



The PHP Company



Participant Course Guide



Zend Framework 2: Fundamentals

Page left blank for online viewing

Introduction

ZEND FRAMEWORK2: FUNDAMENTALS COURSE

The Zend Framework 2 (ZF2) Fundamentals course is designed for experienced PHP programmers who want to learn to combine ZF2 concepts and structural elements to utilize the full power of this software development kit for PHP applications. Zend Framework 2 is an open-source class library with a central theme of “extreme simplicity”. ZF2 helps to reduce the tedious details of coding and instead allows developers to be more productive as they focus on the overall design. Utilizing a collection of customizable PHP classes, ZF2 provides robust functionality suitable for both large and small tasks. This course combines teaching ZF2 with a Test-Driven Development (TDD) approach to building applications, a best practice in PHP app creation. In the Zend Framework:2 Fundamentals course, you learn by doing. Discussions of related components are presented with examples of how best to utilize them in your apps. Hands-on exercises reinforce concepts as you build an online messaging application in the Cloud.

Audience:

This course is designed for people who have a solid foundation in the PHP language and OOP, and have successfully created complex PHP web applications.

Pre-requisite(s):

Solid understanding of PHP at an intermediate+ level and OOP experience

Objectives:

Upon completing the course, active PHP developers should be fully-equipped to apply a best practice approach to web application development, using ZF2 as the framework and TDD as the approach.

About Zend

Zend is the PHP company. Businesses utilizing PHP know Zend as the place to go for PHP expertise and sound technology solutions. Zend delivers premier web application platform products and services for PHP applications. With commercial products and services that enable developers and IT personnel to deliver business-critical PHP applications, Zend is taking the power of PHP to the enterprise.

Note:

While this is a fundamental-level course on Zend Framework 2, it requires more than a fundamental level understanding of PHP. Participants are assumed to be proficient enough in the language to understand intermediate-level code examples and to perform coding exercises.

This course assumes no previous ZF2 experience.

Zend provides a suite of products that supports the entire PHP lifecycle, from development to production, of your business-critical PHP applications. Often, PHP programmers work in demanding environments, where they are called upon to fulfill more than one role. Zend products allow those who work in web application development - as well as related fields like system administration - to seamlessly utilize all their relevant features to rapidly produce a dynamic, robust web application in a stable, reliable environment.

Zend Server

Setting the Standard in PHP Application Performance & Availability

Zend Server is the all-in-one production environment that ensures your PHP applications are available, fast, reliable and scalable. It uniquely guarantees application uptime and reliability through enhanced PHP monitoring and immediate problem resolution.

Zend Studio

The Proven PHP Development Environment

Zend Studio is the leading PHP Integrated Development Environment (IDE) designed for professional developers, which includes all the development components necessary for the full PHP application lifecycle.

Zend Guard

Protect your Code and IP

Zend Guard provides independent software vendors and IT managers with the ability to safely distribute and manage the distribution of their PHP applications while protecting their source code.

Zend Optimizer

Optimize your PHP code

Zend Optimizer is a free application that runs files encoded using Zend Guard and enhances the overall performance of your PHP applications.

Zend Framework 2

Build your apps faster and reduce maintenance costs

Zend Framework 2 embodies extreme simplicity & productivity, the latest Web 2.0 features, simple corporate-friendly licensing, and an agile well-tested code base that your enterprise can depend on.

Zend Engine

The Heart of PHP

At the center of PHP is the Zend Engine, the component that parses and executes PHP files. It is open source software and available under an Apache-style license.

Slide 1



The PHP Company

ZEND FRAMEWORK 2: FUNDAMENTALS

Copyright © 2006-2013, Zend Technologies Inc.

Slide 2



“ Module One..

Copyright ©2006-2013, Zend Technologies Inc.

Slide 3



... Introduction to the Course ...

Copyright ©2006-2013, Zend Technologies Inc.

Slide 4

Course Goals

- In this course, we will study Zend Framework 2 (ZF2) so you will understand...
 - the concept behind the framework
 - the structural components that make up the framework
 - the power of combining these elements for a complete design solution



The PHP Company

Copyright © 2006-2013, Zend Technologies, Inc.

| 4

Course Approach

- **The course approach will be guided learning, with more and more of the coding to be completed by you**
- **The key emphasis is not on what Zend Framework 2 *is*, but how best to use it when building applications**
- **Topics will be addressed at the beginner-level Framework user, leading to an intermediate-level upon completion**
- **Topic Sequence, by module:**

1: Intro & Essential Concepts	6: Controllers and Plugins
2: MVC Introduction	7: Routing
3: Event Manager	8: View Layer
4: Service Manager	9: Forms & Filters & Validators
5: MVC in Detail	10. Databases (Zend\Db)
	11. Sessions, and Optionally, Logging & Mail



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

| 5

Slide 6

Course Applications

There are two applications used for the course:

- There is a **teaching application**, a **Guest Book**, which provides code examples for the important concepts presented in the course
- There is also a **practice application**, an **Online Market**, which you will use to complete the **course exercises**. It is a working application, which will provide you with practical experience in using ZF2 for web application development and reinforce the concepts taught
- Practice exercises include:
 - Installing ZF2, including the Zend Skeleton App
 - Working with Event Manager to attach an Event Handler to a Dispatch Event
 - Defining Services, including "Registration", "Invokable", and "Factory"
 - Creating a Module, and learning how to integrate an existing Module
 - Creating MVC Controllers, configuring Actions, and creating related Routing configs
 - Working with View Models and View Helpers
 - Working with Forms to post data and insert that data into a Database



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

| 6

- Both of these course applications have been created specifically for this course

Slide 7



... Introduction to ZF2

Copyright ©2006-2013, Zend Technologies Inc.

Overview of Zend Framework 2

- Zend Framework (ZF2) is an open-source framework for developing web applications and services with PHP
- ZF2 is implemented using 100% object-oriented code
- The component structure incorporates an "use-at-will" design...
 - Each component has few dependencies on others, resulting in a loosely coupled architecture that provides developers with maximum flexibility
 - When coupled together into an application framework, the components in the MVC layer create a powerful and extensible web application
- ZF2 offers:
 - A robust, high performance MVC implementation
 - A series of form-related components, which implement HTML form rendering, validation, and filtering so that developers can consolidate all of these operations using one easy-to-use, object-oriented interface



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

| 8

- Keep PHP competitive with other technologies like Perl, frameworks like CodeIgniter, Cake PHP, ...
- Provide "clean" IP to enable commercial use - real companies can't just "borrow" code from the Internet without clear licensing
- Exploit the power of PHP 5.3 to its full advantage
- The underlying technologies may be extremely complex, but the interface - the ZF components - is designed for ease of use
- 'Use-At-Will' Architecture - utilize individual components without being forced to use everything
- Based on the Pareto Principle - 80/20 Rule
 - Works for the most common use cases out of the box
 - Built for extensibility
- ZF2 is licensed using the BSD license
 - Anyone can use it, for anything, no strings attached – period
 - Business friendly

Slide 9

ZF2 Key Features

Key Features:

- Uses **Git and GitHub** for code contributions
 - Each contribution is reviewed first before pushed; commentary displayed...**Quality check**
- Uses **BSD license**... **completely open process**
- Requires **PHP 5.3.3 or greater**... **takes advantage of latest PHP features**
- Can install **only desired components**, not entire framework... **as light as you need it**
 - Diverse methods for installation: pyrus, composer
- Two Core Architectures: **Service Manager** and **Event Manager**



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

9

- ZF1 User Alert:
 - Now uses Git and GitHub instead of Subversion for code contributions
 - ZF2 still uses a BSD license but no longer requires a CLA for contributions

Slide 10

Any ZF1 Users?

Zf1 vs. ZF2: A new core, a new concept

ZF1	ZF2
Singletons	Event-Driven Services
Registries	
Hard- & Soft-Coded Dependencies	

- ZF2 resembles ZF1 on the surface in many parts, but structurally it works very differently – it is based on an event-driven architecture

... offers extensive flexibility and customization



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

10

ZF2 Architecture

What is the ZF2 Architecture?

- It utilizes very common design patterns
 - Subject/Observer, PubSub, Decorator, ... use the best design for the purpose
- Takes advantage of PHP 5.3.x features
 - Namespaces allow for co-existence of ZF1 and ZF2 code
 - Lambda Functions & Closures good for callbacks
 - Late Static Binding vary static behavior in extensions
- It uses several methodologies for development:
 - Decoupling inject instead of hard-coding
 - Events - EventManager allows you to shape the application workflow
 - Stdlib - Standard classes and SPL extensions provides low level functionality such as priority queuing and "hydration" of objects



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

11

Methodologies Used in Development

- Decoupling/ Inversion of Control
 - Instead of hard-coding dependencies, inject them
 - Inversion of control containers automate this
 - ServiceManager
 - SM is explicit and easy to understand
- Event Manager:
 - Uses events for decoupling
 - Provides benefits of aspect-oriented programming
 - Concept of message bus, which allows you to shape workflow wherever needed

ZF2 Architecture

What is the ZF2 Architecture?

- It utilizes very common design patterns
 - Subject/Observer, PubSub, Decorator, ... use the best design for the purpose
- Takes advantage of PHP 5.3.x features
 - Namespaces allow for co-existence of ZF1 and ZF2 code
 - Lambda Functions & Closures good for callbacks
 - Late Static Binding vary static behavior in extensions
- It uses several methodologies for development:
 - Decoupling inject instead of hard-coding
 - Events - EventManager allows you to shape the application workflow
 - Stdlib - Standard classes and SPL extensions provides low level functionality such as priority queuing and "hydration" of objects

The PHP Company Copyright © 2006-2013, Zend Technologies Inc. 11

- SPL extensions:
 - Used whenever possible
 - Designated area is Stdlib, although it contains more than SPL extensions
 - Dispatchable interface, for example, is mainly in MVC right now, but will be used for Queuing and Servers later
- Namespaces:
 - Can use ZF1 and ZF2 code in same app because of namespaces... no conflicting names
 - Namespaces make it easier to segregate code and create hierarchies (used in file systems because it makes it easier to find code)
 - Will make migration from ZF1 to ZF2 easier
- Lambda Functions & Closures:
 - Not common in PHP, but often used in JavaScript, whenever configuring an app
 - Most often used with app code, not core; ex: when need a callback, don't want to have to define a class... just define callback in place and pass it in
- Late Static Binding:
 - Mostly used with Active Record – another way to make extension points
 - Classes differ in behavior simply by extension, with respect to their static members

Slide 12

How ZF2 Works

The basic concept behind ZF2 is quite simple. It is based on two key managers - Event Manager and Service Manager - and listeners

Event Manager: Trigger events & Attaches/Detaches listeners

Listeners: Attach to specific events and listen for triggers

Service Manager: Provides the wiring that guides the app workflow



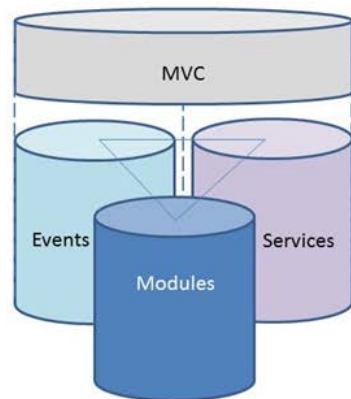
The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

12

Slide 13

ZF2 Architecture and MVC



The PHP Company

Copyright © 2006-2013, Zend Technologies, Inc. | 13

Slide 14



... Essential Concepts

Copyright ©2006-2013, Zend Technologies Inc.

Slide 15

Decoupling...

- **Technique used to improve testability of code, as well as to make maintenance easier**
 - Each class does only one thing and composes other objects when something lies outside its own area of responsibility
- **Favors composition over inheritance**
 - Alternative to extending a class more and more
 - Only way to alter behavior when instantiating an object directly within a class is via extension or configuration
 - Using injection allows you to decide how the interactions should work
 - Composition allows you to inject into them for different behavior
- **Provides Control**
 - Need objects and all their dependencies
 - Grab out of control container
- **Zend\ServiceManager facilitates creation of objects that compose dependencies**



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

15

- Issue with injections and applications, in general, is that you often need more than object injections... also need configuration injections (ex: HTTP client to specify which adapter or port to use; all scalar data)
- Decoupling: removes hard-coding and uses code pieces that can be used in many places, by injection; ex: given a common plug, anything that fits that plug may be used

Namespaces

What are Namespaces?

Namespaces are a method of grouping related PHP code elements within a library or application

Why would you use namespaces?

Using them helps to avoid two overall issues:

- **Accidentally re-defining** functions, classes, or constants that have already been defined
 - Common issue as code libraries grow, or if 3rd party code libraries are used
- Having to use **long, highly descriptive class names**



The PHP Company

Copyright © 2006-2013, Zend Technologies, Inc.

7-16

- You can have multiple namespaces per file, but it is *not recommended*
- A namespace may contain constants, functions, and classes
- Example from php.net:

```
<?php
namespace my\name; // see "Defining Namespaces" section
class MyClass {}
function myfunction() {}
const MYCONST = 1;
$a = new MyClass();
$c = new \my\name\MyClass; // see "Global Space" section
$a = strlen('hi'); // see "Using namespaces: fallback to global
                    // function/constant" section
$d = namespace\MYCONST; // see "namespace operator and __NAMESPACE__
                        // constant" section
$d = __NAMESPACE__ . '\MYCONST';
echo constant($d); // see "Namespaces and dynamic language features" section
?>
```

How Do Namespaces Work?

- You must declare the use of namespaces
 - Use the keyword "namespace" at the immediate start of your code file
 - Next line after <?php
 - Only one construct can precede the namespace declaration – "declare"
 - Only use one namespace per code file
 - You can declare more than one namespace within a file, but is not a good practice, especially for newer programmers... it is tricky to do correctly...
 - Better to use the "one file, once class" approach



The PHP Company

Copyright © 2006-2013, Zend Technologies, Inc. **7-17**

- You cannot nest or define more than one namespace within a single code BLOCK
- You can, however, define more than one within a single code FILE (but it is not a good practice, especially when starting out as a programmer – it can get tricky to use)

Slide 18

How Do Namespaces Work?

- Three code elements are directly affected by the use of namespaces:

- Constants
- Classes
- Functions

```
<?php
Namespace CalendarApp;
const HOURS= 24;
function day() {}
class week
{
    static function
staticmethod() {}
}
?>
```

- Classes and functions are contained within the global space unless a namespace is defined...
 - Prefix with "\ " to indicate a fully qualified (global) namespace



The PHP Company

Copyright © 2006-2013, Zend Technologies, Inc. **7-18**

How Do Namespaces Work?

- Once you define code elements within a namespace they can be utilized within other php files
 - The same namespace can be utilized in multiple files
 - Import namespaces with the "use" operator
- You can sub-divide the library by creating sub-namespaces
 - These are indicated using a "\"
 - Example: `use app\lib1`
- You can create an alias for a namespace (or a class)
 - Example for Namespace alias: `use app\lib1 as L1;`
 - Example for Class alias: `use Zend\Db\Adapter\Driver\Mysqli\Mysqli as MysqliDriver`



