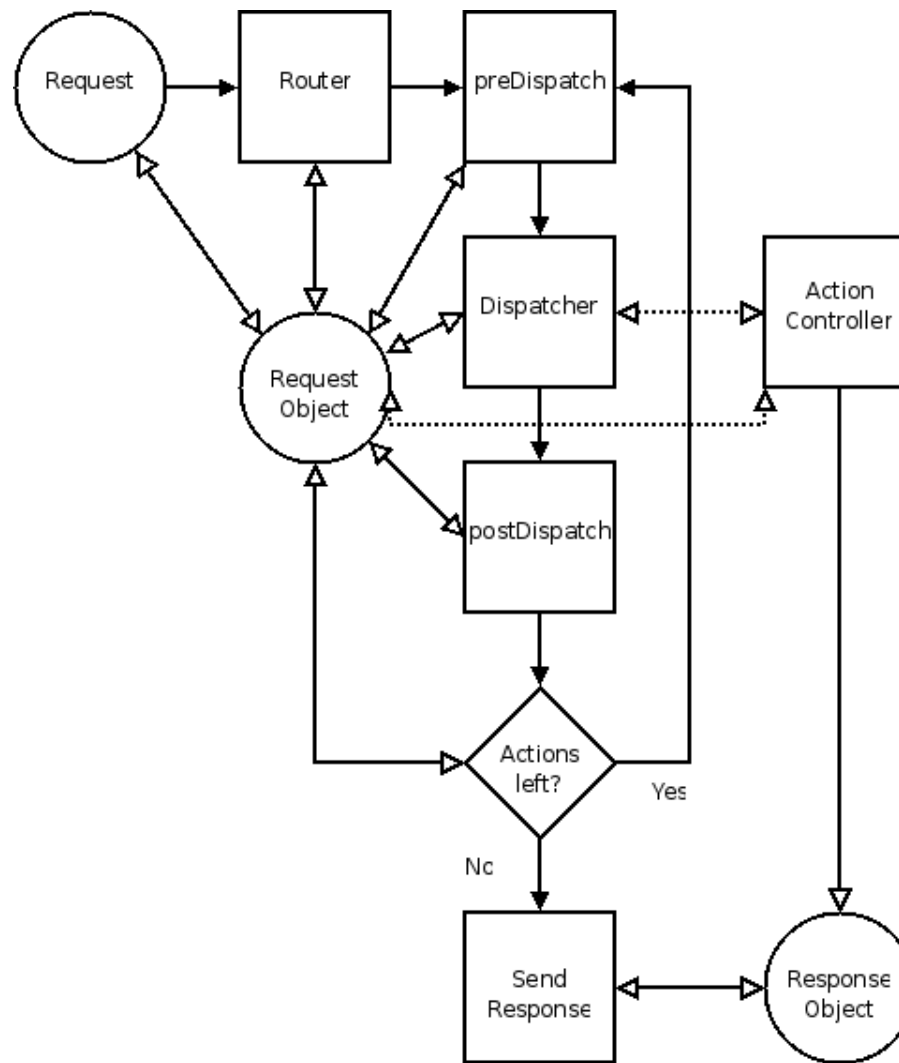
Modules:

- **A set of related action controllers, models, and views**
- **Directory structure mimics application directory structure:**
 - controllers/
 - models/
 - views/
- **Controller class names should be prefixed with module name:**
 - Foo_ViewController ->
foo/controllers/ViewController.php
- **Module names may be camelCased as well; follow rules for controllers**

Zend_Controller: Responsibilities

- **Request object: contains all information on the request environment**
- **Router: decomposes environment into various tokens representing the current module, controller, action, etc.**
- **Dispatcher: maps the tokens from routing to action controller classes and methods, and executes them**
- **Response object: contains the complete response and has the ability to send it**

Zend_Controller: Process Diagram



Zend_Controller: Dispatch Loop

- **\$front->dispatch()** handles the incoming request
- **Instantiates request and response objects if not previously set**
- **Routes request**
- **Enters dispatch loop**
 - Dispatch Action
 - Instantiate controller
 - Call action method
 - Dispatches until request object reports no more actions to dispatch
- **Returns Response (sends by default)**

Zend_Controller: Routing

Default Routing:

- `/controller/action`
- `/controller/action/key1/value1/key2/value2`
- `/module/controller/action`
- `/module/controller/action/key1/value1/...`

Zend_Controller: Routing

Modifying Routing: Rewrite Router:

- **Zend_Controller_Router_Rewrite is the default router implementation**
- **Allows attaching as many named routes as desired**
 - Named routes allow pulling routes for later operations, such as URL assembly or determining what in a URL matched.
- **Routes are executed in a LIFO order**
- **Route interface allows defining your own route types for your applications**

Zend_Controller: Routing

Shipped Route Types:

- **Static** -- match exactly, and dispatch according to defaults
 - Fastest route; straight equality comparison
- **Standard** -- matches by named URL segments
 - Flexible and readable, easiest creation of dynamic routes. However, each URL segment is potentially compared against a regexp, making it slow.
- **Regex** -- matches using PCRE
 - Fastest and most flexible dynamic route, but potentially the hardest to maintain if not all developers are equally versed in PCRE.