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 7-19

Slide 20

Namespaces Structure Example

```
<?php
namespace X;
class Abc
{
    public $abc = 'X';
    public function getAbc()
    {
        return $this->abc;
    }
}
```

```
<?php
namespace Y;
class Abc
{
    public $abc = 'Y';
    public function getAbc()
    {
        return $this->abc;
    }
}
```

```
<?php
namespace Z;

// NOTE: "use" allows
// relative reference to the namespace
use X\Abc;
$x = new Abc();

// outputs 'X'
echo $x->getAbc();

// otherwise we need an
// absolute reference to the namespace
$y = new \Y\Abc();

// outputs 'Y'
echo $y->getAbc();
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 20

Slide 21

Autoloading

- Zend\Loader is a namespace that includes various autoloading-related components, such as:
 - AutoloaderFactory
 - ClassMapAutoloader
 - StandardAutoloader



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 21

- Note: AutoloaderFactory is a factory
- There is also a ModuleAutoLoader, that is consumed by a listener on the ModuleManager (Zend\ModuleManager\ModuleManager) in order to resolve module names to their associated Module classes
 - Typically, you will never interact with this autoloader directly

Slide 22

AutoloaderFactory

What is the AutoLoaderFactory?

- Zend\Loader\AutoLoaderFactory **is a factory that instantiates, configures, and registers autoloaders that implement SplAutoloader**
- You pass it key/value pairs of `autoloader classname => options`

Why use the AutoLoaderFactory?

If the same autoloader is provided multiple times, it simply **configures the existing instance**, which helps **minimize the number of autoloaders** you have at any given time

This is primarily **used internally in a module listener**, but it's important to know how to configure it...

- Modules should return configuration for autoloading, and that configuration will be fed to the AutoloaderFactory



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 22

Slide 23

ClassMapAutoloader

What is the ClassMapAutoloader?

- The `ClassMapAutoloader` allows you to provide `classname => filename` pairs, where the filenames are either absolute or directly resolvable off the `include_path`
- Typically use `__DIR__ . '/path/to/ClassFile.php'` when defining them
 - Define the classmap relative to where the classes live in the filesystem; usually the classes will be at the same level or lower than the classmap itself
- This is a **high performance way to autoload classes**, and does not require any specific naming conventions
- Add bullet: Use `classmap_generator.php` in the `bin` folder of your ZF2 distribution to automatically generate an `autoload_classmap.php` file



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 23

Slide 24

StandardAutoloader

What is a StandardAutoloader?

- StandardAutoloader is a PSR-0 compliant autoloader that assumes a 1:1 map between the class name and the filesystem
- You typically seed it with namespace/vendor prefix => directory pairs
 - Using namespace/directory pairs is optimal for performance
- It can also be used as a "fallback autoloader" that tries to load PSR-0 compliant classes from the `include_path`
 - PSR-0 is a naming convention that uses a 1:1 relationship between the namespace hierarchy and class name, and the filesystem



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 24

- For information on PSR-0, see
<https://github.com/php-fig/fig-standards/blob/master/accepted/PSR-0.md>

Slide 1



The PHP Company

“ Module Two

Copyright © 2006-2013, Zend Technologies Inc.

Slide 2

Module Two Topics

- In this module, we will cover:
 - Design Patterns
 - Model-View-Controller (MVC) Design Pattern Introduction



The PHP Company

Copyright © 2006-2013, Zend Technologies, Inc. | 2

Slide 3

Design Patterns

- Some of the most commonly used PHP design patterns include:
 - **Factory Pattern:** used for creating objects; a set of methods to make & return components of one object in various ways
 - **Singleton Pattern:** used for creating a single instance of an object; a class that distributes the only instance of itself
 - **Observer (PubSub) Pattern:** used for regulating the behavior of objects; an object notifies one or more objects if it changes
 - **Strategy Pattern:** used for regulating the behavior of objects; an object directs the set of methods called; each method, which is in its own class, extends a common base class
 - **Model-View-Controller(MVC):** used for architecting an application; the theoretical basis is to separate the user's interactions in the view layer from the control (controller) and storage (model) of data



The PHP Company

Copyright © 2006-2013, Zend Technologies, Inc. | 3

Slide 4

MVC Elements

- The MVC concept:

Presentation



View

Flow Logic



Controller

Data Access



Model



The PHP Company

Copyright © 2006-2013, Zend Technologies, Inc.

| 4

How MVC is Mapped to the Web

- To implement MVC on the web, URLs must be mapped to Controllers
- ZF2 uses an approach that is completely configurable
- Define route keys >> identifies route type to use and the URL fragment/s
 - URL fragments can be parent/child routes
 - Can also be composed of fixed and variable components
- Routing configuration is often done in the `module.config.php` file

`http://www.example.com/route`

route

Reflects the route defined (possibly in
the `module.config.php` file)



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

| 5

- By default, the first segment of a URL path maps to a controller, and the second to an action
 - If no action is provided, the action index is assumed
 - If no controller is provided, the controller index is assumed (following the Apache convention that maps a DirectoryIndex automatically)
- Zend_Controller's dispatcher then takes the controller value and maps it to a class. By default, it Title-cases the controller name and appends the word Controller
- Similarly, the action value is mapped to a method of the controller class. By default, the value is lower-cased, and the word Action is appended
 - For example, zend.com is built using Zend Framework
 - services = controller
 - training = action
 - course-catalog = parameter
 - zend-framework = value

Slide 6

MVC Implementation Subcomponents

The MVC layer incorporates several subcomponents: `Zend\Mvc*`

`\Router`

contains **classes related to routing a request** (matching request to controller)

`\Controller`

includes **abstract dispatchable implementations** that **compose basic functionality** such as **plugins, event management, service management**, and more

`\Service`

provides a **set of ServiceManager factories and definitions** for the default application workflow

`\View`

provides **defaults** for renderer selection, view script specifications, helper registration, ...; also adds **listeners** to workflow to automate **view model creation & injection**, ...

Listeners



The PHP Company

Copyright © 2006-2013, Zend Technologies, Inc. | 6

- Note that there is no base class for Model because it is too application-specific...

Slide 7

ZF2 MVC: Event-Driven Architecture

Everything is an event within ZF2 MVC

- Starts when modules are loaded
 - Trigger an event to create a module autoloader
 - Trigger another event to load each module
- Create an application instance
 - Call **bootstrap**, and it triggers a **bootstrap event**
 - Call **run**, it triggers a **route & dispatch event**
...Similarly for **dispatch error, render and finish events**
- In each case, there is at least one **listener**, listening in on each event, and doing something

Events

Why is this important?

- Because at different points, the event-driven architecture allows you to **stop propagation**...ex: if you get a response object returned, can retrieve that object easily

Listeners

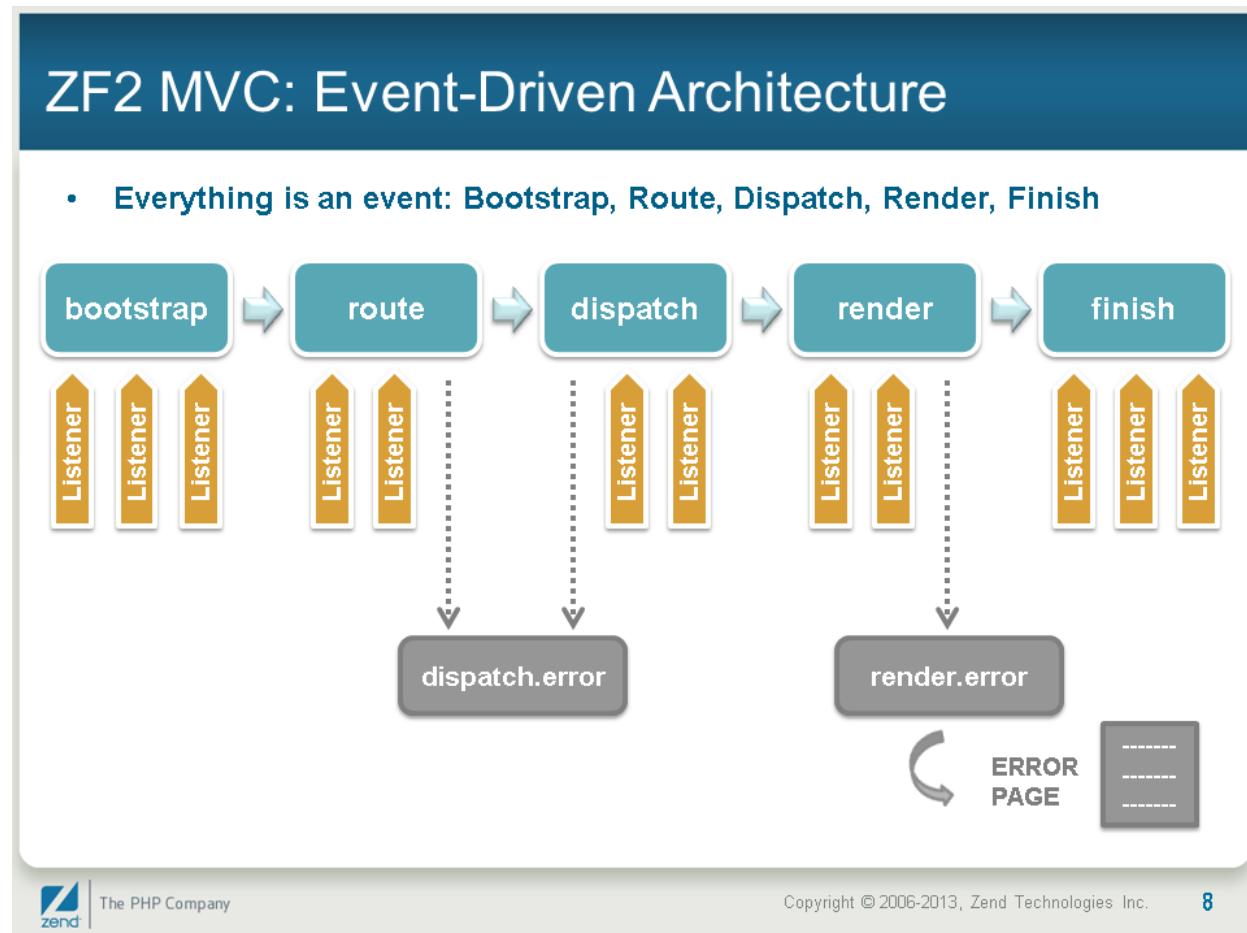
The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

7

- **ZF1 Users Alert**
 - ZF2 MVC allows for easier adjustment of workflow
 - No longer need to rewrite how front controller and dispatcher work
 - Because front controller was a singleton, you could not be easily replace it without potentially breaking the whole application
- Event architecture makes the framework more robust and adaptable

Slide 8



- Event architecture makes the framework more robust and adaptable
- Diagram does not show entire sequence of events, but shows general flow

Slide 9

Key MVC Events

Events / Triggers:

Event

Loading Modules: ModuleManager is a key aspect of MVC

`loadModule`...each module class loaded is passed so it can do something... way to aggregate SM definitions

`loadModules.post`...when done, can merge all service config & create/config SMS

Bootstrap: attaches listeners (ex: `RouteListener`, `DispatchListener`, `ViewManager`) and creates the `MvcEvent`

Route: matches request to controller, action, & parameters; if no match, triggers `dispatch.error` event

Dispatch: instantiates controller and runs `dispatch()` using routing info; if unable to load controller or dispatch action, triggers `dispatch.error` event

Render: view is finally rendered

Finish: used by default to send response to client via `SendResponseListener`; runs at -10000 priority (very late), to aid attaching other listeners, as needed



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

9

- Example: Want to do Caching: create two listeners, one on `bootstrap` with high priority, and one on `finish` with low priority; if it finds a cache, it will return it immediately and you're done!

Slide 10

Services in MVC

Now shifting focus from Events to Services ...

- The Service Manager (SM), within the default MVC, acts as a container that utilizes an Inversion of Control (IoC) approach
- Used to wire the default workflow and event listeners
- Developers can provide additional services and service configurations

In MVC, controllers are services - but how do you inject dependencies?

This is why SM is important... to meet the following needs:

- The ability to unit test controllers, specifically to (1) inject required dependencies, (2) call a method, and (3) test what you get back and make assertions against it
- With the SM, you define what dependencies you need in the controllers, either via the constructor or setters, and then use the SM to create a Factory that goes and injects those dependencies for you
- The SM gets passed to the default controllers defined within MVC ... see notes



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

10

- ZF1 User Alert
 - Service Locator is analogous to the Bootstrap
- The ZF2 Best Practice is to create a Factory service to inject the dependencies you are going to use
- But... the SM is automatically passed to the default controllers defined within ZF2
- Inconsistency with best practice? Somewhat, but here's why it's done:
 - This step is needed for proper app workflow...
 - Need some knowledge of what controllers are available to be able to pull them out of the system with their dependencies

ZFTool

ZF2 Command Line Tool

- Provides a quick way to create essential elements or display important info
- Commands are issued from the command line, and must be preceded with "php"
 - Ex: To create a project: `php zftool.phar create project name-of-project`
- Help can be obtained via: `php zftool.phar --help`

Sample Commands By Type:

- Basic information:

```
zftool.phar modules [list]
```

Displays loaded modules

```
zftool.phar version | --version
```

Displays current Zend Framework version



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

11

Slide 12

ZFTool

- **Project creation:**

```
zftool.phar create project <path>
```

Creates a skeleton application, where

<path> = path of the project to be created

- **Module Creation:**

```
zftool.phar create module <name> [<path>]
```

Creates a module, where

<name> = name of the module to be created

<path> = root path of the ZF2 application in which module is created



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

12

ZFTool

- **ZF2 Installation:**

```
zftool.phar install zf <path> [<version>]
```

Installs Zend Framework 2, where

<path> = directory in which to install the ZF2 library

<version> = version to install; if not specified, uses the last available

- **Application configuration:**

```
zftool.phar config [list]
```

Lists all configuration options

```
zftool.phar config get <name>
```

Displays a single config value, ex: config get db.host

```
zftool.phar config set <name> <value>
```

Sets a single config value (use only to change scalar values)



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

13

ZFTool

- **Classmap Generator:**

```
zftool.phar classmap generate <directory> <classmap file>
[--append|-a] [--overwrite|-w]
```

Creates a classmap, where

```
<directory>      = directory to scan for PHP classes; use "." for current directory
<classmap file> = filename for generated class map file or for standard output;
                  if not supplied, defaults to autoload_classmap.php
                  inside <directory>
--append | -a    = append to classmap file if it exists
--overwrite | -w = whether or not to overwrite existing classmap file
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

14

Slide 15

M2Ex1: MVC-based Zend Skeleton App

- See Participant Exercises Instruction booklet for details...



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

15

Slide 1



The PHP Company

“ Module Three ”

Copyright © 2006-2013, Zend Technologies Inc.

Slide 2

Module Three Topics

- In this module, we will cover:
 - Event Manager
 - Shared Event Manager



The PHP Company

Copyright © 2006-2013, Zend Technologies, Inc. | 2

Slide 3

Event Manager

What is the Event Manager?

- It is an object, used to perform two basic functions:
 - ❖ Trigger events
 - ... specify conditions to be notified
 - ❖ Trigger/notifies listeners on a per-instance basis, or by shared collections
 - Attach – Detach – Halt Execution - Aggregate listeners, which
 - listen & react to triggered events
 - ... not notified unless event is triggered, but know it is checking

Event**Listener**

- What is an event? a listener?

An event is an action
... a message that gets passed around

A listener is a callback that reacts to an event
... gets a single argument, the event action itself



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

3

- There are two important concepts attached to the Event Manager: Events & Listeners
- If you have done some JavaScript programming, the concept of events is familiar
- The Event Manager component is designed to implement:
 - Simple Pub/Sub (also called subject/observer) patterns
 - Aspect-Oriented designs
 - Event-driven architectures
- The basic EventManager architecture allows you to:
 - Attach and detach listeners to named events on a per-instance basis, or via shared collections
 - Trigger events
 - Interrupt execution of listeners
- An event is a message in the OOP meaning of the word; in other words, an object

Slide 4

Event Manager: Implementation

The Implementation of EM in ZF2 is actually a combination of several common patterns:

- Publish - Subscribe (**Pub/Sub**) pattern
 - a messaging pattern that is loosely coupled
- Observer (**Subject/Observer**) pattern
 - Pattern in which an object (or subject) notifies dependencies (observer objects) of state changes by calling their methods
 - Commonly used with Event-based systems, and the MVC architectural pattern
- Elements of **aspect-oriented programming**, but not as automated (AOP is usually compiler assisted)
- Concept of **events**, which allow you to do things like to stop propagation and aggregate results



The PHP Company

Copyright © 2006-2013, Zend Technologies, Inc.

4

Slide 5

Event Manager: Code Example

Simple Code Example: A "Per Instance" Event Manager

Event**Listener**

```
1 use Zend\EventManager\EventManager;
2
3 $events = new EventManager();
4 $events->attach('do', function ($e) {
5     $event = $e->getName();
6     $params = $e->getParams();
7     printf(
8         'Handled event "%s" with parameters "%s"',
9         $event,
10        json_encode($params)
11    );
12 });
13 $params = array('foo' => 'bar', 'baz' => 'bat');
14 $events->trigger('do', null, $params);
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

5

What's Happening in this Code Example?

- Zend\EventManager\EventManager is the class provided by the base implementation
- \$events is a new EM, attached to the 'do' event
 - Event names are typically named after the method triggered (Best Practice)
- Using a callback here (this ex: the anonymous function) for the listener, but could be any PHP callable;
 - always gets a single argument – the event \$e in this case
- The anonymous function (closure) grabs the name of the event and related parameters, and executes a simple print function (printf)
- The Parameters are 'foo' and 'baz' , as defined in line 13
- Event triggered in L14
- The 2nd argument "null" is the context; usually the object calling the event, the target for the event

Event Manager: Trigger Events

- When the Event Manager triggers events, it takes three arguments:
 - The **event name** (usually the current function/method name)
 - The "context" (usually the current object instance;
 - The **parameters** (usually provided to the current function/method

```
class Foo
{
    // ... assume events definition from above

    public function bar($baz, $bat = null)
    {
        $params = compact('baz', 'bat');
        $this->getEventManager()->trigger(__FUNCTION__, $this,
                                         $params);
    }
}
```



- Note that you can stop execution, either invoking the `stopPropagation()` method or providing a callback to the `trigger()` method; most commonly used for error handling

Slide 7

Event Manager: Sample Method

```
trigger(string $event, mixed $context, mixed $params, callback  
$callback);
```

What does this method do?

- It triggers all listeners to a named event

How should this method be used?

- Use the current function/method name for `$event`, appending it with values such as `".pre"`, `".post"`, etc. as needed (**Best Practice**)
- `$context` should be the current object instance, or the name of the function, if not triggering within an object
- `$params` should typically be an associative array or `ArrayAccess` instance; Best Practice: Use the parameters passed to the function/method - `compact()` is often useful here; this method can also take a callback and behave in the same way as `triggerUntil()`



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 7

- This is only one of the available methods for Event Manager... the entire list can be found in the ZF2 Reference Manual, Section 8.1.4.

Slide 8

Event Manager: Available Methods

```
attach(string $event, callback $callback, int $priority);
```

If a `$callback` is provided, it attaches the `EventManager` instance, listening for the event `$event`

If a `$priority` is provided, the listener will be inserted into the internal listener stack using that priority; higher values execute earliest (Default is "1", negative priorities are allowed**)

The method returns an instance of `Zend\Stdlib\CallbackHandler`
This value can later be passed to `detach()` if desired

If you want this listener to respond to more than one event, simply use an array of event names, or use the wildcard "*" to indicate ANY event

```
attachAggregate(string|ListenerAggregate $aggregate);
```

If a string is passed for `$aggregate`, it instantiates that class

The `$aggregate` is then passed the `EventManager` instance to its `attach()` method so that it may register listeners

The `ListenerAggregateInterface` instance is returned



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

| 8

- Note that negative priorities are required to allow a listener to execute after those with default priority
- These are only some of the available methods for Event Manager... the entire list can be found in the ZF2 Reference Manual, Section 8.1.4.

Slide 9

Event Manager: Aggregates

What are Aggregates?

- Aggregates are used for '**stateful**' operations, for an object that has 'state' and needs to listen to more than one event, or is 'statefully' listening to one event
- They are also used to group related listeners that may not share 'state' but have related operations semantically
- An aggregate can attach directly to an event manager (EM)
 - So, events attach and pass an object that calls the aggregate and its attachments, OR calls and passes to the EM instance on the aggregate (either works)
 - Implement the `ListenerAggregateInterface`, which defines the two methods `attach()` and `detach()`
 - Pass the EM instance in the `attach()`, and do something with it
 - For `detach()`, loop through the listeners; if it finds one that is 'true', that one is pulled out of the aggregate

Why are they important?

- They allow for stateful operations and group-related operations in a single class



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

9

Zend\EventManager: Usage Examples

Example 1

```
$this->getEventManager()->trigger(__FUNCTION__ . '.pre',
    $this, array('entry' => $entry));
$this->getEntryMapper()->insert($entry);
$this->getEventManager()->trigger(__FUNCTION__ . '.post',
    $this, array('entry' => $entry));
```

- **Code Example:** demonstrates the use of triggering events for which other modules/code may have registered listeners.
... Note that in the Teaching app, this is not the case, but it commonly occurs.

Why is this important?

- This is a pattern that is frequently employed in the various stages of the MVC execution



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

10

Zend\EventManager: Usage Examples

Example 2

```
public function onBootstrap(EventInterface $event)
{
    $sm = $event->getApplication()->getServiceManager();
    $sm->get('guestbook-entry-service')->getEventManager()->attach(
        'add.post',
        array($this, 'onNewEntry')
    );
}
```

- **Code Example:** This simple code shows how to attach a listener to a local event manager



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

11

Zend\EventManager: Usage Examples

Example 3

```
{{{  
    public function create(Entry $entry)  
    {  
        $events = $this->getEventManager();  
        $events->attach(__FUNCTION__, function ($e) {  
            $mapper = $e->getTarget()->getEntryMapper();  
            $entry = $e->getParam('entry');  
            $mapper->insert($entry);  
        });  
        $events->trigger(__FUNCTION__, $this, array('entry' => $entry));  
    }  
}}}
```

- **Code Example:** This shows attaching a default listener in the method, and simply triggering it... this allows you to alter the method completely



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

12

Slide 13

Event Manager: Shared Event Listeners

What is a Shared Event Listener?

Listener

A shared event listener is simply a **listener** that has been **attached** to the shared event manager (**SEM**) instead of the local event manager (**EM**) instance.

Why would you use Shared Event Listeners?

- May want to attach to objects not yet created

Ex: in MVC, may want to attach to a controller not yet requested

- May want to attach to a group of objects

Ex: attach to all controllers



The PHP Company

Copyright © 2006-2013, Zend Technologies, Inc.

13

- Recall that with the SEM...
 - All EM instances pulled from SM are injected with a common SharedEventManager (SEM) instance
 - Retrieve SEM from the SM, or any EM instance
 - API is the same, except need to provide a 'context' or 'identifier' (usually a class or interface name) as a preliminary argument to attach()

Shared Event Manager

What is a Shared Event Manager? (SEM)

A shared event manager is an instantiable class that is used within the event manager to **allow events to propagate across the entire application**. Unlike the event manager, multiple instances of the shared event manager should not exist.

It is also used to aggregate shared listeners and provide them to individual EM instances.

It is possible to **access** the shared event manager from the `getSharedManager()` method of any event manager instance generated by the MVC.

Why would you use an SEM?

A typical use for the SEM is to aid in the implementation of aspect-oriented programming (AOP).

Ex: One aspect in your app is a **logging module**. The **logging aspect** would **attach a listener to the SEM during initialization**. When an app component triggers the “logging” event on its local EM instance, that trigger **propagates up to the SEM and the logging listener is notified**. This provides inter-aspect communication without having to use DI to inject the same logging object into every aspect.



- You will see EM and SEM often in ZF2 because *everything* in the framework is event-driven... this is how you are going to time different events
- Example: if you wanted to add authentication, typically would want to do this as an event listener that will listen in on a specific event, on dispatch, and will check whether the user is authenticated... if not, it sends back a 401 error code
- Example: in MVC, you have a route and a batch event; if you wanted to trigger some event/s before the route event, you would simply attach to the route event and give it/them a very large priority, like 100; for events you want to occur after routing, attach it/them with a negative priority
- Same process for events you want to occur pre- and post-dispatch

Shared Event Manager: Code Example

Two Different Ways to Get a Shared Event Manager Instance

- Line 1 shows grabbing it from an EM instance
- Line 2 shows grabbing it out of the Service Manager

Listeners

```
1 $shared = $events->getSharedManager();  
2 $shared = $services->get('SharedEventManager');  
3  
4 $shared->attach('Zend\Stdlib\DispatchableInterface', 'dispatch',  
5     $callback, $priority);
```

- **Code Example:** shows a common practice - attaching a listener to an SEM instance
 - ... Note that an SEM does not trigger events – it simply aggregates listeners that will trigger other event managers and send notification

Why is this important?

- This means any class implementing the dispatchable interface, if it triggers a dispatch event, will notify this particular callback



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

15

- The code example attaches the SEM to the dispatch interface from Zend Stdlib, to the 'dispatchment' for a dispatchable, and then adds a callback and priority

- The priority is always passed as a final argument to attach(), after the callback
- Internally, listeners are registered with an SplPriorityQueue instance

Default priority

== 1

High priority

== positive integers >1

run FIRST

Low priority

== negative integers

run LAST

- Way to manage the queue

Zend\EventManager: Usage Examples

Example 4

```
public function onBootstrap(EventInterface $event)
{
    $eventManager      = $event->getApplication()->getEventManager();
    $moduleRouteListener = new ModuleRouteListener();
    $moduleRouteListener->attach($eventManager);
    $eventManager->getSharedManager()->attach(
        'Zend\Stdlib\DispatchableInterface',
        MvcEvent::EVENT_DISPATCH,
        function (MvcEvent $event) {
            $sm      = $event->getApplication()->getServiceManager();
            $request = $event->getRequest();
            $logger  = $sm->get('Zend\Log\Logger');

            $logger->debug(sprintf(
                'Incoming request was of type %s',
                get_class($request)
            ));
        }
    );
}
```

- **Code Example:** See notes...



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

16

- The first three lines of this method demonstrate a somewhat backwards form of event registration
- The ModuleRouteListener prefixes a controller service with a module name, if provided in the route configuration
 - The purpose is to disambiguate controllers
- The remainder of the example demonstrates how to attach a listener to the shared event manager

Event Manager: Identifiers

What are Identifiers?

Identifiers are strings that group listeners within the SEM. When attaching an event to the SEM, you provide an identifier to specify the context where the listener should be called.

Best Practices

- Assign identifiers in a class' `setEventManager()` method
- Use class name, any parent classes a/o any interfaces the class implements as identifiers
 - provides ability to attach to a class, to a group of classes all extending a specific class, to all classes implementing specific interfaces, more

Setting Identifiers

- Typically set within the `setEventManager()` method
 - ... see notes

```
1 public function setEventManager(EventManagerInterface $events)
2 {
3     $events->setIdentifiers(array(
4         __CLASS__,
5         get_class($this),
6         'Some\Interface\I\Implement',
7     ));
8     $this->events = $events;
9     return $this;
10 }
```

Event



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

17

- When you inject with an EM, you want to set the Identifiers at the same time; this is why the Initializers work for setting the EM; you can have a bare instance, but then needs to be configured
 - Call `setIdentifiers()`
 - Pass in an array; often `__CLASS__` (magic constant), which is the defining class name
 - Call `get_class($this)` so if extend class later, do not need to define `setEventManager`; will have parent class defined and whatever the current class is
 - Then implement any interfaces; allows attachment to any of the contexts when attached to an SEM
 - Set event properties and return

Slide 18

M3Ex1: Event Manager

- See Participant Exercises Instruction booklet for details...



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 18

Slide 1



“ Module Four

Copyright © 2006-2013, Zend Technologies Inc.

Slide 2

Module Four Topics

- In this module, we will cover:
 - Service Manager (SM)
 - SM Service Types
 - SM Configuration



The PHP Company

Copyright © 2006-2013, Zend Technologies, Inc. | 2

Slide 3

Service Manager

What is the Service Manager (SM)?

- The ServiceManager provides an Inversion of Control (IoC) container for managing object life-cycles and dependencies via a configurable and programmable interface.
- ZF2 provides a number of default services via the MVC layer, and expects the developer to provide application-specific service configuration

How Do You Use It?

- Typically, you use a SM to:
 - Specify services you want to provide via instances, class name, or factory
 - Retrieve those services by name later
 - With a SM, you write the code upfront that shows what the injections are - no "magic"
... Explicit wiring makes debugging simpler... whenever SM responds that it cannot recreate an instance, you know exactly why



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

3

Dependency Injection: A solution for managing objects and their dependencies

- The concept of *how to manage object dependencies* within PHP has been evolving recently
- The solution as represented by Service Manager is generally named "dependency injection", while the abstract principle is actually called "inversion of control"

Slide 4

SM Service Types: Service (Explicit)

Service (Explicit): sets a particular instance to react as a service within the SM

Service commonly used to: list of parameters, ex: database configuration

```
'service_manager' => array(
    'services' => array(
        // This could also be an instance of an object or anything else really.
        'myConfigArray' => array(
            'db' => 'MySQL',
            'user' => 'web1',
            'password' => 'just4show',
        ),
    ),
);
// Or
$serviceManager->setService('myConfigArray', array(
    'db' => 'MySQL',
    'user' => 'web1',
    'password' => 'just4show',
));
```

Code Example: see notes



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

4

Services:

- Service/Explicit (name => object pairs)
 - Invokables (name => instantiable class)
 - Factories (name=> callable returning object)
 - Aliases (alias => some service or alias)
 - Abstract Factories (callable that handles many services)
 - Shared (or not; adjustable setting)
 - Initializers (perform operations on new instances)
- **Code Example:** sets an instance of a configuration array into the service manager. It is likely that this array will be consumed by one of the defined factories

SM Service Types: Invokables

Invokables: pairs a service name with an extensible class; constructor does not require arguments and/or setter injections

Service commonly used to: Identify classes that can be directly instantiated without dependencies

```
'service_manager' => array(
    'invokables' => array(
        'guestbook_servicce_dataMapper' =>
        'Guestbook\Service\DataMapper',
    ),
);

// or

$serviceManager->setInvokableClass('guestbook_servicce_dataMapper',
    'Guestbook\Service\DataMapper');
```

Code Example:

- Assigns a label `guestbook_servicce_dataMapper` to the `Guestbook\Service\DataMapper` class



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

5

Services:

- Service/Explicit (name => object pairs)
- Invokables (name => instantiable class)
- Factories (name=> callable returning object)
- Aliases (alias => some service or alias)
- Abstract Factories (callable that handles many services)
- Shared (or not; adjustable setting)
- Initializers (perform operations on new instances)

Slide 6

SM Service Types: Factories

Factories: When requested, execute code to give back the instance needed

Alternative to using callables; implement a Factory interface where you can pass a class name; will instantiate it and see if it is a factory

Service commonly used to: Set up factories that can be used to create instances of classes with dependencies injected

Code Example ... picture on next page

- Assigns a label `guestbook_entry_mapper` to an anonymous function
- Function produces an instance of the `Mapper\Entry` class
- Function sets db adapter, hydrator, and entity prototype
- Assigns a label `guestbook_entry_form` to an anonymous function
- Function produces an instance of the `Form\Entry` class
- Function sets hydrator and input filter



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

6

Services:

- Service/Explicit (name => object pairs)
- Invokables (name => instantiable class)
- Factories (name=> callable returning object)
- Aliases (alias => some service or alias)
- Abstract Factories (callable that handles many services)
- Shared (or not; adjustable setting)
- Initializers (perform operations on new instances)
- Factories:
 - Can be any PHP callable -- static method, instance method, function, closure, even functor (class instance implementing `__invoke()`)
 - If you specify a class name, it can implement FactoryInterface *OR* `__invoke()`

Slide 7

SM Service Types: Factories (Code Example)

Code Example

```
$serviceManager->setFactory('guestbook_entry_form', function ($sm) {
    $form = new Form\Entry();
    $form->setHydrator(new ClassMethods());
    $form->setInputFilter($sm->get
        ('guestbook_entry_filter'));
    return $form; });

// or

'service_manager' => array(
    'factories' => array(
        'guestbook_entry_form' => function ($sm) {
            $form = new Form\Entry();
            $form->setHydrator(new ClassMethods());
            $form->setInputFilter($sm->get('guestbook_entry_filter'));
            return $form;
        },
    ),
),
)
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

7

Services:

- Service/Explicit (name => object pairs)
- Invokables (name => instantiable class)
- Factories (name=> callable returning object)
- Aliases (alias => some service or alias)
- Abstract Factories (callable that handles many services)
- Shared (or not; adjustable setting)
- Initializers (perform operations on new instances)

SM Service Types: Factories (Alternate Example)

Code Example (using factory classes instead of closure)

```
'service_manager' => array(
    'factories' => array(
        'guestbook_entry_form'=>'Guestbook\Service\EntryFormFactory',
    ),
);

// Or
(serviceManager->setFactory('guestbook_entry_form',
    'Guestbook\Service\EntryFormFactory');

// And the factory class itself:
// Guestbook\Service\EntryFormFactory
class EntryFormFactory implements FactoryInterface
{
    public function createService(ServiceLocatorInterface $sm)
    {
        $form = new Form\Entry();
        $form->setHydrator(new ClassMethods());
        $form->setInputFilter(new Form\EntryFilter());
        return $form;
    }
}
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

8

Services:

- Service/Explicit (name => object pairs)
- Invokables (name => instantiable class)
- Factories (name=> callable returning object)
- Aliases (alias => some service or alias)
- Abstract Factories (callable that handles many services)
- Shared (or not; adjustable setting)
- Initializers (perform operations on new instances)

Slide 9

SM Service Types: Aliases

Aliases: when this service is requested, actually indicating another; aliases are especially useful when writing reusable modules (ex: need DB adapter for a given module - alias it to a well-known service; then can have it go to a different service for another type of DB)

Service commonly used to: Create internal labels for other services, to improve the readability of your code

```
'service_manager' => array(
    'aliases' => array(
        'my_foo' => 'foo',
        'foo_master' => 'my_foo',
    ),
);
// or
$serviceManager->setAlias('my_foo', 'foo');
$serviceManager->setAlias('foo_master', 'my_foo');
```

Code Example:

- First line is aliasing a service
- Line 3 is aliasing an alias... this provision is made in case you wish to refer to the same service using a more "friendly" label



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

9

Services:

- Service/Explicit (name => object pairs)
- Invokables (name => instantiable class)
- Factories (name=> callable returning object)
- Aliases (alias => some service or alias)
- Abstract Factories (callable that handles many services)
- Shared (or not; adjustable setting)
- Initializers (perform operations on new instances)

SM Service Types: Abstract Factories

Abstract Factories: check whether it is possible to create an instance for an undefined service and, if so, whether it has a method that will create and return it; implements the `AbstractFactory` Interface

Service commonly used to: set up conditional services; plug-ins

Code Example: `abstract_factories` is key; can provide either the name of a class that implements the AF interface or can provide an instance of that AF;

Has to be this interface because it looks at two methods (can't just use a callback)

```
array('abstract_factories'=> array(
    'Class\Implementing\AbstractFactoryInterface',
    $someAbstractFactoryInstance,
));
class SampleAbstractFactory implements AbstractFactoryInterface
{
    public function canCreateServiceWithinName(
        ServiceLocatorInterface $services, $name, $requestedName
    ) /* */
    public function createServiceWithinName(/* same signature */)
    /* */
}
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

10

Services:

- Service/Explicit (name => object pairs)
- Invokables (name => instantiable class)
- Factories (name=> callable returning object)
- Aliases (alias => some service or alias)
- Abstract Factories (callable that handles many services)
- Shared (or not; adjustable setting)
- Initializers (perform operations on new instances)

SM Service Types: Shared

Shared: default setting - every service within the SM container is shared; each time you fetch the service from the SM, you get the same instance

ZF2 has implemented 2 services:

- A shared EM service (SEM) that is invokable and shared by default
- An EM service, not shared by default, which has a factory that pulls the SEM service and injects it into the EM instance
- This allows for multiple instances of EM

Code Example:

```
array('shared' => array(
    'EventManager' => false, //default is true
));
```

Service commonly used to: this option is generally used when you need new specific instances (as services are shared by default);

You would mark such services as shared FALSE. For times when you *want* multiple instances, choose *not* to share

Ex: for validators and filters, which have their own SM instances; not sharing allows different REGEX validators to have their own patterns



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

11

Services:

- Service/Explicit (name => object pairs)
- Invokables (name => instantiable class)
- Factories (name=> callable returning object)
- Aliases (alias => some service or alias)
- Abstract Factories (callable that handles many services)
- Shared (or not; adjustable setting)
- Initializers (perform operations on new instances)

SM Service Types: Initializers

Initializers: Once an instance is created/returned, an initializer will determine if that instance needs to do something else. Typically, they check if an instance implements a specific interface, and if so, whether it injects something; An initializer can be the name of a class or an instance of a class implementing `InitializerInterface`, or any valid PHP callback; any initializer receives two arguments, the instance just created, and the SM

Service commonly used to: check the instance passed against a specific interface and, if it matches, inject it with a service from the service manager.

Code Example: When this object is created using the Service Manager, the method `injectControllerDependencies()` will be called

```
/**  
 * Constructor  
 * After invoking parent constructor, add an initializer to inject the SM, EM,  
 * Plugin Manager... @param null|ConfigInterface $configuration  
 */  
public function __construct(ConfigInterface $configuration = null)  
{  
    parent::__construct($configuration);  
    // Pushing to bottom of stack to ensure this is done last  
    $this->addInitializer(array($this, 'injectControllerDependencies'), false);  
}
```



Services:

- Service/Explicit (name => object pairs)
- Invokables (name => instantiable class)
- Factories (name=> callable returning object)
- Aliases (alias => some service or alias)
- Abstract Factories (callable that handles many services)
- Shared (or not; adjustable setting)
- Initializers (perform operations on new instances)

Configuring Services

You can configure services anywhere!

- **Main Application Configuration:** Where you define what modules are available, you can also provide some initial configuration for the SM
- **Module Classes:** Can define a number of methods that are also part of implementing the interfaces; allow you to modify various SMs; main Service Factory, invokable factories, and sometimes Abstract factories
- **Module Configuration:** returns an array; aggregated; can provide a single config to the SM before you bootstrap
- **Local Override Configuration:** specify config and when to merge; alter and provide substitutions



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

13

Slide 14

M4Ex1: Service Manager

- See Participant Exercises Instruction booklet for details...



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 14

Slide 1



The PHP Company

“ Module Five

Copyright © 2006-2013, Zend Technologies Inc.

Slide 2

Module Five Topics

- In this module, we will cover:
 - MVC in detail
 - Modules
 - MVC and Modules
 - ModuleManager
 - Module-specific Configuration
 - Best Practices for Module Creation



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 2

- The config directory has two purposes: configuring the ModuleManager, as stated above, but also configuring the ServiceManager
- config/application.config.php is used to provide default services for the SM, and to provide configuration to the module manager and its listeners
- config/autoload/* is used to provide service and application configuration
- It also has an "autoload" directory, and these are configuration files for overriding defaults provided by modules... this configuration usually seeds the ServiceManager

Slide 3

MVC: Basic Structure of an Application

`public/index.php` performs the basic work of configuring the application; it then runs the app and returns the response

- The `config` directory will typically contain configuration used by `Zend\ModuleManager` in order to load modules and merge configurations
- The `vendor` subdirectory should contain all third-party modules or libraries (ZF, custom, ...); these libraries and modules should *not* be modified from their original, distributed state
- The `module` directory is where to write modules specific to the app; write modules as if they will be reused; module dir should contain few modules

```
application_root/
  init_autoloader.php
  config/
    application.config.php
    autoload/
      global.php
      local.php
      // etc.
  data/
  module/
  public/
    css
    images
    js
    .htaccess
    index.php
  vendor/
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 3

Slide 4

Modules in MVC

So, in ZF2 MVC:

Controllers are services... they can trigger events or be triggered by events...

Therefore, these are **event-driven services**

Where do we define the controllers and how do we configure them?

Answer: **Modules**

Generally, a module is a set of related code & assets that solve a specific problem



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

4

MVC: Basic Structure of a Module

- The ZF team recommends this generic structure for ZF2 modules

```
module_root/
  Module.php
  config/
    module.config.php
  public/
    images/
    css/
    js/
  src/
    <module_namespace>/           <MVC directories & code files>
    <MVC directories & test code files>
  test/
    phpunit.xml
    bootstrap.php
    <module_namespace>/           <MVC directories & test code files>
    <dir-named-after-module-namespace>/<dir-named-after-a-controller>/<.phtml files>
```



- The config directory should contain any module-specific configuration; these files may be in any format Zend\Config supports
- We recommend naming the main configuration "module.format", and for PHP-based configuration, "module.config.php"; typically, you will create configuration for the router as well as for the dependency injector
- The src directory should be a PSR-0 compliant directory structure with your module's source code. Typically, you should at least have one subdirectory named after your module namespace; however, you can ship code from multiple namespaces if desired.
- The test directory should contain your unit tests. Typically, these will be written using PHPUnit, and contain artifacts related to its configuration (ex: phpunit.xml, bootstrap.php)
- The public directory can be used for assets that you may want to expose in your application's document root. These might include images, CSS files, JavaScript files, etc. How these are exposed is left to the developer
- The view directory contains view scripts related to your controllers

MVC: Bootstrapping an Application

- An application has six basic dependencies, which can be satisfied at instantiation:

Configuration
• Usually an array or object implementing ArrayAccess
ServiceManager
• An instance
EventManager
• An instance, pulled from ServiceManager
ModuleManager
• An instance, pulled from ServiceManager
Request
• An instance, pulled from ServiceManager
Response
• An instance, pulled from ServiceManager

- Then, the application can be bootstrapped and run...



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

| 6

- Sample code for instantiation:

```
use Zend\EventManager\EventManager;
use Zend\Http\PhpEnvironment;
use Zend\ModuleManager\ModuleManager;
use Zend\Mvc\Application;
use Zend\ServiceManager\ServiceManager;

$config = include 'config/application.php';

$serviceManager = new ServiceManager();
$serviceManager->setService('EventManager', new EventManager());
$serviceManager->setService('ModuleManager', new ModuleManager());
$serviceManager->setService('Request', new PhpEnvironment\Request());
$serviceManager->setService('Response', new PhpEnvironment\Response());

$application = new Application($config, $serviceManager);
```

MVC: Bootstrapping an Application

The Application Instance:

1. At instantiation



- Grabs the ModuleManager and calls loadModules()**
 - This has the end result of grabbing and merging all configuration, as well as configuring the ServiceManager
- Grabs the Request and Response from the ServiceManager**
- Stores the merged configuration from the ModuleManager**



The PHP Company

Copyright © 2006-2013, Zend Technologies, Inc.

| 7

MVC: Bootstrapping an Application

The Application Instance:

2. During `Bootstrap()`



Retrieves the `RouteListener` and `DispatchListener` from the `ServiceManager`; attaches them to the composed `EventManager`

Retrieves the `ViewManager` from the `ServiceManager`

Creates an `MvcEvent`

Triggers the "bootstrap" event, passing it the `MvcEvent`



The PHP Company

Copyright © 2006-2013, Zend Technologies, Inc.

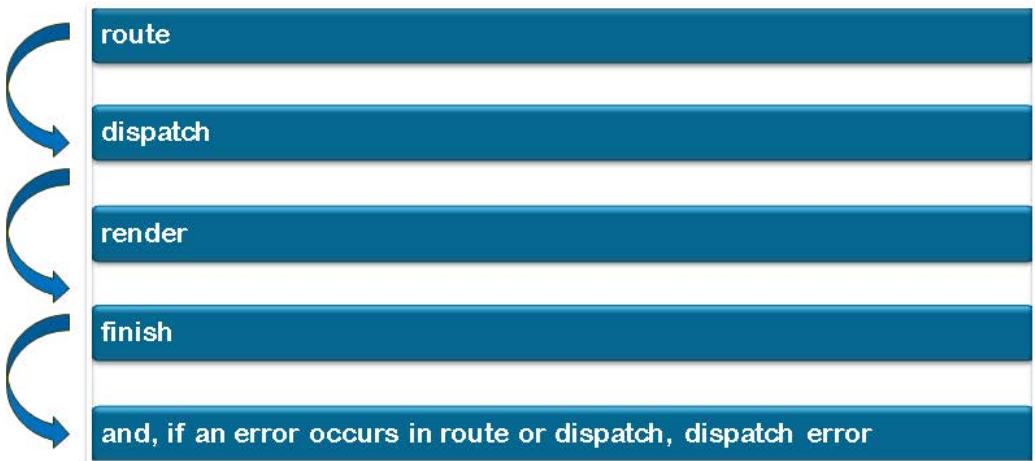
| 8

Slide 9

MVC: Bootstrapping an Application

The Application Instance:

3. During `run()`, triggers the following events, passing the `MvcEvent`



The PHP Company

Copyright © 2006-2013, Zend Technologies, Inc.

| 9

MVC: Bootstrapping an Application

The Application Instance:

RouteListener: Uses the `TreeRouteStack` to match a request to a controller
(default for HTTP; CLI requests use the Console router)

DispatchListener: From the service manager, it gets a controller manager, which
then builds an instance of the controller named by the route
match; attempts to run the controller's `dispatch()` method;
triggers MVC "dispatch" event upon success,
"dispatch.error" otherwise

View: triggers View Render and Response events; the chosen rendering strategy
renders the View using the view model; the chosen response strategy
populates the response object



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 10

- DispatchListener: It does not pull the controller from the `ServiceManager`! It pulls the `Zend\Mvc\Controller\ControllerManager` from the `ServiceManager`, and then pulls the controller from that
- The "view" event is triggered, and that calls on `Zend\View\View::render()`, passing it the view model composed in the MvcEvent. That method triggers two events on its own `EventManager` instance, "render" and `response`; these are used to select a renderer, and populate the response, respectively.

Slide 11

MVC: Bootstrapping a Modular Application

Modular applications require an additional consideration...

- Each module has its own configuration, which must be aggregated
- `Zend\ModuleManager\ModuleManager` allows you to specify which modules exist and where they are located, and then initialize them
- Module classes can tie into various listeners on the `ModuleManager` to provide
 1. Configuration
 2. Services
 3. Listeners
 4. more...



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 11

MVC: Bootstrapping a Modular Application

1. Configuring the Module Manager:

- Specify which modules to load; (optional) provide configuration for the module listeners