Zend_Controller: Routing

Creating a new route:

- Want to match this:
/news/view/12
- Route: /news/view/:id

```
$route = new Zend_Controller_Router_Route(
    'news/view/:id', // route expression
    // Now set defaults:
    array(
        'module'      => 'default',
        'controller' => 'news',
        'action'      => 'view'
    ),
    // and set a requirement for the 'id' param:
    array(
        'id' => '\d+', // only allow 1 or more digits
    )
);

// Attach to router:
$router->addRoute('newsItem', $route);
```

Zend_Controller: Action Controllers

Action Controllers

- **Simply classes that extend `Zend_Controller_Action`**
- **Define public action methods for each action you want the controller to handle**
- **Use regular public methods when you want to have re-usable or testable functionality**

Zend_Controller: Action Controllers

Action controller triggers and listens to the following events:

- **init()** -- *object instantiation*
- **preDispatch()** -- *prior to dispatching the action*
- **postDispatch()** -- *after the action has executed*

Zend_Controller: Action Controllers

Utility Methods:

- **`_forward($action, $controller = null, $module = null, array $params = null)`** -- *forward to another action*
- **`_redirect($url)`** -- *redirect to another location*
- **`render($action, $name, $noController)`** – *render an alternate view script*
- **`__call($method, $params)`** -- *use to create 'dynamic' actions or internally forward to a default action*

Zend_Controller: ViewRenderer

- View integration is automatically available
- Registered by *ViewRenderer* action helper
- Can be disabled
- \$view property of controller contains view object
- Assign variables to view:
`$this->view->model = $model;`

Zend_Controller: ViewRenderer

- **View scripts are rendered automatically during `postDispatch()` event**
- **View scripts named after controller and action:**
 - `FooController::barAction()` renders `foo/bar.phtml`
 - `NewsController::listAction()` renders `news/list.phtml`
- **Disabling the ViewRenderer**
 - `setNoRender()` will disable it for the current action
 - Calling `_forward()` or `_redirect()` never auto-renders

Zend_Controller: ViewRenderer

Customizing the ViewRenderer:

- **setView()**
 - Set view object (allows for custom view implementations!)
- **setViewSuffix()**
 - Change the file suffix used
- **setView(Base | Script)PathSpec()**
 - Set the path specification used for auto-determining the view location
- **setResponseSegment()**
 - Set the named response segment to render into

Zend_Controller

Sample Action Controller:

```
class FooController extends Zend_Controller_Action
{
    // simply render a view
    public function indexAction()
    {}

    public function barAction()
    {
        $this->_forward('baz'); // forward to baz action
    }

    public function bazAction()
    {
        // Assign view variables
        $this->view->message = 'In /foo/baz';
    }
}
```

Zend_Controller: Plugins

What are Plugins?

- Triggered by front controller events
- Events bookend each major process of the front controller
- Allow automating actions that apply globally

Zend_Controller: Plugins

Events:

- **routeStartup()** -- *prior to routing*
- **routeShutdown()** -- *after routing*
- **dispatchLoopStartup()** -- *prior to fist iteration of dispatch loop*
 - *preDispatch()* -- *prior to dispatching an action*
 - *postDispatch()* -- *after dispatching an action*
- **dispatchLoopShutdown()** -- *at dispatch loop termination*

Zend_Controller: Plugins

Creating Plugins:

- **Extend Zend_Controller_Plugin_Abstract**
- **Extend one or more of the event methods**
 - Create multi-purpose plugins by extending multiple methods
 - Create targetted plugins by extending a single method

Zend_Controller: Plugins

Example: Two-Step View Plugin

```
class TwoStepView extends Zend_Controller_Plugin_Abstract
{
    public function dispatchLoopShutdown()
    {
        $view = Zend_Controller_Action_HelperBroker::getStaticHelper('viewRenderer');
        if (empty($view)) {
            return;
        }

        $response = $this->getResponse();
        $content = $response->getBody();
        $view->content = $content;

        $response->setBody($view->render('site.phtml'));
    }
}
```

Zend_Controller: Action Helpers

What are Action Helpers?