```
// config/application.config.php
<?php
return array(
    'modules' => array(
        'Application',
        'Guestbook',
        'GuestbookNotification',
        'ZfcBase',
    ),
    'module_listener_options' => array(
        'config_glob_paths'      => array(
            'config/autoload/{,*.}{global,local}.php',
        ),
        'module_paths' => array(
            './module',
            './vendor',
        ),
    ),
);
```



Slide 13

MVC: Bootstrapping a Modular Application

2. Add items to the `modules` array as each module is added to the app:

- Module classes that need to inform the application about its config should define a `getConfig()` method
- This method should return an array or traversable object, such as `Zend\Config\Config`
- Define other methods as needed - providing autoloader configuration, providing services to the `ServiceManager`, ...



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 13

MVC: Bootstrapping a Modular Application

3. Launch the application

- Create a new vhost pointing its doc root to the `public` directory of the app
- Run it in a browser... you should see a page of view script output that corresponds to the module, controller, and action requested
- If the page is not found, then the "Route Not Found" strategy will be triggered
- If there is an exception, then the "Exception Strategy" will be triggered, and the output will vary depending upon the stage of development

Zend\ Mvc\ View\ *		
Context		
Default Rendering Strategy	Exception Strategy	Route Not Found Strategy
<i>How is the view going to be rendered?</i>	<i>How are you going to handle exceptions?</i>	<i>What do you do if the route is not found?</i>



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 14

MVC Events

The MVC implementation in ZF2 incorporates and utilizes a custom `Zend\EventManager\EventInterface` type, `Zend\Mvc\MvcEvent`

- This event is created during `bootstrap()`, and is passed directly to all events that method triggers
- Additionally, if you mark your controllers with the `Zend\Mvc\InjectApplicationEvent` interface, it will be injected into them
- `MvcEvent` adds accessors & mutators for certain objects, stores the `RouteMatch` object, and defines certain methods

Accessor	Mutator	Object Involved
<code>getApplication()</code>	<code>setApplication()</code>	<code>Zend\Mvc\Application</code>
<code>getController()</code>	<code>setController()</code>	Custom controller class
<code>getRequest()</code>	<code>setRequest()</code>	<code>Zend\Stdlib\RequestInterface</code>
<code>getRouteMatch()</code>	<code>setRouteMatch()</code>	<code>\Router\RouteMatch</code>



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 15

- The full list of accessors, mutators that `MvcEvent` adds and the methods it defines can be viewed at: <http://packages.zendframework.com/docs/latest/manual/en/zend.mvc.mvc-event.html>
- The Application, Request, Response, Router, and ViewModel are all injected during the bootstrap event
- Following the route event, it will be injected with the `RouteMatch` object encapsulating the results of routing
 - As this object is passed around throughout the MVC, it is a common location for retrieving the results of routing, the router, and the request and response objects
 - Setting the results of execution in the event allows event listeners to introspect them and utilize them within their execution; Ex: the results could be passed into a view renderer

MVC-Related Module Configuration

As we have seen in previous sections, the Service Manager can be configured to help instantiate classes in various ways.

So far we have only looked at generic configuration, which is not used by the MVC components of ZF2.

There are, however, some dedicated sections of the configuration / module class used by the MVC for various purposes. These are as follows:

Use	Configuration Key	Module method
Configure controller classes	<code>controllers</code>	<code>getControllerConfig()</code>
Configure controller plugin classes	<code>controller_plugins</code>	<code>getControllerPluginConfig()</code>
Configure view helper classes	<code>view_helpers</code>	<code>getViewHelperConfig()</code>



- These keys are *top-level* keys

Slide 17

SM Configuration for MVC: Code Example

```
class Module
{
    public function getControllerConfig()
    {
        return array(
            'invokables' => array(
                'Guestbook\Controller\Index' =>
                    'Guestbook\Controller\IndexController',
            ),
            'factories' => array(...),
            'services'  => array(...),
            'aliases'   => array(...),
        );
    }
}

// or

'controllers' => array(
    'invokables' => array(
        'Guestbook\Controller\Index' => 'Guestbook\Controller\IndexController',
    ),
    'factories' => array(...),
    'services'  => array(...),
    'aliases'   => array(...),
),
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

Zend\ModuleManager

ZF2 has redesigned the way the framework handles modules

- A module is a PHP namespace
- Modules can contain anything: PHP code, Library code, View Scripts, Public assets (ex: CSS, JS)
- The Module system is composed of:

ModuleAutoloader

A specialized autoloader responsible for locating and loading each modules' `Module` class

ModuleManager

Events

Commences a sequence of events for each array of module names, permitting the module system behavior to be defined entirely by Listeners attached to the Module Manager

ModuleManager Listeners

Listeners

Listeners can be attached to a module manager's events, and can do multiple things, including loading and resolving modules, performing complex initialization tasks, and introspection into each returned module object



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 18

- The name of a module in a typical ZF2 application is simply a PHP namespace (which must conform with namespace naming rules)
- For a module to integrate into the MVC, it needs a `Module` class in its namespace

Slide 19

Module Autoloader

We discussed Autoloaders in the first module... here is one specific to modules:

`Zend\Loader\ModuleAutoloader`

Listeners

- The `ModuleAutoloader` is consumed by `Zend\ModuleManager\Listener\ModuleLoaderListener` in order to resolve module names to their associated Module classes

... Typically, you will never interact with this autoloader directly



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 19

Zend\ModuleManager

What is the ZF2 Module Manager?

`Zend\ModuleManager\ModuleManager`

Events

- A simple class responsible for iterating over an array of module names and triggering an event for each
- Attached event listeners are responsible for...
 - Gathering autoloader configuration
 - Gathering application configuration
 - Triggering module-specific initialization tasks

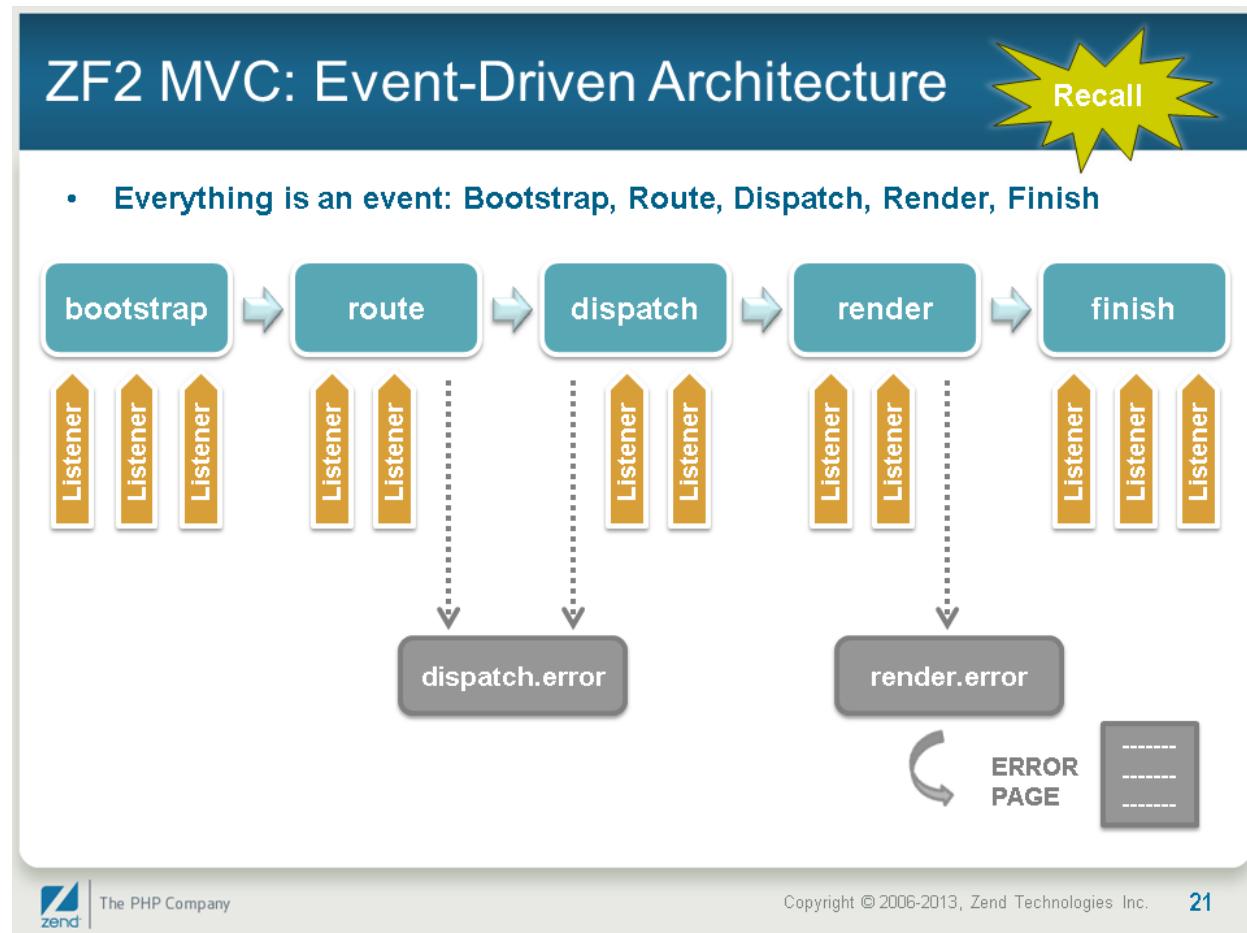
Listeners



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 20

Slide 21



- Event architecture makes the framework more robust and adaptable
- Diagram does not show entire sequence of events, but shows general flow

Slide 22

ModuleManager Events

What events does the Module Manager trigger?

[Events](#)[Listeners](#)

`loadModules`

Triggered when `loadModules()` is called, it wraps the calls to `loadModule()` (and the `loadModule` event). The module autoloader is attached early in this event, among other things

`loadModule.resolve`

Triggered for each module to be loaded; The listener(s) to this event are responsible for taking a module name and resolving it to an instance of some class. The default module resolver shipped with ZF2 simply looks for the class `{modulename}\Module`, instantiating and returning it if it exists

`loadModule`

Once a module resolver listener has resolved the module name to an object, the ModuleManager triggers this event, passing the created object to all listeners

`loadModules.post`

Triggered by Module Manager to allow listeners to perform work after every module is finished loading



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 22

- An example of how `loadModules.post` works:

The default configuration listener, `Zend\ModuleManager\Listener\ConfigListener`, attaches to this event to incorporate additional user-supplied configuration meant to override the default configurations of installed modules

ModuleManager Event Listeners

ZF2 provides a set of pre-made Listeners... some important examples are:

DefaultListenerAggregate

Attached by default, and attaches the default set of `ModuleManager` listeners, providing each configuration

Listeners

AutoloaderListener

Checks each module to see if it has implemented `Zend\ModuleManager\Feature\AutoloaderProviderInterface` or just defined `getAutoloaderConfig()`; if the latter is true, the method is called on the module class & passes the returned array to `Zend\Loader\AutoloaderFactory`

ConfigListener

If a module class has a `getConfig()` method, this listener will call it & merge the returned array (or `Traversable` object) into the main application configuration

OnBootstrap

With the application fully configured, this listener gets passed an `MvcEvent` providing access to the `ServiceManager`, `Application` instance, and main `EventManager` instance



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 23

- The complete list of provided listeners is:
 - `Zend\ModuleManager\Listener\DefaultListenerAggregate`
 - `Zend\ModuleManager\Listener\AutoloaderListener`
 - `Zend\ModuleManager\Listener\ConfigListener`
 - `Zend\ModuleManager\Listener\InitTrigger`
 - `Zend\ModuleManager\Listener\LocatorRegistrationListener`
 - `Zend\ModuleManager\Listener\ModuleResolverListener`
 - `Zend\ModuleManager\Listener\OnBootstrapListener`
 - `Zend\ModuleManager\Listener\ServiceListener`
- When implementing `getConfig()`, you can return a `Zend\Config` instance. This allows you to provide configuration in your module that uses one of the alternate formats `Zend\Config` supports
- The `init()` method is executed after the Module is instantiated, but other modules may not yet be instantiated - so, do not do anything in here that depends on other modules

Slide 24

Autoload Files for Modules

Three autoload files are provided as a default mechanism for autoloading module classes... a simple way to consume the module without requiring the Module Manager

These files are recommended, not required

`autoload_classmap.php`

Returns an array `classmap` of class name/ filename pairs (with the filenames resolved via the `_DIR_` magic constant; `bin/classmap_generator.php` is a utility to create this

`autoload_function.php`

Returns a PHP callback that can be passed to `spl_autoload_register()`. Typically, this callback utilizes the map returned by `autoload_classmap.php`

`autoload_register.php`

Registers a PHP callback (typically returned by `autoload_function.php` with `spl_autoload_register()`)



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 24

- Autoload_classmap.php should return an empty array during early development; generate it once your module code is solidifying
- Will be covered in your lab...

Slide 25

Module Classes

The module class is the single entry point for `ModuleManager` listeners to interact with a module

Listeners

- ZF2 expects each module name to be resolvable to an object instance
- Default resolver `Zend\ModuleManager\Listener\ModuleResolverListener` instantiates an instance of `moduleName\Module` for each enabled module
- Ex: Minimal Module class ...

Module: `MyModule` ...
Expected class: `MyModule\Module`

```
// Define your MyModule\Module
// class in Module.php
namespace MyModule;
class Module
{
}
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 25

Typical Module Class

```
namespace MyModule;

class Module
{
    public function getAutoloaderConfig()
    {
        return array(
            'Zend\Loader\ClassMapAutoloader' => array(
                __DIR__ . '/autoload_classmap.php',
            ),
            'Zend\Loader\StandardAutoloader' => array(
                'namespaces' => array(
                    __NAMESPACE__ => __DIR__ . '/src/',
                    __NAMESPACE__,
                ),
            ),
        );
    }
    public function getConfig()
    {
        return include __DIR__ . '/config/module.config.php';
    }
}
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | **26**

Slide 27

SM Configuration for Modules

```
class Module {
    public function getServiceConfig()
    {
        return array(
            'invokables' => array(
                'guestbook_service_dataMapper' =>
                    'Guestbook\Service\DataMapper',
            ),
            'factories' => array(
                'guestbook_entry_form' => function($sm) {
                    $form = new Form\Entry();
                    $form->setHydrator(new ClassMethods());
                    $form->setInputFilter(new Form\EntryFilter());
                    return $form;
                },
                'guestbook_entry_form_alt' =>
                    'Guestbook\Service\EntryFormFactory',
            ),
            'services' => array(
                // Could also be an instance of an object or anything else really.
                'myConfigArray' => array(
                    'db' => 'MySQL',
                    'user' => 'web1',
                    'password' => 'just4show',
                ),
                'aliases' => array(
                    'myAlias' => 'guestbook_entry_form',
                ),
            );
        );
    }
}
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 27

- Note that the "getServiceConfig()" method returns an array (or Traversable) of configuration exactly like the configuration we saw in Module One.
- Also, the module configuration should typically be done in module.config.php; if you want to create closures, or will be working with object instances to configure the SM, it should ONLY be done in this method

Slide 28

MVC 'bootstrap' Event

Modules based on an MVC design pattern may need access to additional parts of the app in the Module class

Events

- Ex: An instance of `Zend\Mvc\Application` or its registered SM instance
- An MVC bootstrap event can be used to accomplish this
- Bootstrap event is triggered by `Zend\Mvc\Application::bootstrap`
 - At the time the bootstrap event is triggered ...
 - All modules are loaded
 - The SM is fully configured
 - The application instance is available (as it is the event target)



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 28

Slide 29

MVC 'bootstrap' Event: Code Example

```
// module/GuestbookNotification/Module.php
class Module implements BootstrapListenerInterface
{
    public function onBootstrap(EventInterface $event)
    {
        $sm = $event->getApplication()->getServiceManager();
        $sm->get('guestbook-entry-service')
            ->getEventManager()
            ->attach('add.post', array($this, 'onNewEntry'));
    }

    public function onNewEntry(EventInterface $event)
    {
        $message = new Message();
        $message->addFrom('noreply@localhost', 'Guestbook Notifier')
            ->addTo($event->getParam('entry')->getEmail())
            ->setSubject('Thank you!')
            ->setBody('Thank you for leaving a guestbook entry!')
            ->setEncoding('utf-8');
        $transport = new SendmailTransport();
        $transport->send($message);
    }
}
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 29

Slide 30

ModuleAutoloader & DefaultListenerAggregate

ZF2 ships with a default, specialized module autoloader that is responsible for the location and on-demand loading of Module classes

`Zend\Loader\ModuleAutoloader`

Listeners

- Use the default listener and provide an array of module paths (absolute or relative) to the app root, for the module autoloader to check when loading modules
- `Zend\ModuleManager\Listener\DefaultListenerAggregate`, is an aggregate listener that registers the default module listeners; one of those is the `ModuleLoaderListener`, which consumes the `ModuleAutoloader`



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 30

Module Autoloader: Default Listener Aggregate

```
// Zend\ModuleManager\Listener\DefaultListenerAggregate::attach()
public function attach(EventManagerInterface $events)
{
    $options                  = $this->getOptions();
    $configListener           = $this->getConfigListener();
    $locatorRegistrationListener = new LocatorRegistrationListener($options);
    $moduleAutoloader         = new ModuleAutoloader($options->
                                                getModulePaths());
    // High priority, module autoloading should be available before anything else
    $this->listeners[] = $events->attach(ModuleEvent::EVENT_LOAD_MODULES,
                                             array($moduleAutoloader, 'register'), 9000);
    $this->listeners[] = $events->attach(
        ModuleEvent::EVENT_LOAD_MODULE_RESOLVE,
        new ModuleResolverListener());
    // High priority, because most other loadModule listeners will assume the
    // module's classes are available via autoloading
    $this->listeners[] = $events->attach(ModuleEvent::EVENT_LOAD_MODULE,
                                           new AutoloaderListener($options), 9000);
    $this->listeners[] = $events->attach(ModuleEvent::EVENT_LOAD_MODULE,
                                           new InitTrigger($options));
    $this->listeners[] = $events->attach(ModuleEvent::EVENT_LOAD_MODULE,
                                           new OnBootstrapListener($options));
    $this->listeners[] = $events->attach($locatorRegistrationListener);
    $this->listeners[] = $events->attach($configListener);
    return $this;
}
```



- **Important:** In order for paths relative to the application directory to work, the directive - chdir(dirname(__DIR__)); - must be contained within public/index.php

Slide 32

Module Best Practices

The ZF community makes the following recommendations for creating modules:

Keep `init()` and `onBootstrap()` light ...

Do not overload these methods with actions, as they run with each page request...

- Appropriate action would be to register event listeners Listeners
- Inappropriate to set up / configure instances of app resources, like DB connections, application logger, mailer
- Appropriate place to set up app resources is in `ServiceManager factories`
- These may be part of the given `Module class`
 - The method `getServiceConfiguration()` can return configuration for the `ServiceManager`
 - Factories can be defined as callbacks - but by defining a factory, you're offloading actual execution to when the service is requested



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 32

- Setting up and configuring instances of application resources are better done through the Service Manager

Module Best Practices

The ZF community makes the following recommendations for creating modules:

Do not perform writes within a module

- Never code a module to perform or expect any writes within a module's directory... the files within a directory should always exactly match the distribution
- User-provided configuration should be performed using overrides in the application-level configuration files
- As `chdir()` is in the `index.php`, you can simply specify `data/` for writes, as it will resolve to the application's data directory
- Two major advantages:
 - not allowing internal module writes retains compatibility with Phar packaging
 - keeping the module in sync with the upstream distribution updates via mechanisms like GIT, Pyrus and Composer are easy



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 33

Slide 34

Module Best Practices

The ZF community makes the following recommendations for creating modules:

Utilize a vendor prefix for module names

Prefix your module namespace with a vendor prefix

Ex:	<u>Module Namespace</u>	<u>Vendor Prefix</u>
	PhlyBlog	Phly
	EdpSuperLuminal	Edp
	OcraComposer	Ocra
	ZfcUser	Zfc

The module name should have a good semantic meaning... For example, "Blog" is easy to understand, but "Fury" is hard to interpret-- what does this module offer the user?



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 34

Slide 35

M5Ex1: MVC and Modules

- See Participant Exercises Instruction booklet for details...



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 35

Slide 1



The PHP Company

“ Module Six

Copyright © 2006-2013, Zend Technologies Inc.

Slide 2

Module Six Topics

- In this module, we will cover:
 - Controllers
 - Controller Plugins



The PHP Company

Copyright © 2006-2013, Zend Technologies, Inc. | 2

Slide 3

MVC Events and Controllers

- **Controllers are objects that implement Zend\Stdlib\DispatchableInterface**
- **There are two base controller classes for you to start with:**

`Zend\Mvc\Controller\AbstractActionController` allows routes to match an “action”

Once matched, a method named after the action is dispatched by the controller

Ex: if you had a route that returned “foo” for the “action” key, the “fooAction” method would be invoked

`Zend\Mvc\Controller\AbstractRestfulController` introspects the Request to determine what HTTP method was used, and calls a method accordingly



The PHP Company

Copyright © 2006-2013, Zend Technologies, Inc. | 3

- This means they need to implement a `dispatch()` method that takes minimally a `Request` object as an argument.

Controllers: Zend Skeleton App Example

```
<?php  
namespace Application\Controller;  
  
use Zend\Mvc\Controller\AbstractActionController;  
use Zend\View\Model\ViewModel;  
  
class IndexController extends AbstractActionController  
{  
    public function indexAction()  
    {  
        return new ViewModel();  
    }  
}
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 4

MVC-based Action Controllers

Controllers in the MVC implementation are objects that implement Zend\Stdlib\DispatchableInterface, which describes a single method

- In ZF2, this is primarily done by extending the AbstractActionController
- The AbstractActionController implements a series of interfaces that can provide controllers with additional capabilities

```
use Zend\Stdlib\DispatchableInterface,  
Zend\Stdlib\RequestInterface as Request,  
Zend\Stdlib\ResponseInterface as Response;  
  
class Foo implements DispatchableInterface  
{  
    public function dispatch(Request $request, Response $response = null)  
    {  
        // ... do something, and preferably return a Response ...  
    }  
}
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

| 5

- NOTE: As long as you extend the AbstractActionController, you do not need to implement the following interfaces... that is done for you by the Abstract class
- Here is the list of interfaces that the ActionController within MVC implements:

```
\Zend\Mvc\InjectApplicationEventInterface  
\Zend\Stdlib\DispatchableInterface  
\Zend\EventManager\EventManagerAwareInterface  
\Zend\ServiceManager\ServiceLocatorAwareInterface
```

AbstractActionControllers Interfaces

InjectApplicationEvent

- `Zend\Mvc\InjectApplicationEventInterface` hints to the Application instance it should inject its `MvcEvent` into the controller itself
- This is useful because the `MvcEvent` composes a number of objects: the Request and Response, and also the router, the route matches (a `RouteMatch` instance), and potentially the "result" of dispatching
- A controller that has the `MvcEvent` injected, then, can retrieve or inject these



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

| 6

- The `InjectApplicationEventInterface` defines simply two methods:

```
use Zend\EventManager\EventDescription as Event;  
  
public function setEvent(Event $event);  
public function getEvent($event);
```

Slide 7

AbstractActionControllers Interfaces

ServiceLocatorAwareInterface

- Generally, define your controllers such that dependencies are injected by the application's `ServiceManager`, via either constructor arguments or setter methods
- Controllers which extend `Zend\Mvc\Controller\AbstractActionController` will automatically implement `Zend\ServiceManager\ServiceLocatorAwareInterface`
- `ServiceLocatorAwareInterface` hints to the `ServiceManager` that it should inject itself into the controller... it defines one method:

```
use Zend\ServiceManager\ServiceManager;
use Zend\ServiceManager\ServiceLocatorAwareInterface;
// NOTE: ServiceManager implements ServiceLocatorAwareInterface
public function setServiceManager(ServiceLocatorAwareInterface $serviceManager);
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 7

- The main purpose for composing the service locator is actually for plugins that interact with the application workflow
- For example, the forward() plugin needs to be able to fetch another controller, and so needs access to the service manager and controller manager

Slide 8

Common Interfaces used with Controllers

EventManagerAwareInterface

- Use `EventManager` to tie into a controller's workflow without needing to extend or hardcode behavior into it
- `EventManagerAwareInterface` extends `EventsCapableInterface`
- Only need to implement the `EventManagerAwareInterface` (but you need to implement both methods)
- Define two methods: a setter and a getter
- The setter should also set any `EventManager` identifiers to listen on
 - The getter should return the `EventManager` instance

```
use Zend\EventManager\EventManagerAwareInterface;
use Zend\EventManager\EventManagerInterface;
use Zend\EventManager\EventsCapableInterface;

public function setEventManager(EventManagerInterface $events);
public function getEventManager();
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

| 8

- The `EventManagerAwareInterface` tells the `ServiceManager` to inject an `EventManager`
- The `EventsCapableInterface` tells other objects that this class has an accessible `EventManager` instance

Slide 9

Controller Plugins

There are a number of pre-built plugins available when you use:

- `AbstractActionController`
- `AbstractRestfulController`
- For custom plugins, if you define a method `setPluginManager()`, the controller will be injected with an instance of `Zend\Mvc\Controller\PluginManager`

Built-in Plugins

```
Zend\Mvc\Controller\Plugin\FlashMessenger  
Zend\Mvc\Controller\Plugin\Forward  
Zend\Mvc\Controller\Plugin\Params  
Zend\Mvc\Controller\Plugin\Redirect  
Zend\Mvc\Controller\Plugin\Url
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

| 9

- The `ActionController` and `RestController` have `__call()` implementations that allow you to retrieve plugins via method calls:

```
$plugin = $this->url();
```

Controller Plugins: FlashMessenger

The FlashMessenger Plugin is designed to create and retrieve self-expiring, session-based messages. It exposes a number of methods, some of which are listed here:

`addMessage ()`

adds a message to the current session namespace

`hasMessages ()`

determines if there are any flash messages in the current session namespace

`getMessages ()`

retrieves the flash messages from the current session namespace

`clearMessages ()`

clears all flash messages in current session namespace



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 10

- The complete list of FlashMessenger Plugin methods can be found in the ZF2 Reference Manual, Section 16.7.1

<http://packages.zendframework.com/docs/latest/manual/en/zend.mvc.controller-plugins.html#zend.mvc.controller-plugins.flashmessenger>

Slide 11

Controller Plugins: FlashMessenger Example

```
public function processAction()
{
    // ... do some work ...
    $this->flashMessenger()->addMessage('You are now logged in.');
    return $this->redirect()->toRoute('user-success');
}

public function successAction()
{
    $return = array('success' => true);
    $flashMessenger = $this->flashMessenger();
    if ($flashMessenger->hasMessages()) {
        $return['messages'] = $flashMessenger->getMessages();
    }
    return $return;
}
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 11

- The complete list of FlashMessenger Plugin methods can be found in the ZF2 Reference Manual, Section 16.7.1

<http://packages.zendframework.com/docs/latest/manual/en/zend.mvc.controller-plugins.html#zend.mvc.controller-plugins.flashmessenger>

Controller Plugins: Redirect

The Redirect Plugin performs the multiple steps required to complete redirects within the application

- The plugin provides two methods:

```
toRoute ($route, array $params = array(), array $options = array())
```

Redirects to a named route, using `$params` and `$options` to assemble the URL

```
toUrl ($url)
```

Simply redirects to the given URL

- In each case, the Response object is returned; if returned immediately, you can effectively short-circuit execution of the request

```
return $this->redirect()->toRoute ('login-success');
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 12

- To set up redirect operations manually, you would need to:
 - Assemble a URL using the router
 - Create and inject a "Location" header into the Response object, pointing to the assembled URL
 - Set the status code of the Response object to one of the 3xx HTTP statuses
- The Redirect plugin saves you all this work...
- Note: this plugin requires that the controller invoking it implements `InjectApplicationEvent`, and therefore has an `MvcEvent` composed, as it retrieves the router from the event object

Slide 13

Controller Plugins: Url

The Url Plugin provides a convenient way of generating URLs from route definitions within the controllers, to seed the view, generate headers, ...

```
$url = $this->url()->fromRoute('route-name', $params);
```

- The `fromRoute()` method is the only public method defined, and has the following signature:

```
public function fromRoute($route, array $params = array(),
                           array $options = array())
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 13

Controller Plugins: Accessing Parameters

The Params Plugin can be used within a controller to access routing parameters, as well as parameters originating from GET or POST

- `fromRoute(string $param, mixed $default = null)`
returns a parameter from a route match
- `fromQuery(string $param, mixed $default = null)`
returns a parameter from the URL (`$_GET`)
- `fromPost(string $param, mixed $default = null)`
returns a post parameter (`$_POST`)
- `fromFiles($name=null, $default=null)`
returns information from uploaded files (ex: `$_FILES`)
- `fromHeader($header=null, $default=null)`
returns header parameters

```
// examples using the params plugin
$match = $this->params()->fromRoute('admin');
$name = $this->params()->fromPost('name');
$page = $this->params()->fromQuery('page');
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 14

Accessing a Request & Response

When using `ActionController` or `RestfulController`, the request and response objects are composed directly into the controller, as soon as `dispatch()` is called

- You can access them in the following ways

```
// Using explicit accessor methods
$request = $this->getRequest();
$response = $this->getResponse();
```

- If the controller implements `InjectApplicationEventInterface` (as both `ActionController` and `RestfulController` do), you can also access these objects from the attached `MvcEvent`

```
$event = $this->getEvent();
$request = $event->getRequest();
$response = $event->getResponse();
```



Accessing Routing Parameters

The parameters returned when routing completes are wrapped in a `Zend\Mvc\Router\RouteMatch` object... this object is detailed in the section on routing

- Within your controller, if you implement `InjectApplicationEventInterface` (as both `ActionController` and `RestfulController` do), you can access this object from the attached `MvcEvent`
- Once you have the `RouteMatch` object, you can pull parameters from it

```
$event    = $this->getEvent();
$match = $event->getRouteMatch();
```

- There is a `params()` plugin that can be used to access GET and POST parameters, as well as provide easy access to routing parameters



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 16

Returning Early

You can effectively short-circuit execution of the application at any point by returning a Response from your controller or any event

- When such a value is discovered, it halts further execution of the EM, bubbling up to the Application instance, where it is immediately returned
- As an example, the Redirect plugin returns a Response, which can be returned immediately so as to complete the request as quickly as possible
- Other use cases might be for returning JSON or XML results from web service endpoints, returning "401 Forbidden" results, etc.



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 17

Registering module-specific listeners

You may want to use module-specific listeners as a simple and effective way to introduce authorization, logging, or caching into your application

Listeners

- Each Module class can have an optional `onBootstrap()` method
 - Typically, you'll do module-specific configuration, or setup event listeners, for your module here
 - The `onBootstrap()` method is called for every module on every page request and should only be used for performing lightweight tasks such as registering event listeners
- In ZF2, the provided base `Application` class has an `EventManager` associated with it, and once the modules are initialized, it triggers a "bootstrap" event, with a `getApplication()` method on the event



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 18

Registering module-specific listeners

So, one way to accomplish module-specific listeners is to listen to that event, and register listeners at that time

Listener

- Example:

```
namespace SomeCustomModule;
class Module
{
    public function onBootstrap($e)
    {
        $application = $e->getApplication();
        $view       = $application->getServiceManager()->get('View');

        $listeners   = new Listeners\ViewListener();
        $listeners->setView($view);
        $application->getEventManager()->attachAggregate($listeners);
    }
}
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 19

- The above demonstrates several things:

First, it demonstrates a listener on the application's "bootstrap" event - the onBootstrap() method

Second, it demonstrates that listener, and how it can be used to register listeners with the application

It grabs the Application instance; from the Application, it is able to grab the attached service manager and configuration

These are then used to retrieve the view, configure some helpers, and then register a listener aggregate with the application event manager

Slide 20

M6Ex1: Controllers and Plugins

- See Participant Exercises Instruction booklet for details...



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 20

Slide 1



The PHP Company

“ Module Seven ...”

Copyright © 2006-2013, Zend Technologies Inc.

Slide 2

Module Seven Topics:

- In this module, we will cover:
 - Routing Basics
 - TreeRouteStack
 - Routing Types



The PHP Company

Copyright © 2006-2013, Zend Technologies, Inc. | 2

Slide 3

Routing Basics

- Routing is the act of matching a request to a specific controller
- Typically, routing will examine the request URI, and attempt to match the URI path segment against provided constraints
 - If the constraints match, a set of “matches” are returned, one of which should be the controller name to execute
 - Other portions of the request URI or environment also can be used (ex: query parameters, headers, request method, ...)

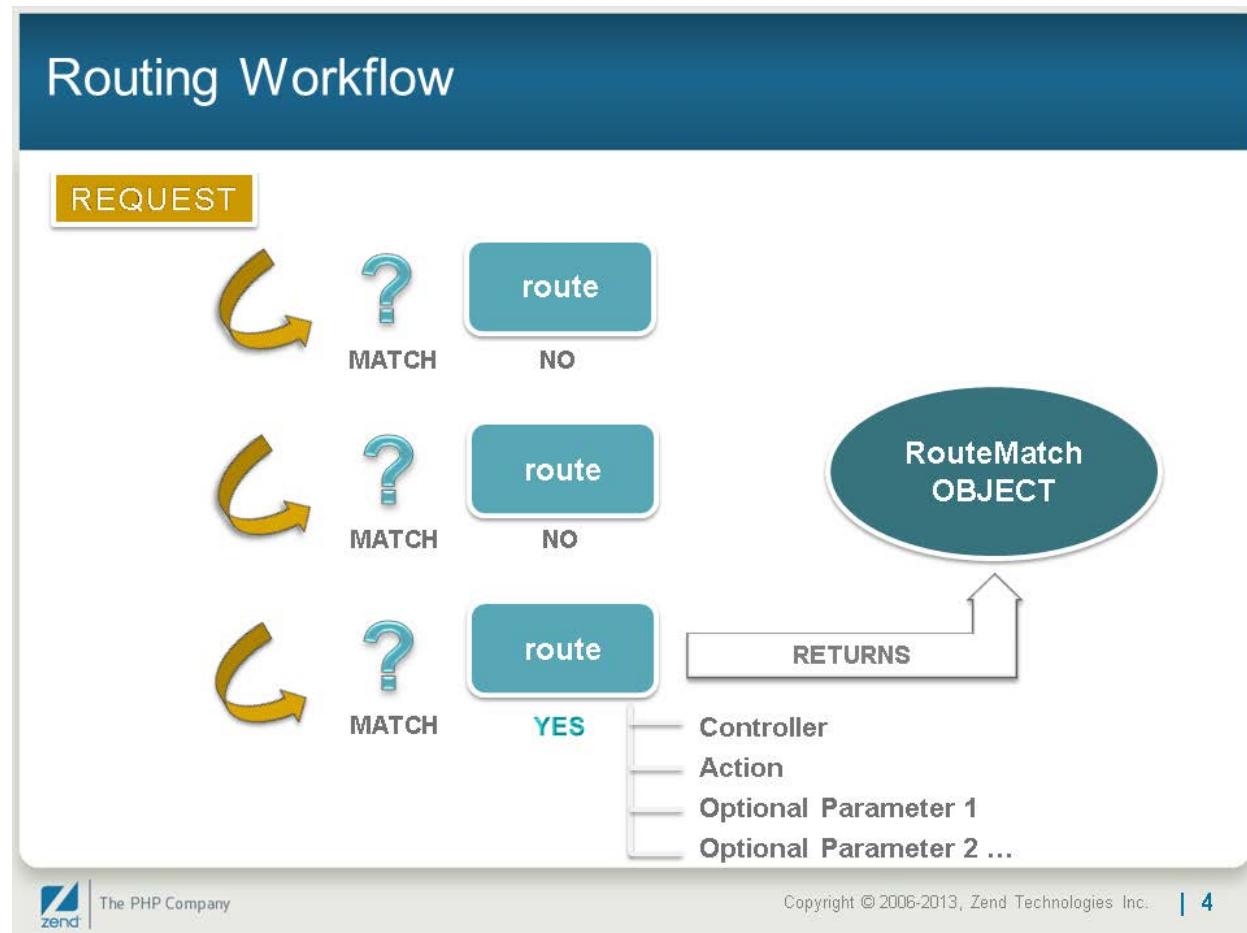


The PHP Company

Copyright © 2006-2013, Zend Technologies, Inc. | 3

- ZF1 Alert
 - Routing has been completely rewritten for ZF2. While execution is similar, the internal workings are more consistent, perform better, and are often simpler
 - Some of the old terminology does not apply in Zend Framework 2.x. In the new routing system we don't have a router as such, as every route can match and assemble URIs by themselves, which makes them routers as well
 - In most cases, this is something the developer does not need to worry about because Zend Framework 2.x takes care of this automatically

Slide 4



- The base unit of routing is a Route
- A Route accepts a Request and determines if it matches
- If so, it returns a RouteMatch object
- The Controller Params plugin has a `fromRoute()` method, which allows you to retrieve the values of specific parameters from the RouteMatch object

Slide 5

ZF2 Router Types: TreeRouteStack

- By default, Zend\Mvc uses the TreeRouteStack as the router
 - Zend\Mvc\Router\Http\TreeRouteStack registers trees of routes, and uses a B-tree algorithm to match routes; you register a single route with many children
- A TreeRouteStack consists of the following configuration:
 - A base "route", describing the base match needed (root of the tree)
 - An optional "route_plugins", a configured Zend\Mvc\Router\RoutePluginManager that can lazy-load routes
 - The option "may_terminate" can be set to TRUE, which informs the router it is permissible for the route match to stop at this point
 - An optional "child_routes" array, containing additional routes that stem from the base "route" (that is, build from it)
 - Each child route can itself be a TreeRouteStack if desired; in fact, the Part route works exactly this way



The PHP Company

Copyright © 2006-2013, Zend Technologies, Inc. | 5

Slide 6

ZF2 Router Types: TreeRouteStack

- When a route matches against a TreeRouteStack, the matched parameters from each segment of the tree will be returned
- A TreeRouteStack can be the sole route for the application, or can describe particular path segments of the application



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 6

Slide 7

ZF2 HTTP Route Types

ZF2 provides eight HTTP route types:

1. Zend\Mvc\Router\Http\Hostname

attempts to match the hostname registered in the request against specific criteria

2. Zend\Mvc\Router\Http\Literal

used for exact matching of the URI path

3. Zend\Mvc\Router\Http\Method

used to match the http method or ‘verb’ specified in the request

4. Zend\Mvc\Router\Http\Part

allows crafting a tree of possible routes based on segments of the URI path;
it actually extends the `TreeRouteStack`



The PHP Company

Copyright © 2006-2013, Zend Technologies, Inc.

| 7

Slide 8

ZF2 HTTP Route Types

ZF2 provides eight HTTP route types:

5. **Zend\Mvc\Router\Http\Regex**

utilizes a regular expression to match against the URI path... any valid regular expression is allowed

6. **Zend\Mvc\Router\Http\Scheme**

matches the URI scheme only (must be an exact match); takes what you want to match and the “defaults”, parameters to return on a match (like Literal)

7. **Zend\Mvc\Router\Http\Segment**

allows matching any segment of a URI path; segments are denoted using a colon, followed by alphanumeric characters

8. **Zend\Mvc\Router\Http\Query**

allows you to specify and capture query string parameters for a given route



The PHP Company

Copyright © 2006-2013, Zend Technologies, Inc.

| 8

Slide 9

Code Example: Zend\Mvc\Router\Http\Hostname

```
// from the module.config.php file under the key 'router' => 'routes'  
'subdomain-post' => array(  
    'type' => 'Zend\Mvc\Router\Http\Hostname',  
    'options' => array(  
        'route' => 'post.onlinemarket.com',  
        'defaults' => array(  
            'controller' => 'market-post-controller',  
            'action'      => 'index',  
        ),  
    ),  
) ,
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

| 9

Code Example: Zend\Mvc\Router\Http\Literal

```
// from the module.config.php file under the key 'router' => 'routes'  
'market-post' => array(  
    'type' => 'Zend\Mvc\Router\Http\Literal',  
    'options' => array(  
        'route' => '/market/post',  
        'defaults' => array(  
            'controller' => 'market-post-controller',  
            'action'      => 'index',  
        ),  
    ),  
) ,
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 10

Code Example: Zend\Mvc\Router\Http\Method

```
// from the module.config.php file under the key 'router' => 'routes'  
'market-delete-confirm' => array(  
    'type' => 'Zend\Mvc\Router\Http\Method',  
    'options' => array(  
        'verb' => 'post,put',  
        'defaults' => array(  
            'controller' => 'market-delete-controller',  
            'action'      => 'delete-confirm',  
        ),  
    ),  
) ,
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 11

Code Example: Zend\Mvc\Router\Http\Part

```
// from the module.config.php file under the key 'router' => 'routes'
'market-item' => array(
    'type'      => 'Literal',
    'options'   => array(
        'route'    => '/market/item',
        'defaults' => array(
            'controller' => 'market-view-controller',
            'action'      => 'item',
        ),
    ),
    'may_terminate' => true,
    'child_routes' => array(
        'default' => array(
            'type'      => 'Segment',
            'options'   => array(
                'route'    => '[/:id][]{/}',
                'constraints' => array(
                    'id'       => '[0-9]*',
                ),
            ),
        ),
    ),
),
)
```



Code Example: Zend\Mvc\Router\Http\Regex

```
// from the module.config.php file under the key 'router' => 'routes'  
'market-view-item' => array(  
    'type'      => 'Regex',  
    'options'   => array(  
        'regex'     => '/view/ (?<id>[0-9]+)',  
        'spec'      => '/view/%id%',  
        'defaults'  => array(  
            'controller' => 'market-view-controller',  
            'action'       => 'item',  
            'id'           => 1,  
        ),  
    ),  
) ,
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 13

Code Example: Zend\Mvc\Router\Http\Scheme

```
// from the module.config.php file under the key 'router' => 'routes'  
'market-purchase' => array(  
    'type' => 'Zend\Mvc\Router\Http\Scheme',  
    'options' => array(  
        'scheme' => 'https',  
        'defaults' => array(  
            'https'      => true,  
            'controller' => 'market-purchase-controller',  
            'action'      => 'index',  
        ),  
    ),  
) ,
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 14

Code Example: Zend\Mvc\Router\Http\Segment

```
// from the module.config.php file under the key 'router' => 'routes'  
'market-general' => array(  
    'type'  => 'Zend\Mvc\Router\Http\Segment',  
    'options' => array(  
        'route' => '/[:controller][][][:action]',  
        'constraints' => array(  
            'controller' => '[a-zA-Z][a-zA-Z0-9_-]+',  
            'action'      => '[a-zA-Z][a-zA-Z0-9_-]+',  
        ),  
        'defaults' => array(  
            'controller' => 'market-index-controller',  
            'action'      => 'index',  
        ),  
    ),  
)
```



Code Example: Zend\Mvc\Router\Http\Query

```
// from the module.config.php file under the key 'router' => 'routes'  
'market-item-query' => array(  
    'type'      => 'Zend\Mvc\Router\Http\Segment',  
    'options'   => array(  
        'route'     => '/market/item[/:id]',  
        'defaults'  => array(  
            'controller' => 'market-view-controller',  
            'action'      => 'item',  
        ),  
    ),  
    'may_terminate' => true,  
    'child_routes' => array(  
        'query' => array(  
            'type'      => 'Zend\Mvc\Router\Http\Query',  
            'options'   => array(  
                'defaults' => array(),  
            ),  
        ),  
    ),  
,  
),  
// in Market/view/market/view/item.phtml  
<a href="php echo $this-&gt;url('market-item-query/query',<br/    array('id' => $random[rand(1, count($random)-1)],  
    'name' => 'TEST')) ; ?>">View Random Item</a>
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 16

Slide 17

M7Ex1: Routing

- See Participant Exercises Instruction booklet for details...