- Reusable functionality
- Functionality that can be used in multiple controllers
- Functionality you want to be able to discretely unit test
- Objects you wish to persist across controllers
- Useful for automating processes that involve the action controllers
- Initialized on-demand, or may be registered with helper broker

Zend_Controller: Action Helpers

Creating Action Helpers:

- **Extend *Zend_Controller_Action_Helper_Abstract***
- **Last segment of class name is helper name**
 - `My_Helper_Foo` -> 'foo' helper
 - `My_Helper_FooBar` -> 'fooBar' helper
- **Optionally implement a `direct()` method for method-like invocation**
 - Allows helper to be called as if it were a method of the helper broker

Zend_Controller: Action Helpers

Using Action Helpers as Action Controller Event Listeners:

- **init():**
 - when the action controller is initialized
- **preDispatch():**
 - executes after front controller preDispatch() plugins but before action controller preDispatch
- **postDispatch()**
 - *executes after action controller postDispatch() but before front controller postDispatch() plugins*
- **Note: helper must be registered with broker for events to trigger**

Zend_View: The 'V' in MVC

Zend_View: Overview

- Implement *Zend_View_Interface* to create your own template engine
- Default implementation (*Zend_View*) uses PHP as the template language
- Assign and retrieve view variables as if they were object members:
`$view->content = $body`
- Access view variables in view scripts from *\$this* object:
`<?= $this->content ?>`
- Benefits: All of PHP is at your disposal
- Issues: All of PHP is at your disposal

Zend_View: View Scripts

- **Mix HTML and PHP**
- **Access template variables using *\$this* notation**
- **Keeps assigned variables in their own scope**
- **Easily distinguish assigned variables from local variables**
- **Easy placeholder implementation: simply assign from view scripts and use in later view scripts**

Zend_View: View Scripts

- Use PHP short tags for shorthand notation:

```
<?= $this->content ?>  
<? if ($this->foo): ?>  
    foo  
<? endif ?>
```

Zend_View: View Helpers

- **Classes that extend the functionality of Zend_View**
- **Uses**
 - Access models (e.g. add a del.icio.us feed to your page)
 - Format or escape output (e.g. transform wiki text to XHTML)
 - Display logic (e.g., show login buttons if user not logged in)
 - Re-usable display snippets (e.g., search form box)

Zend_View: View Helpers

Using View Helpers:

- Call as if the helper were a method of the view object

```
<?= $this->formText('username') ?>
```

Zend_View: View Helpers

Creating and Using View Helper:

- **Helper name is last segment of class name**
 - My_View_Helpers_Foo: foo helper
 - My_View_Helpers_FooBar: fooBar helper
- **Register helper paths with Zend_View object**
 - Optionally specify a class prefix
 - Paths searched in LIFO order
 - Override a helper by registering late

Zend_View: View Helpers

View Helper Classes:

- **Must have a method named after the helper:**

```
class My_View_Helpers_Foo
{
    public function foo() {}
}

class My_View_Helpers_FooBar
{
    public function fooBar() {}
}
```

Zend_View: View Helpers

- Optionally allow view awareness by creating a `setView()` method:

```
class My_View_Helpers_Foo
{
    public $view;

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

Zend_View: Filters

- **Allow filtering rendered content prior to returning it**
- **Similar to helpers, one class and method per filter**
- **Use Cases**
 - Transform HTML to PDF
 - Transform HTML to JSON
 - Pass X/HTML through tidy
 - Inject session IDs

*Zend_...:
Where's the 'M'?*

Zend_Model?

What is a Model?

- Database
- Web Services
- Feeds
- Configuration files
- Filesystem
- Images

Zend_Model?

How does Zend Framework address the Model?

- **We don't, at least not as a generalized component.**
- **But we do support it with our specialized components:**
 - Zend_Db_Table
 - Zend_Service
 - Zend_Feed
 - etc.

Putting it Together

Putting it Together

Filesystem Layout:

```
application/  
  controllers/  
    IndexController.php  
    ErrorController.php  
  models/  
  views/  
    scripts/  
      index/  
        index.phtml  
      error/  
        error.phtml  
      <controller>/  
        <action>.phtml  
    helpers/  
html/  
  .htaccess  
  index.php  
  images/  
  styles/  
  javascript/
```

Putting it Together

The Bootstrap file (index.php): Simplest:

```
require_once 'Zend/Controller/Front.php';  
Zend_Controller_Front::run('/path/to/controllers');
```

Putting it Together

Bootstrap File (index.php): More Advanced:

```
require_once 'Zend/Controller/Front.php';
$front = Zend_Controller_Front::getInstance();

// Register some action helpers by class prefix;
// it then assumes they're in My/Helpers/*.php
Zend_Controller_Action_HelperBroker::addPrefix('My_Helpers');

// When using modules, group them under a common directory, and this
// method will register their 'controllers' directories with the
// front controller:
$front->setModuleDirectory('/path/to/application/modules')
    ->registerPlugin(new TwoStepView()); // add a plugin!

// Dispatch the front controller!
$front->dispatch();
```

Putting it Together

Hello World! IndexController:

```
class IndexController extends Zend_Controller_Action
{
    public function indexAction()
    {
    }
}
```

Putting it Together

Hello World! 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:
                // Page not found (404) error
                $this->render('404');
                break;
            default:
                // Application (500) error
                $this->render('500');
                break;
        }
    }
}
```

Putting it Together

- **Hello World! View scripts:**

```
<? // index/index.phtml ?>
Hello world!

<? // error/404.phtml ?>
Page not found!

<? // error/500.phtml ?>
An error in occurred in this application!
```



Thank you!

Zend/PHP Conference & Expo 2007
<http://www.zendcon.com>

More on Zend Framework:
<http://framework.zend.com>