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 17

Slide 1



“ Module Eight

Copyright © 2006-2013, Zend Technologies Inc.

Slide 2

Module Eight Topics

- In this module, we will cover:
 - View Layer
 - Layouts
 - PHP Renderer
 - View Helpers



The PHP Company

Copyright © 2006-2013, Zend Technologies, Inc. | 2

Slide 3

Zend\View

`Zend\View` provides the "View" layer of Zend Framework's MVC system

- It is a multi-tiered system allowing a variety of mechanisms for extension, substitution, ...
- **View Layer** components:
 - **Variables containers**... hold *variables* and *callbacks* that you wish to represent in the view; often, a Variables container will also provide mechanisms for context-specific escaping of variables and more
 - **View Models**... hold *Variables containers*, specify the template to use, if any, and optionally provide rendering options; View Models may be nested in order to represent complex structures
 - **Renderers**... take *View Models* and provide a representation of them to return. ZF2 includes three renderers by default: (1) a "PHP" renderer, which utilizes PHP templates in order to generate markup; (2) a JSON renderer; (3) a Feed renderer, capable of generating RSS and Atom feeds

[... more ...](#)



The PHP Company

Copyright © 2006-2013, Zend Technologies, Inc. | 3

Slide 4

Zend\View

- **View Layer** components:
 - **Resolvers**... resolve a *template name* to a **resource** that a **renderer** may consume; as an example, a resolver may take the name "blog/entry" and resolve it to a PHP view script
 - **The View**... consists of *strategies* that map the current **request** to a **renderer**, and strategies for injecting the **response** with the result of rendering
 - **Renderer and Response Strategies**... *Renderer Strategies* listen to the "**renderer**" event of the View, and decide which **renderer** should be selected, based on the **request** or other criteria;

Response strategies are used to inject the **response** object with the results of **rendering** – this may also include taking actions such as setting **Content-Type** headers



The PHP Company

Copyright © 2006-2013, Zend Technologies, Inc.

| 4

Slide 5

Zend\View Configuration

The "out-of-the-box" configuration will work in most cases, with some additional steps...

- You must both **select** and **configure** resolver strategies
- You may want to **specify alternate template names** (Ex: site layout, error pages)



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

| 5

- Section 20.1.2 Usage provides a complete code example for the View configuration

Slide 6

View

View is a relatively simple component – it summons a ViewEvent, then triggers two subsequent events – renderer and response

- Strategies can then be added to select a renderer, as well as manipulate the response (covered shortly...)

Manual Use

Create a `ViewModel` instance in your controller,

Set any variables you want to pass to the renderer,

Specify the template name to use,

Return the instance



The PHP Company

Copyright © 2006-2013, Zend Technologies, Inc.

| 6

View Model

Automated Use

When the template name is based on the action and controller, and you are only passing variables, ZF2 provides an automated process for this, based on registering two listeners for controllers

Listener

Listener 1: Is an associative array returned from your controller?



If yes, it creates a view model and injects this associative array as the view variables container; the view model replaces the MVC event's result

If empty/null, it creates a view model without any variables attached; the view model replaces the MVC event's result

Listener 2: Is the MVC event result a view model, and if so, does it have a template associated with it?



If yes, no action needed

If no, it uses the module namespace, controller name, and (if available), the action route parameter to create the template name



The PHP Company

Copyright © 2006-2013, Zend Technologies, Inc.

| 7

Controllers and View Models: Code Example

```
<?php
namespace Guestbook\Controller;
use Zend\Mvc\Controller\AbstractActionController;
use Zend\Stdlib\ResponseInterface as Response;
use Zend\View\Model\ViewModel;
class IndexController extends AbstractActionController
{
    public function indexAction()
    {
        $prg      = $this->prg();
        $entryForm = $this->getServiceLocator()->get('guestbook_entry_form');
        $entryService = $this->getServiceLocator()-
            >get('guestbook_entry_service');
        if ($prg instanceof Response) {
            return $prg;
        } elseif ($prg !== false) {
            $entry = $entryService->add($prg);
            if ($entry) {
                return $this->redirect()->toRoute('guestbook');
            }
        }
        return new ViewModel(array(
            'entryForm' => $entryForm,
            'entries'   => $entryService->findAll()
        ));
    }
}
```



Layouts

The site layout includes the: (1) default stylesheets; (2) any necessary JavaScript; (3) basic markup structure into which all site content is injected

- In ZF2, layouts are composed of nested view models
 - The MVC event composes a View Model that acts as a "container" for the nested models, and represents the layout template for the site
 - All other content is then rendered and captured to view variables for this central view model
- The default rendering strategy sets the layout template as "layout"; this can be changed within the configuration for the Dependency Injector
- `Zend\Mvc\View\InjectViewModelListener`, a listener on a controller, will take the view model returned by a controller and inject it as a child to the container view model
 - View models capture the "content" variable of the container view model (default)
 - To specify a different view variable to capture, explicitly create a view model in your controller, and set the "captureTo" value using the `captureTo()` method



The PHP Company

Copyright © 2006-2013, Zend Technologies, Inc. | 9

- The layout template is referred to as "layout" in the view manager configuration

Slide 10

Alternate Rendering & Response Strategies

Rendering & Response Strategies

- A "**renderer strategy**" examines the Request object (or other criteria) to select an appropriate renderer (which may be none...)
- A "**response strategy**" determines how to populate the **response**, based on the rendering result
- The methods `addRendererStrategy()` and `addResponseStrategy()` are used to attach strategies to the two events, "renderer" and "response"



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 10

Slide 11

Alternate Rendering & Response Strategies

ZF2 includes three renderer/response strategies:

1. `Zend\View\Strategy\PhpRendererStrategy`

This strategy is a "catch-all" in that it will always return the `Zend\View\Renderer\PhpRenderer` and populate the response body with the results of rendering

2. `Zend\View\Strategy\JsonStrategy`

This strategy returns the `Zend\View\Renderer\JsonRenderer` if the view model is a `JsonModel` instance

3. `Zend\View\Strategy\FeedStrategy`

This strategy returns the `Zend\View\Renderer\FeedRenderer` if the view model is a `FeedModel` instance



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 11

- Note: Only `Zend\View\Strategy\PhpRendererStrategy` is registered by default... you need to register the other strategies manually to use them

Registering Multiple Strategies: Example

```
namespace Application;
use Zend\EventManager\StaticEventManager;
class Module
{
    public function init($manager)
    {
        // Register a bootstrap event
        $events = StaticEventManager::getInstance();
        $events->attach('bootstrap', 'bootstrap', array($this, 'bootstrap'));
    }
    public function bootstrap($e)
    {
        // Register a "render" event, at high priority
        $app = $e->getParam('application');
        $app->getEventManager() ->attach('render', array($this, 'registerJsonStrategy'), 100);
    }
    public function registerJsonStrategy(MvcEvent $e)
    {
        $locator = $e->getTarget() ->getServiceManager();
        $view = $locator->get('Zend\View\View');
        $jsonStrategy = $locator->get('ViewJsonStrategy');
        $view->getEventManager() ->attach($jsonStrategy, 100);
        // NOTE: alternative approach, in module.config.php:
        // 'view_manager'=>['strategies'=>['ViewJsonStrategy', 'ViewFeedStrategy', ...],]
    }
}
```



- Note the priority values -- these are "pre" listeners, as they are high priority, both for the application "render" event, and for the view events

PHP Renderer: Usage

`Zend\View\Renderer\PhpRenderer` renders view scripts written in PHP, capturing and returning the output

- It composes:
 - Variable containers and/or View Models
 - A plugin manager for helpers
 - Optional filtering of the captured output
- The `PhpRenderer` is independent of template system - you can use PHP as your template language, or create instances of other template systems and manipulate them within your view script
- It is recommended you create renderers and strategies for any template system you want to use outside of the `PhpRenderer`



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 13

- Example: ZfcTwig

PHP Renderer: Usage

- Basic usage consists of:

- instantiating / obtaining an instance of the PhpRenderer
- providing it with a resolver which will resolve templates to PHP view scripts
- calling its `render()` method

- Instantiating a renderer is easy:

```
use Zend\View\Renderer\PhpRenderer;  
$renderer = new PhpRenderer();
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 14

PHP Renderer: Usage

ZF2 includes several types of "resolvers", which are used to resolve a template name to a resource a renderer can consume

- Commonly used with the `PhpRenderer` are: `Zend\View\Resolver*`
 - `\TemplateMapResolver`
simply maps template names directly to view scripts
 - `\TemplatePathStack`
creates a LIFO stack of script directories in which to search for a view script
 - By default, it appends the suffix ".phtml" to the requested template name, and then loops through the script directories;
 - If it finds a file matching the requested template, it returns the full file path
 - `\AggregateResolver`
allows attaching a FIFO queue of resolvers to consult
- **By default, the `AggregateResolver` is used, and composes first the `TemplateMapResolver` and then the `TemplatePathStack`**
 - This provides optimal performance



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 15

PHP Renderer: Options & Configuration

Zend\View\Renderer\PhpRenderer utilizes several collaborators in order to do its work... use the following methods to configure the renderer

HelperPluginManager

```
setHelperPluginManager(string|HelperPluginManager $helpers); also "get"
```

resolver

```
setResolver(Zend\View\Resolver $resolver);
```

filters

```
setFilterChain(Zend\Filter\FilterChain $filters);
```

canRenderTrees

```
setCanRenderTrees(bool $canRenderTrees);
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 16

- Methods for configuring the renderer:

setBroker(Zend\View\HelperBroker \$broker);

Set the broker instance used to load, register, and retrieve helpers

resolver setResolver(Zend\View\Resolver \$resolver);

Set the resolver instance

filters setFilterChain(Zend\Filter\FilterChain \$filters);

Set a filter chain to use as an output filter on rendered content

canRenderTrees setCanRenderTrees(bool \$canRenderTrees);

Set flag indicating whether or not we should render trees of view models

If set to true, the Zend\View\View instance will not attempt to render children separately, but instead pass the root view model directly to the PhpRenderer. It is then up to the developer to render the children from within the view script. This is typically done using the RenderChildModel helper:
\$this->renderChildModel('child_name')

PHP Renderer: Additional Methods

You usually only access variables and helpers within your view scripts or when interacting with the `PhpRenderer`

- There are a few additional methods you may be interested in:

render

```
render(string|Zend\View\Model $nameOrModel, $values = null);
```

resolver

```
resolver();
```

plugin

```
plugin(string $name, array $options = null);
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 17

- Some PHP renderer additional methods:

render render(string|Zend\View\Model \$nameOrModel, \$values = null);
Render a template/view model;

- If `$nameOrModel` is a string, it is assumed to be a template name; that template will be resolved using the current resolver, and then rendered; If `$values` is non-null, those values, and those values only, will be used during rendering, and will replace whatever variable container previously was in the renderer; However, the previous variable container will be reset when done;
- If `$values` is empty, the current variables container (see `setVars()`) is injected when rendering
- If `$nameOrModel` is a Model instance, the template name will be retrieved from it and used;
- Additionally, if the model contains any variables, these will be used when rendering; otherwise, the variables container already present, if any, will be used

addTemplate addTemplate(string \$template);
Add a template to the stack

- When used, the next call to `render()` will loop through all template added using this method, rendering them one by one; the output of the last will be returned

PHP Renderer View Scripts

Once you call `render()`, `Zend\View\Renderer\PhpRenderer` then includes the requested view script and executes it "inside" the scope of the `PhpRenderer` instance

- References to `$this` in your view scripts actually point to the `PhpRenderer` instance itself
- Variables assigned to the view - either via a view model, variables container, or simply by passing an array of variables to `render()` - may be retrieved in three ways
 1. Explicitly, retrieving them from the variables container composed in the `PhpRenderer`:
`$this->vars()->varname`
 2. As instance properties of the `PhpRenderer` instance: `$this->varname`
In this situation, instance property access is simply proxying to the composed variables instance
 3. As local PHP variables: `$varname`
The `PhpRenderer` extracts the members of the variables container locally



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 18

- It is generally recommended that you use the second notation, as it's less verbose than the first, but differentiates between variables in the view script scope and those assigned to the renderer from elsewhere

Slide 19

View Helpers

Helper (plugin) classes can be used to perform complex, repetitive functions with a view script; ZF2 comes with a large number of helpers

- A helper is simply a class that implements the interface `Zend\View\Helper\HelperInterface`
- Helper defines two methods:
 - `setView()` accepts a `Zend\View\Renderer` instance/implementation
 - `getView()` is used to retrieve that instance
- `Zend\View\Renderer\PhpRenderer` creates a `HelperPluginManager` instance, from which it pulls helpers upon request
- Register new helpers by adding them to the plugin manager



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 19

Included View Helpers

Zend Framework comes with an initial set of helper classes, most of which relate to form element generation and perform the appropriate output escaping automatically

- There are helpers for creating route-based URLs and HTML lists, as well as declaring variables
- There is also a rich set of helpers for providing values for, and rendering, the various HTML <head> tags, such as HeadTitle, HeadLink, and HeadScript
- The following are some of the more commonly used helpers:

Action View Helper	HeadLink Helper
BasePath Helper	HeadMeta Helper
Placeholder Helper	HeadStyle Helper
Doctype Helper	HeadTitle Helper
HTML Object Helper	Inline Script Helper



- A complete list of all included helpers is available within the ZF2 Reference Manual (section 20.4.2):

<http://framework.zend.com/manual/2.0/en/modules/zend.view.helpers.html>

Invoking View Helpers

```
// $view is a PhpRenderer instance

// Via the plugin manager:
$plugins = $view->getHelperPluginManager();
$helper  = $plugins->get('lowercase');

// Retrieve the helper instance, via the method "plugin",
// which proxies to the plugin broker:
$helper = $view->plugin('lowercase');

// If the helper does not define __invoke(), the following also retrieves it:
$helper = $view->lowercase();

// If the helper DOES define __invoke, you can call the helper as if it were
// a method:
$filtered = $view->lowercase('some value');
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 21

View Helpers: Escaping Output

Escaping output is an important task to perform in a view script for a variety of reasons, including security

- Unless you are using a function, method, or helper that does escaping on its own, you should always escape variables when you output them
 - Context-specific escaping is critical for security, as HTML escaping does not work for CSS, JS, or even HTML attributes; URL escaping is also context specific
 - Zend\View includes a class Zend\View\Helper\Escaper\AbstractHelper, which is extended by a series of useful escape view helpers... ultimately using Zend\Escaper\Escaper to do the work
 - The `escapeHtml()` method uses the `htmlspecialchars()` function for escaping, using UTF-8 as the charset, and the ENT_QUOTES setting
- ```
// BAD view-script practice:
echo $this->variable;
```

```
// GOOD view-script practice:
echo $this->escapeHtml($this->variable);
```
- Other escape view helpers include: `escapeCss()`, `escapeHtmlAttr()`, `escapeJs()`, and `escapeUrl()`



## View Helpers: Escaping Output Code Example

```
// escapeHtml() example
<?php echo $this->escapeHtml('<bad>tag</bad>!@#$%^&') ; ?>
// produces this output:
// <bad>tag</bad>!@#$%^&

// escapeJs() example
<?php echo $this->escapeJs('<script>alert("hello") ;</script>') ; ?>
// produces this output:
// \x3Cscript\x3Ealert\x28\x22hello\x22\x29\x3B\x3C\x2Fscript\x3E

// escapeCss() example
<?php echo $this->escapeCss('<style>b { color: red;}</style>') ; ?>
// produces this output:
// \3C style\3E b\20 \7B \20 color\3A \20 red\3B \7D \3C \2F style\3E

// escapeUrl() example
<?php echo $this->escapeUrl('http://zend.com/name=Doug&id=12345') ; ?>
// produces this output:
// http%3A%2F%2Fzend.com%2Fname%3DDoug%26id%3D12345
```



## Registering Concrete Helpers

At times, it is convenient to instantiate a view helper, and then register it with the renderer ... this can be done by injecting it into the `HelperPluginManager`

```
// Guestbook\Service\HtmlTableRowHelper
class HtmlTableRowHelper extends AbstractHtmlElement
{
 public function __invoke(array $items)
 {
 $output = '';
 foreach ($items as $value) {
 $output .= '<td>' . htmlspecialchars($value) . '</td>';
 }
 return '<tr>' . $output . '</tr>' . PHP_EOL;
 }
}
// in module.config.php
'view_helpers' => array(
 'invokables' => array(
 'htmlTableRow' => 'Guestbook\Service\HtmlTableRowHelper',
),
),
// in the view script
<?php echo $this->htmlTableRow(array($entry->getName(),
$entry->getEmail(), $entry->getWebsite(), $entry->getMessage()));?>
```



Slide 25

## M8Ex1: View Layer

- See Participant Exercises Instruction booklet for details...



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 25

Slide 1



The PHP Company

## “ Module Nine . . .

Copyright © 2006-2013, Zend Technologies Inc.

Slide 2

## Module Nine Topics:

- In this model, we will cover:
  - Zend\InputFilter
  - Zend\Validator
  - Zend\Filter
  - Zend\Form
  - Zend\Form\View\Helper
  - Zend\Stdlib\Hydrator



The PHP Company

Copyright © 2006-2013, Zend Technologies, Inc. | 2

Slide 3

## Zend\InputFilter\Input

The `Zend\InputFilter` component family can be used to sanitize or otherwise transform input data

- For instance, you could use it to filter `$_GET` or `$_POST` values, CLI arguments, ...
- The `Zend\InputFilter` family consists of two primary subcomponents:
  - (1) `Zend\InputFilter\Input`
  - (2) `Zend\InputFilter\InputFilter`
- `InputFilter` is a container class to which you can add a sequence of `Input` and `InputFilter` objects
  - To pass input data to the `InputFilter`, you can use the `setData()` method
  - The data is specified using an associative array or traversable object
    - Ex: how to validate the data coming from an HTML form using the `POST` method



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 3

- As with validators (covered next), filters are often used with `Zend\InputFilter\input` objects
- `Zend\Filter` includes sub-components which specialize in encryption, compression, file manipulation, and word transformation
- Core filters include: Alnum, Alpha, BaseName, Boolean, Callback, Compress, Decompress, Decrypt, Digits, Dir, Encrypt, FilterBroker, FilterChain, FilterInterface, FilterLoader, HtmlEntities, Inflector, InputFilter, Int, LocalizedToNormalized, NormalizedToLocalised, Null, PregReplace, RealPath, StaticFilter, StringToLower, StringToUpper, StringTrim, StripNewlines, and StripTags

Slide 4

## Zend\Validator

`Zend\Validator` is a component family that can be used to determine if an item meets certain criteria

- `Zend\Validator` components are often used in conjunction with `Zend\InputFilter` objects
- Many validators are under other namespaces, such as `Zend\I18n`
- Core validators include (but are not limited to):

Core Validators

|                       |                           |
|-----------------------|---------------------------|
| <code>Alnum</code>    | <code>EmailAddress</code> |
| <code>Barcode</code>  | <code>Hostname</code>     |
| <code>Callback</code> | <code>InArray</code>      |
| <code>Csrf</code>     | <code>Regex</code>        |
| <code>Date</code>     | <code>StringLength</code> |



The PHP Company

Copyright © 2006-2013, Zend Technologies, Inc. | 4

- These are other core validators:

`Alpha`, `Between`, `CreditCard`, `Digits`, `Float`, `GreaterThan`, `Hex`, `Iban`, `Identical`, `Int`, `Ip`, `Isbn`, `LessThan`, `NotEmpty`, `PostCode`, `StaticValidator`,

Slide 5

## Zend\Validator: Barcode, Db, and File

The Zend\Validator\Barcode component family includes these validators:

|         |                 |
|---------|-----------------|
| Codabar | Identcode       |
| Code*   | Intelligentmail |
| Ean*    | Royalmail       |
| Gtin*   |                 |

- The Zend\Validator\Db component family includes:

|              |                |
|--------------|----------------|
| RecordExists | NoRecordExists |
|--------------|----------------|

- The Zend\Validator\File component family includes:

|                  |           |
|------------------|-----------|
| Count            | Hash      |
| Crc32            | Md5       |
| ExcludeExtension | MimeType  |
| Exists           | NotExists |
| Extension        | Size      |
| FileSize         | Upload    |



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

| 5

- Other Zend\Validator\Barcode validators include:

Code128, Code25, Code25interleaved, Code39, Code39ext, Code93, Code93ext  
Ean12, Ean13, Ean14, Ean18, Ean2, Ean5, Ean8  
Gtin12, Gtin13, Gtin14  
Issn, Itf14, Leitcode, Planet, Postnet, Sscc, Upca, Upce

- Other Zend\Validator\File component family includes:

ExcludeMimeType, ImageSize, IsCompressed, IsImage, Sha1, WordCount

## Validator Methods

You can add one or more validators to each input using the `addValidator()` method for each validator

- When working with instances of the `Zend\InputFilter\Input` class you can obtain its validator chain and call `addValidator()` to add validators to the chain (an example of this is shown on the following slide)
- When working with a `Zend\InputFilter\InputFilter` instance, it is also possible to specify a "validation group", a subset of the data to be validated; this may be done using the `setValidationGroup()` method
- You can specify the list of input names as an array or individual parameters

```
// As individual parameters
$filterInput->setValidationGroup('email', 'password');

// or as an array of names
$filterInput->setValidationGroup(array('email', 'password'));
```



- `addValidator()` belongs to `Zend\Validator\ValidatorChain`, which an `Input` composes
- `setValidationGroup()` belongs to `Zend\InputFilter\InputFilterInterface`
- In this example, the email and password values are validated
- The email must be a valid address and the password must be composed of at least 8 characters
- If the input data are not valid, ZF2 reports the list of invalid input using the `getInvalidInput()` method

Slide 7

## Zend\InputFilter

```
use Zend\InputFilter\InputFilter;
use Zend\InputFilter\Input;
use Zend\Validator;

$email = new Input('email');
$email->getValidatorChain()
 ->addValidator(new Validator\EmailAddress());

$password = new Input('password');
$password->getValidatorChain()
 ->addValidator(new Validator\StringLength(8));

$inputFilter = new InputFilter();
$inputFilter->add($email)
 ->add($password)
 ->setData($_POST);

if ($inputFilter->isValid()) {
 echo "The form is valid\n";
} else {
 echo "The form is not valid\n";
 print_r ($inputFilter->getMessages());
}
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

| 7

Slide 8

## Filtering

You can validate and/or filter the data using `InputFilter`

- To filter data, use the `getFilterChain()` method of individual `Input` instances, and attach filters to the returned filter chain
- The `getValue()` method returns the *filtered* value of the 'foo' input, while `getRawValue()` returns the *original* value

```
use Zend\InputFilter\Input;
use Zend\InputFilter\InputFilter;

$input = new Input('foo');
$input->getFilterChain()
 ->attachByName('stringtrim')
 ->attachByName('alpha');
$inputfilter = new InputFilter();
$inputfilter->add($input, 'foo')
 ->setData(array(
 'foo' => 'Bar3 '
));
echo "Before:\n";
echo $inputFilter->getRawValue('foo') . "\n"; // the output is 'Bar3'
echo "After:\n";
echo $inputFilter->getValue('foo') . "\n"; // the output is 'Bar'
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

| 8

- Note that filtering in the context of the `InputFilter` is more accurately "data normalization"
- Example: Filtering without validation

Slide 9

## Zend\InputFilter\Factory

**Zend\InputFilter\Factory provides initialization of the InputFilter based on a configuration array (or Traversable object)**

- Below is an example that creates a password input value with the constraint that the string is at least 8 characters

```
use Zend\InputFilter\Factory;
$factory = new Factory();
$inputFilter = $factory->createInputFilter(array(
 'password' => array(
 'name' => 'password',
 'required' => true,
 'validators' => array(
 array(
 'name' => 'notempty',
),
 array(
 'name' => 'stringlength',
 'options' => array(
 'min' => 8
),
),
),
),
));
$inputFilter->setData($_POST);
echo $inputFilter->isValid() ? "Valid form" : "Invalid form";
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 9

- The factory may be used to create not only Input instances, but also nested InputFilters, allowing you to create validation and filtering rules for hierarchical data sets
- The standard InputFilter class is actually backed by a Factory instance
- It allows you to call add(), with a specification just like that shown on this slide, in order to create and attach an Input (or InputFilter, if you have a "type" key that specifies a class implementing the InputFilterInterface)

## Zend\Form

`Zend\Form` is intended primarily as a bridge between your domain models and the View Layer

- It composes a thin layer of objects representing form elements, an `InputFilter`, and methods for binding data to/from the form and attached objects
- The component consists of:
  - `Elements` that consist of a name and attributes (also specialized elements)
  - `Fieldsets` that extend from `Elements`, but allow composing other fieldsets and elements as well as data binding
- Forms, which extend from Fieldsets (>> Elements), provide data and object binding, and compose `InputFilters`; use `Zend\Stdlib\Hydrator` for data binding
- To facilitate usage with the view layer, the `Zend\Form` component also aggregates a number of form-specific view helpers
- These accept elements, fieldsets, and/or forms, and use the attributes they compose to render markup



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 10

- A small number of specialized elements are provided for accomplishing application-centric tasks, such as multcheckbox or radio elements
- These include the `Csrf` element, used to prevent Cross Site Request Forgery attacks, and the `Captcha` element, used to display and validate CAPTCHAs
- Finally, a Factory is provided to facilitate creation of elements, fieldsets, forms, and the related input filter
- The default Form implementation is backed by a factory to facilitate extension and ease the process of form creation

Slide 11

## Zend\Form

### Forms are relatively easy to create

- At a bare minimum, each element or fieldset requires a name; typically, you'll also provide some attributes to inform the view layer how it might render the item
- The form itself will also include an `InputFilter`, which is required for form validation; you can conveniently create one directly in the form via a factory
- Individual elements can hint as to what defaults to use when generating a related input for the input filter
- Form validation encompasses two methods:
  - `setData()` used to provide the data to validate
  - `isValid()` used to perform validation using the provided data and `InputFilter`
- If you want to simplify your work even more, you can bind an object to the form; on successful validation, it will be populated from the validated values



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 11

Slide 12

## Zend\Stdlib\Hydrator\\*

When assigning values to form elements, Zend\Form\Form::setData() uses the `ArraySerializable` hydrator

- The `Zend\Stdlib\Hydrator` component family is designed to provide a means of populating properties of an object
- The `hydrate()` method is used to set object properties. Conversely, the `extract()` method is used to retrieve object properties.
- Here is a summary of the members of this component family and how hydration and extraction are accomplished

| Zend\Stdlib\Hydrator\...       | hydrate()                                                          | extract()                          |
|--------------------------------|--------------------------------------------------------------------|------------------------------------|
| <code>ArraySerializable</code> | <code>exchangeArray()</code> or<br><code>populate()</code> methods | <code>getArrayCopy()</code> method |
| <code>ClassMethods</code>      | “set” methods                                                      | “get” methods                      |
| <code>ObjectProperty</code>    | public properties                                                  | public properties                  |
| <code>Reflection</code>        | PHP Reflection Object                                              | PHP Reflection Object              |



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 12

- `Zend\Db\ResultSet` also has a ‘HydratingResultSet’ that will accept a ‘Hydrator’ and object prototype.
- This makes it simple to wire your domain entities to forms and persistence; the ‘Hydrator’ component then serves as a lightweight data mapper

Slide 13

## Zend\Form: Form Creation

```
use Zend\Captcha;
use Zend\Form\Element;
use Zend\Form\Fieldset;
use Zend\Form\Form;
use Zend\InputFilter\Input;
use Zend\InputFilter\InputFilter;

$name = new Element('name');
$name->setAttribute('type', 'text');
$name->setLabel('Your name');

$email = new Element\Email('email');
$email->setAttribute('class', 'emailCss');
$email->setLabel('Your email address');

$subject = new Element('subject');
$subject->setAttributes(array('type' => 'text', 'class' => 'subjectCss'));
$subject->setLabel(array('label' => 'Subject'));

$message = new Element('message');
$message->setAttributes(array('type' => 'textarea'));
$message->setLabel(array('label' => 'Message'));
);

// Continued on next page
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 13

- There are components for Captcha, Csrf, Date, DateTime, DateTimeLocal, Email, Month, MultiCheckbox, Number, Radio, Range, Time, Url, and Week

(ex: Zend\Form\Element\Captcha, Zend\Form\Element>Email, ...)

Slide 14

## Zend\Form: Form Creation

```
$captcha = new Element\Captcha('captcha');
$captcha->setCaptcha(new Captcha\Dummy());
$captcha->setOptions(array('label' => 'Please verify you are human'));

$csrf = new Element\Csrf('security');

$submit = new Element('send');
$submit->setAttributes(array(
 'type' => 'submit',
 'label' => 'Send',
));
$submit->setAttribute('value', 'Send');

$form = new Form('contact');
$form->add(array($name, $email, $subject, $message, $captcha, $csrf, $send));

$nameInput = new Input('name');

// configure input... and all others
$inputFilter = new InputFilter();

// attach all inputs
$form->setInputFilter($inputFilter);
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 14

Slide 15

## Zend\Form: Creation via Factory

```
use Zend\Form\Factory;
use Zend\Form\Element;
$factory = new Factory();
$form = $factory->createForm(array(
 'hydrator' => 'Zend\Stdlib\Hydrator\ArraySerializable',
 'elements' => array(
 array(
 'name' => 'name',
 'type' => 'text',
 'options' => array('label' => 'Your name'),
),
 array(
 'name' => 'email',
 'type' => 'email',
 'options' => array('label' => 'Your email address'),
),
 array(
 'name' => 'subject',
 'type' => 'text',
 'options' => array('label' => 'Subject'),
),
 array(
 'name' => 'message',
 'type' => 'textarea',
 'options' => array('label' => 'Message'),
),
),
 // Continued on next page
)
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 15

- You can create the entire form, and input filter, using the Factory
- This is particularly nice if you want to store your forms as pure configuration - you can simply pass the configuration to the factory

## Zend\Form: Form Creation

```
array(
 'name' => 'captcha',
 'type' => 'captcha',
 'options' => array(
 'label' => 'Please verify you are human',
 'captcha' => array('class' => 'dumb'),
),
),
array(
 'name' => 'security',
 'type' => 'csrf',
),
array(
 'name' => 'send',
 'type' => 'submit',
 'options' => array('label' => 'Send'),
),
),
/* If we had fieldsets, they would go here; fieldsets contain "elements" and
 * "fieldsets" keys, and potentially a "type" key indicating the specific
 * FieldsetInterface implementation to use.
'fieldsets' => array(
),
*/
// Configuration to pass on to Zend\InputFilter\Factory::createInputFilter()
'input_filter' => array(
 /* ... */
),
));
});
```



- You can create the entire form, and input filter, using the Factory
- This is particularly nice if you want to store your forms as pure configuration - you can simply pass the configuration to the factory

## Validating Forms

### Validating forms requires three steps:

First, the form must have an input filter attached

Second, you must inject the data to validate into the form

Third, validate the form; if invalid, you can retrieve the error messages, if any

```
$form = new Contact\ContactForm();

// If the form doesn't define an input filter by default, inject one
$form->setInputFilter(new Contact\ContactFilter());

// Get the data. In an MVC application, you might try:
$data = getPost(); // for POST data
$data = getQuery(); // for GET (or query string) data

$form->setData($data);

// Validate the form
if ($form->isValid()) {
 $validatedData = $form->getData();
} else {
 $messages = $form->getMessages();
}
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 17

- You can get the raw data if you want, by accessing the composed input filter

```
$filter = $form->getInputFilter();
$rawValues = $filter->getRawValues();
$nameRawValue = $filter->getRawValue('name');
```

# Rendering Forms

## Form Rendering

- Forms are typically rendered in a view script
  - Before rendering, execute `prepare()` to ensure any injections take place
  - Start the form by executing `$this->form()->openTag($form)`
  - End the form by executing `$this->form()->closeTag()`
- Form rendering makes use of a series of `Zend\Form\View\Helper` classes that include: `Form`, `FormCaptcha`, and `FormElement`



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 18

- The `openTag()` call will generate a `` tag with attributes based on your form. However, most editors will indicate an error in your markup if you simply have "</form>" in your markup; thus the `closeTag()` method
- There are form view helpers for every single element type supported
- Other Helper classes include:  
`FormElementErrors`, `FormInput`, `FormLabel`, `FormMultiCheckbox`, `FormRadio`, `FormSelect`, and `FormTextarea`

# Rendering Form Elements

## Form Element Rendering

- Form elements are rendered one at a time within the view script
  - Obtain the element from the form using `$form->get('element')`
- Specific element helpers include: `FormCaptcha`, `FormInput`, `FormMultiCheckbox`, `FormRadio`, `FormSelect`, and `FormTextarea`
- The generic form element view helper is: `FormElement`
- `FormLabel` displays the element's label
- `FormElementErrors` displays any validation error messages
- `formRow()` helper can generate the label, element, and error messages in one step



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 19

- The Zend Framework 2 manual includes a complete list of element helpers  
<http://packages.zendframework.com/docs/latest/manual/en/index.html>

Slide 20

## M9Ex1: Forms Lab

- See Participant Exercises Instruction booklet for details...



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 20

Slide 21

## M9Ex2: Filters Lab

- See Participant Exercises Instruction booklet for details...



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 21

Slide 1



## “ Module Ten

Copyright ©2006-2013, Zend Technologies Inc.

Slide 2

## Module Ten Topics

- In this module, we will cover:
  - `zend\Db`
  - `zend\Db\Adapter`
  - `zend\Db\Sql`
  - `zend\Db\TableGateway`



The PHP Company

Copyright © 2006-2013, Zend Technologies, Inc. | 2

Slide 3

## Zend\Db\Adapter

### What is the Zend\Db\Adapter?

The `Adapter` object is the most important sub-component of `Zend\Db`; it provides a database abstraction layer for the database PHP supports.

- The "Driver" part of `Zend\Db\Adapter` creates an abstraction layer for PHP extensions
- The "Platform" part of `Zend\Db\Adapter` creates a lightweight abstraction layer for vendor-specific platform requirements in its SQL/RDBMS implementation



The PHP Company

Copyright © 2006-2013, Zend Technologies, Inc. | 3

Slide 4

## Creating an Adapter: Quick Start

**How can you easily create an adapter ?**

By instantiating the `Zend\Db\Adapter` class

**Common use case?**

Pass an array of information to the Adapter

```
$adapter = new Zend\Db\Adapter\Adapter($driverArray);
```

- When using this approach, the `Adapter` will attempt to create any dependencies that were not explicitly provided
- A `Driver` object is created from the `$driver` array provided in the constructor
- A `Platform` object is created based on the Driver object type instantiated
- A default `ResultSet` object is created and used
- All three objects can be configured



The PHP Company

Copyright © 2006-2013, Zend Technologies, Inc.

| 4

- Effectively, if the PHP manual uses a particular naming, this naming will be supported by our Driver

Slide 5

## Zend\Db\Adapter: Sample Connections

```
// A MySQL connection using ext/mysql
$adapter = new Zend\Db\Adapter\Adapter(array(
 'driver' => 'Mysqli',
 'database' => 'zend_db_example',
 'username' => 'developer',
 'password' => 'developer-password'
));

// A Sqlite connection via PDO
$adapter = new Zend\Db\Adapter\Adapter(array(
 'driver' => 'Pdo_Sqlite',
 'database' => 'path/to/sqlite.db'
));

// Alternative/generic Sqlite connection via PDO; assumes DB in "data" folder
$adapter = new Zend\Db\Adapter\Adapter(array(
 'driver' => 'pdo',
 'dsn' => 'sqlite:'.__DIR__. '/data/guestbook.db'
));
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 5

## Zend\Db\Adapter: Sample Connections

```
<?php
// Main application configuration array
return array(
 'service_manager' => array(
 'factories' => array(
 'Zend\Db\Adapter\Adapter' =>
 'Zend\Db\Adapter\AdapterServiceFactory',
),
),
 // The AdapterServiceFactory will search for and internally use the
 // element below
 'db' => array(
 'driver' => 'pdo',
 'dsn' =>
 'mysql:dbname=zend_db_example;host=localhost',
 'username' => 'developer',
 'password' => 'developer-password',
 'driver_options' => array(PDO::MYSQL_ATTR_INIT_COMMAND
 => "SET NAMES 'UTF8'"),
),
);
}
```



Slide 7

## Creating an Adapter: Dependency Injection

**What is a more explicit way to create an adapter ?**

By injecting all the dependecies up front

With `Zend\Db\Adapter\Adapter`, all required dependencies are injected through the constructor, which has the following signature in 'pseudo' code

**Common use case?**

```
use Zend\Db\Adapter\Platform\PlatformInterface;
use Zend\Db\ResultSet\ResultSet;

class Zend\Db\Adapter\Adapter {
 public function __construct($driver, PlatformInterface
 $platform = null, ResultSet $queryResultSetPrototype = null)
}
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc.

| 7

Slide 8

## Creating an Adapter: By Injecting Dependencies

**These are parameters you can inject:**

`$driver`  
an array of connection parameters (Quick start) or an instance of  
`Zend\Db\Adapter\Driver\DriverInterface`

`$platform`(optional)  
an instance of `Zend\Db\Platform\PlatformInterface`; the default will be created based  
on the driver implementation

`$queryResultSetPrototype`(optional)  
an instance of `Zend\Db\ResultSet\ResultSet`



The PHP Company

Copyright © 2006-2013, Zend Technologies, Inc.

| 8

Slide 9

## Query Preparation

`query()` is a convenience method provided for quick "one-off" processing

- The first parameter is an SQL statement
- The second parameter can be an array of values which substitute for the parameters....
  - This generally means supplying a SQL statement with its values substituted with placeholders, and the parameters for those placeholders supplied separately
  - Example of this workflow with `Zend\Db\Adapter\Adapter`

```
$adapter->query('SELECT * FROM `artist` WHERE `id` = ?', array(5));
```
- The second parameter can also be the query "mode"
  - The default mode is `Adapter::QUERY_MODE_PREPARE`



The PHP Company

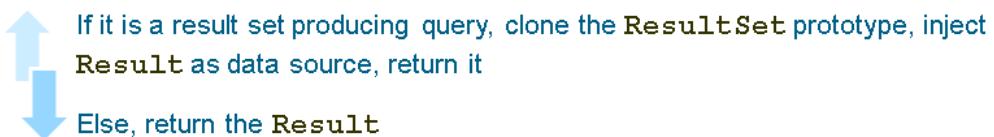
Copyright © 2006-2013, Zend Technologies, Inc. | 9

## Query Preparation

```
$adapter->query('SELECT * FROM `artist` WHERE `id` = ?', array(5));
```

- The corresponding steps are:

- Create a new **Statement** object >>
- Prepare an array into a **ParameterContainer**, if necessary >>
- Inject the **ParameterContainer** into the **Statement** object >>
- Execute the **Statement** object, producing a **Result** object >>
- Check the **Result** object to check if the supplied sql was a "query", or a result set producing statement >>



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 10

## Query Execution

In some cases, you have to execute statements directly, rather than through preparation

- Ex: Executing a DDL statement, which cannot be prepared for most extensions and vendor platforms
- In this case, you will need to provide `Adapter::QUERY_MODE_EXECUTE` as the second parameter

```
$adapter->query('ALTER TABLE ADD INDEX(`foo_index`) ON (`foo_column`)',
Adapter::QUERY_MODE_EXECUTE);
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 11

## Creating Statements

**While `query()` is useful for one-off or quick querying of a database via Adapter, it usually is better to create a statement and interact with it directly**

**To accomplish this, Adapter provides a routine called `createStatement()`**

- This component allows you to create a driver-specific statement, providing greater control over the Prepare-Then-Execute workflow

```
$statement = $adapter->createStatement($sql, $optionalParameters);

$result = $statement->execute();
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 12

## Using the Platform Object

The Platform object provides an API to assist in crafting queries specific to the SQL implementation of a particular vendor

- This object can handle nuances such as how identifiers or values are quoted, or what character acts as the identifier separator
  - Ex: The interface for a platform object

```
interface Zend\Db\Adapter\Platform\PlatformInterface
{
 public function getName();
 public function getQuoteIdentifierSymbol();
 public function quoteIdentifier($identifier);
 public function quoteIdentifierChain($identifierChain);
 public function getQuoteValueSymbol();
 public function quoteValue($value);
 public function quoteValueList($valueList);
 public function getIdentifierSeparator();
 public function quoteIdentifierInFragment($identifier, array
 $additionalSafeWords = array());
}
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 13

## Using the Platform Object

- Ex: Quoting a column name, specific to MySQL

```
$platform = new Zend\Db\Adapter\Platform\Mysql;

// returns `first_name`
$column = $platform->quoteIdentifier('first_name');
```

- Ex: Alternative approach: Rather than directly instantiating a platform object, you could obtain an instance from the adapter

```
$platform = $adapter->getPlatform();

// or

$platform = $adapter->platform;
// magic property access
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 14

## Using the Parameter Container

The `ParameterContainer` object is a container for parameters that need to be passed into a statement object for SQL statements

- This object implements the `ArrayAccess` interface
- `formatParameterName` selects what type of placeholder is used – "?" or ":name"

```
$adapter = new Zend\Db\Adapter\Adapter($driverConfig);

// $qi stands for "quote identifier", and $fp stands for "format parameter"
$qi = function($name) use ($adapter) { return $adapter->platform->
 quoteIdentifier($name); };
$fp = function($name) use ($adapter) { return $adapter->driver->
 formatParameterName($name); };

// Produces the string: UPDATE 'artist' SET 'name'=:name WHERE 'id'=:id
// (assuming MySQL)
$sql = 'UPDATE ' . $qi('artist')
 . ' SET ' . $qi('name') . ' = ' . $fp('name')
 . ' WHERE ' . $qi('id') . ' = ' . $fp('id');
```

- Note: the code for the parameter container is on the next slide



## Using the Parameter Container

```
// @var $statement Zend\Db\Adapter\DriverStatementInterface
$statement = $adapter->createStatement($sql);
$parameters = array(
 'name' => 'Updated Artist',
 'id' => 1
);
$statement->execute($parameters);

// DATA INSERTED, NOW CHECK @var $statement Zend\Db\Adapter\DriverStatementInterface
$statement = $adapter->createStatement('SELECT * FROM '
 . $qi('artist')
 . ' WHERE id = ' . $fp('id'));

// @var $results Zend\Db\ResultSet\ResultSet
$results = $statement->execute(array('id' => 1));
$row = $results->current();
$name = $row['name'];
```



## Zend\Db\Sql

`Zend\Db\Sql` is a SQL abstraction layer for building platform-specific SQL queries via an object-oriented API

- The end result of an `Zend\Db\Sql` object will be to either produce
  - (1) a `Statement & Parameter` container that represents the target query  
or
  - (2) a full string that can be directly executed against the database platform
- `Zend\Db\Sql` objects require a `Zend\Db\Adapter\Adapter` object in order to produce the desired results

```
use Zend\Db\Sql\Sql;
$sql = new Sql($adapter);
$select = $sql->select(); // @return Zend\Db\Sql\Select
$insert = $sql->insert(); // @return Zend\Db\Sql\Insert
$update = $sql->update(); // @return Zend\Db\Sql\Update
$delete = $sql->delete(); // @return Zend\Db\Sql\Delete
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 17

## Zend\Db\Sql

- Ex: Preparing & Executing Statements using a Select object

```
// Prepare Execute Model
use Zend\Db\Sql\Sql;
$sql = new Sql($adapter);
$select = $sql->select();
$select->from('foo');
$select->where(array('id' => 2));
$statement = $sql->prepareStatementForSqlObject($select);
$results = $statement->execute();

// SQL-Generation/Query Model
use Zend\Db\Sql\Sql;
$sql = new Sql($adapter);
$select = $sql->select();
$select->from('foo');
$select->where(array('id' => 2));
$selectString = $sql->getSqlStringForSqlObject($select);
$results = $adapter->query($selectString,
$adapter::QUERY_MODE_EXECUTE);
```



- The Statement keeps track of names, even if the platform doesn't support named parameters – this means you can always use a hash map to specify parameters
- The second example is simply an additional case, showing it works in any type of SQL statement

## Zend\Db\Sql

- Zend\Db\Sql\Sql objects can also be bound to a particular table so that in getting a select, insert, update, or delete object, they are all primarily seeded with the same table when produced

```
use Zend\Db\Sql\Sql;
$select = new Sql($adapter, 'foo');
$select->where(array('id' => 2));
// $select already has the from('foo') applied
```



- Zend\Db\Sql\Sql objects can also be bound to a particular table so that in getting a select, insert, update, or delete object, they are all primarily seeded with the same table when produced.

```
use Zend\Db\Sql\Sql;
$select = new Sql($adapter, 'foo');
$select->where(array('id' => 2));
// $select already has the from('foo') applied
```

## Zend\Db\Sql : Select, Insert, Update, Delete

Each of the objects implement the following two interfaces:

```
public function prepareStatement(Adapter $adapter, StatementInterface
 $statement);
// And
public function getSqlString(PlatformInterface $adapterPlatform = null);
```

- These functions can be called to either
  - (1) produce a prepared statement or
  - (2) generate SQL



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 20

## Zend\Db\Sql\Select

The primary function of the `Zend\Db\Sql\Select` object is to present a unified API for building platform specific SQL SELECT queries

- It can be instantiated and consumed without `Zend\Db\Sql\Sql`

```
use Zend\Db\Sql\Select;
$select = new Select();

// or, to produce a $select bound to a specific table
$select = new Select('foo');
```

- The available methods, which all correspond to SQL equivalents
- All of these methods implement a fluent interface

Methods

|                        |                       |                       |
|------------------------|-----------------------|-----------------------|
| <code>from()</code>    | <code>limit()</code>  | <code>having()</code> |
| <code>columns()</code> | <code>join()</code>   | <code>order()</code>  |
| <code>group()</code>   | <code>offset()</code> | <code>where()</code>  |



- NOTE: 'Select' is a child class of `Zend\Db\Sql\AbstractSql`

# Zend\Db\Sql\Insert

## The Insert class

```
class Insert implements SqlInterface, PreparableSqlInterface
{
 const VALUES_MERGE = 'merge';
 const VALUES_SET = 'set';

 public function __construct($table = null);
 public function into($table);
 public function columns(array $columns);
 public function values(array $values, $flag = self::VALUES_SET);
}
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 22

# Zend\Db\Sql\Update

## The Update class

```
class Update
{
 const VALUES_MERGE = 'merge';
 const VALUES_SET = 'set';

 public $where; // @param Where $where
 public function __construct($table = null);
 public function table($table);
 public function set(array $values, $flag = self::VALUES_SET);
 public function where($predicate, $combination =
 Predicate\PredicateSet::OP_AND);
}
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 23

# Zend\Db\Sql\Delete

## The Delete class

```
class Delete
{
 public $where; // @param Where $where
 public function __construct($table = null);
 public function from($table);
 public function where($predicate, $combination =
 Predicate\PredicateSet::OP_AND);
}
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 24

## Zend\Db\Sql\Where & Predicates

Zend\Db\Sql\Where **is used to build a SQL Where clause**

- When building the `Where` clause, you make use of "predicates"
- Predicates include `Between`, `Equal To`, `In`, `IsNull`, `Like`, ...
- During parameterization, the parameters are replaced with their proper placeholder (a named or positional parameter), and the values are stored inside an `Adapter\ParameterContainer`
- When executed, the values will be interpolated into the fragments to which they belong and properly quoted



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | **25**

## Zend\Db\Sql\Where API

**It is important to realize that, in this API, a distinction is made between which elements are considered identifiers (`TYPE_IDENTIFIER`) and values (`TYPE_VALUE`)**

- There is also a special use case type for literal values (`TYPE_LITERAL`)
- These are all exposed via the `Zend\Db\Sql\ExpressionInterface` interface
- The API code follows ...



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 26

Slide 27

## Zend\Db\Sql\Where & Predicates: Code Example

```
// Example using Zend\Db\Select and Zend\Db\Where
// Note: import statements in notes

// create the objects
$platform = new Mysql();
$select = new Select();
$where = new Where();

// specify the "WHERE" conditions
$where->greaterThanOrEqualTo('qty_oh', 10);
$where->lessThan('cost', 100);
$select->from('products')
 ->where($where);

// extract the SQL
echo $select->getSqlString($platform);

// outputs:
// SELECT `products`.* FROM `products` WHERE `qty_oh` >= '10' AND `cost` < '100'
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 27

## Zend\Db\Sql\Where & Predicates: Code Example

```
// Note: import statements in notes

// setup Zend\Db\Sql/* objects
$platform = new Mysql();
$select = new Select();
$where = new Where();
$like1 = new Like('city', '%moon%');
$like2 = new Like('city', '%lake%');
$where->orPredicate($like1);
$where->orPredicate($like2);

// consolidate SQL
$select->from('names')
 ->columns(array('fn' => 'first_name', 'ln' => 'last_name', 'city'))
 ->where($where);
echo $select->getSqlString($platform);

/* outputs:
SELECT `names`.`first_name` AS `fn`, `names`.`last_name` AS `ln`, `names`.`city`
AS `city` FROM `names` WHERE `city` LIKE '%moon%' OR `city` LIKE '%lake%'
*/
```



- Note the following import statements:

```
// "use" the appropriate classes
use Zend\Db\Adapter\Platform\Mysql;
use Zend\Db\Sql>Select;
use Zend\Db\Sql\Where;
```

- Code continued on next slide/page...

## Zend\Db\Sql\Where API: Code Example

```
// Note: import statements in notes

$platform = new Mysql();
$select = new Select();
$where = new Where();
$where->like('city', '%salt%');
$where->nest->like('city', '%lake%')->or->like('city', '%wood%')->unnest;

$select->from('names')
 ->columns(array('fn' => 'first_name', 'ln' => 'last_name', 'city'))
 ->where($where);
echo $select->getSqlString($platform);

// Outputs:
// SELECT `names`.`first_name` AS `fn`, `names`.`last_name` AS `ln`,
// `names`.`city` AS `city` FROM `names` WHERE `city` LIKE '%salt%' AND
// (`city` LIKE '%lake%' OR `city` LIKE '%wood%')
```



- Note the following import statements:

```
// use Zend\Db\Sql* classes
use Zend\Db\Sql\Select;
use Zend\Db\Sql\Where;
```

## Zend\Db\TableGateway

The Table Gateway object provides an object that represents a table in a database

- The methods of this object mirror the most common table operations
- The interface for such an object looks like:

```
interface Zend\Db\TableGateway\TableGatewayInterface
{
 public function getTable();
 public function select($where = null);
 public function insert($set);
 public function update($set, $where = null);
 public function delete($where);
}
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 30

## Creating a Table: By Injecting Dependencies

- The most explicit way to create an adapter is to inject all the dependencies up front
- Zend\Db\TableGateway\TableGateway can use constructor injection, where all required dependencies are injected through the constructor. The example below shows a tablegateway class where the adapter is obtained via constructor injection.

```
class SomeTable extends AbstractTableGateway
{
 public function __construct(Adapter $adapter, ResultSetInterface
 $resultSetPrototype=null)
 {
 /* ... */
 }
}

// Alternately, show the _actual_ `TableGateway` constructor:
class TableGateway extends TableGateway
{
 public function __construct($table, Adapter $adapter, $features = null
 ResultSetInterface $resultSetPrototype = null, Sql $sql = null)
 {
 /* ... */
 }
}
```



## Creating a Table: By Injecting Dependencies

**These are parameters you can inject:**

**\$table**

a string representing the name of the table as represented in the database

**\$adapter**

an instance of `Zend\Db\Adapter\Adapter` or `Zend\Db\Adapter\Driver`\*

**\$resultSetPrototype**

an object of a class which implements `Zend\Db\ResultSet\ResultSetInterface`  
(such as `Zend\Db\ResultSet\ResultSet`)

**\$sql**

an instance of `Zend\Db\Sql\Sql`

**\$table**

the name of the table as it appears in the database

**\$features**

an implementation of a class `Zend\Db\TableGateway\Feature`\*; lets you define magic  
calls, **GETs & SETs**, as well as an `initialize()` method



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 32

## Zend\Db\TableGateway: Code Examples

### Table Gateway examples:

- Selecting Rows

```
$artistTable = new Zend\Db\TableGateway\TableGateway('artist', $adapter);
$rowset = $artistTable->select(array('id' => 2));
$artistRow = $rowset->current();
```

- Inserting Rows

```
$artistTable = new Zend\Db\TableGateway\TableGateway('artist', $adapter);
$data = array('id' => 3,
 'email' => 'bruce@springsteen.com',
 'band' => 'The E Street Band');
$success = $artistTable->insert($data);
```



- Note that select() always returns a ResultSet, even when a single row is returned

## Zend\Db\TableGateway: Code Examples

### Table Gateway examples:

- Updating Rows

```
$artistTable = new Zend\Db\TableGateway\TableGateway('artist', $adapter);
$result = $artistTable->update(array('name' => 'New Artist'),
 array('id' => 2));
$name = $row['name'];
```

- Deleting Rows

```
$artistTable = new Zend\Db\TableGateway\TableGateway('artist', $adapter);
$result = $artistTable->delete(array('id' => 2));
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 34

## Zend\Db\TableGateway: Code Examples

### Table Gateway examples:

- Using a Where object with Select

```
$artistTable = new TableGateway('artist', $adapter);
$where = $artistTable->getSql()->select()->where();
$where->like('name', 'Bar%');
$rowset = $artistTable->select($where);
$row = $rowset->current();
```

- Using a Closure with Select

```
$artistTable = new TableGateway('artist', $adapter);
$rowset = $artistTable->select(function (Select $select) {
 $select->where()->like(/* ... */);
});
$row = $rowset->current();
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 35

Slide 36

## M10Ex1: Database Connectivity

- See Participant Exercises Instruction booklet for details...



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 36

Slide 1



The PHP Company

## “ Module Eleven ...

Copyright © 2006-2013, Zend Technologies Inc.

Slide 2

## Module Eleven Topics

- In this model, we will cover:
  - `zend\Session`
  - (Optional) `zend\Log`
  - (Optional) `zend\Mail`



The PHP Company

Copyright © 2006-2013, Zend Technologies, Inc. | 2

Slide 3



## Zend\Session\\*

Copyright ©2006-2012, Zend Technologies Inc.

Slide 4

## Zend\Session\SessionManager

- Within PHP applications, a Session represents a logical, 1:1 connection between Server-side, persistent state data and a particular User Agent Client (Ex: a web browser)
  - Thus, a standard PHP session id, stored either in a client's cookie or embedded in URLs, maintains the association between a client and session state data
- Zend\Session\SessionManager helps to manage and preserve session data across multiple page requests by the same client
  - Unlike cookie data, session data is *not* stored on the client but on the server
- Zend\Session\SessionManager is not intended to work directly on the contents of session namespace containers - instead, we use Zend\Session\Container



The PHP Company

Copyright © 2006-2012, Zend Technologies, Inc.

| 4

Slide 5

## Zend\Session and Zend\Session\Container

- Session data, normally accessed via `$_SESSION[]`, is managed by `Zend\Session\SessionManager` and manipulated by `Zend\Session\Container` accessor objects
- If no namespace is specified when instantiating `Zend\Session\Container`, all data will be transparently stored in a namespace called "Default"
- `Zend\Session\Container` is used to segregate all session data
  - Default exists to store all session data as one namespace
- `Zend\Session\SessionManager` utilizes `ext/session` and its special `$_SESSION` superglobal as the storage mechanism for session state data
  - NOTE: While `$_SESSION` is still available in PHP's global namespace, developers should refrain from directly accessing it, so that `Zend\Session\SessionManager` and `Zend\Session\Container` can operate most effectively and securely



The PHP Company

Copyright © 2006-2012, Zend Technologies Inc.

| 5

## Zend\Session\Config\StandardConfig

- Zend\Session\Config\SessionConfig is a class that provides access to php.ini settings for the session extension
- Zend\Session\Config\standardConfig provides a way to configure the session *without* altering php.ini settings
- Get methods allow you to retrieve session settings
- Set methods allow you to set session settings
- session.\* php.ini settings include save\_path, use\_cookies, cache\_expire, cookie\_lifetime, cookie\_path, cookie\_secure
- There is also a method, `setRememberMeSeconds()`, which is a shortcut for session expiration



The PHP Company

Copyright © 2006-2012, Zend Technologies, Inc. | 6

- Other session.\* php.ini settings include:  
cookie\_domain, cookie\_httponly, entropy\_file, name, gc\_divisor, gc\_maxlifetime, gc\_probability, hash\_bits\_per\_character

Slide 7

## Sessions

```
use Zend\Session\Container;
use Zend\Session\Config\SessionConfig;
use Zend\Session\SessionManager;

$container = new Container('my-app');
$container->foo = 'bar';
// next request:
$container = new Container('my-app');
if (isset($container->foo)) {
 echo $container->foo;
}
// Configuration
$config = array(/* array of config options */);
$sessionConfig = new SessionConfig($config);
$manager = new SessionManager($sessionConfig);
Container::setDefaultManager($manager);
```



The PHP Company

Copyright © 2006-2012, Zend Technologies Inc.

| 7

## Zend\Session Best Practices

### Identity Persistence

- Important to support maintaining the authenticated identity without having to present the credentials with each request (due to stateless nature of HTTP)
- The `Zend\Session` variant class `Zend\Authentication\Storage\Session` can be configured and assigned to an authentication service, to provide for persistent identity storage

### Lock

- Sessions can be effectively locked using the `isImmutable()` and `markImmutable()` methods of the session storage object assigned to the session manager
- Locks are transient, and do not persist across requests



The PHP Company

Copyright © 2006-2012, Zend Technologies, Inc.

| 8

- HTTP is a stateless protocol, however, and techniques such as cookies and sessions have been developed in order to facilitate maintaining state across multiple requests in server-side web applications
- `Zend\Session` is used by default within `Zend\Authentication` to provide persistent storage of the identity from a successful authentication attempt using the PHP session
- By default, `Zend\Authentication` uses a storage class based on `Zend\Session`. The storage class may be changed by providing a different storage object to `Zend\Authentication::setStorage()`

## Zend\Session Best Practices

### Expiration

- Limits can be placed on the longevity of both containers and individual keys in containers – example:
  - for passing temporary information between requests
  - for reducing exposure to certain security risks by removing access to potentially sensitive information 'x' times after authentication occurs
- Expiration can be based on elapsed seconds, or the concept of "hops", where a hop occurs for each successive request that activates the container, such as:

```
$space = new Zend\Session\Container('myspace');
```

- Can also use containers to separate session access by controllers, to protect variables from contamination
- Ex: An **Authentication** controller might keep its session state data separate from all other controllers



The PHP Company

Copyright © 2006-2012, Zend Technologies, Inc.

| 9

Slide 10

## M11Ex1: Sessions Lab

- See Participant Exercises Instruction booklet for details...



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 10

Slide 11



## (Optional) Zend\Log

Copyright ©2006-2013, Zend Technologies Inc.

## Zend\Log

`Zend\Log\Logger` is a general purpose logging class that supports multiple log backends, formatting messages sent to the log and filtering messages from being logged via 4 types of objects:

- **Log** (instance of `Zend\Log\Logger`)
  - Object that an application uses most; Unlimited number; Does not interact; Must contain at least one Writer
- **Writer** (inherits from `Zend\Log\Writer\AbstractWriter`)
  - Responsible for writing the log message; Can contain one or more Filters
- **Filter** (implements `Zend\Log\Filter\FilterInterface`)
  - Determines whether the writer is interested in a given message
- **Formatter** (implements `Zend\Log\Formatter\FormatterInterface`)
  - Formats the log data before it is written by a Writer
  - Some writers (notably `Zend\Log\Writer\Db`) do not support Formatters



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 12

- Logs data to the console, flat files, or a database
- Its simple, procedural API reduces the hassle of logging to one line of code and is perfect for cron jobs and error logs
  - provides a simple object-oriented interface inspired by log4j
  - supports extensible output channels
  - supports extensible output formats
- All logs must have at least one Writer
  - To create a log, all you have to do is instantiate a Writer and then pass it to a Log instance

Slide 13

## Create a Log

- **Create a Log:**

```
use Zend\Log\Logger as Logger;
use Zend\Log\Writer\Stream as Stream;

$logger = new Logger();
$fp = fopen(__DIR__ . '/example.log', 'w+');
$writer = new Stream($fp);
$logger->addWriter($writer);
```

- **Log a Message :**

```
$logger->log(Logger::INFO, 'Informational message');
```

- Zend\Log can be used in a simplified way for a single log, can be configured for multiple logs, and can also be used to log the internal operations of many ZF2 classes



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 13

- Note that the use of log() requires a priority (listed on next slide)

## Log Levels

- The Zend\Log\Logger class defines the following priorities:

```
EMERG = 0; // Emergency: system is unusable
ALERT = 1; // Alert: action must be taken immediately
CRIT = 2; // Critical: critical conditions
ERR = 3; // Error: error conditions
WARN = 4; // Warning: warning conditions
NOTICE = 5; // Notice: normal but significant condition
INFO = 6; // Informational: informational messages
DEBUG = 7; // Debug: debug messages
```



- Priority numbers descend in order of importance. EMERG (0) is the most important priority; DEBUG (7) is the least important priority of the built-in priorities
- There are corresponding methods for each one:

|         |          |
|---------|----------|
| emerg() | warn()   |
| alert() | notice() |
| crit()  | info()   |
| err()   | debug()  |
- You may define priorities of lower importance than DEBUG using \$log->addPriority(). When selecting the priority for your log message, be aware of this priority hierarchy and choose appropriately
- These priorities are always available, and not arbitrary. They come from the BSD syslog protocol, which is described in RFC-3164
- The names and corresponding priority numbers are also compatible with another PHP logging system, PEAR Log, which perhaps promotes interoperability between it and Zend\_Log

## Using Zend\Log with Zend\Debug\Debug

### Debugging with Zend\Log\Logger:

- **Static method** `Zend\Debug\Debug::dump()` prints or returns information about an expression
- `Zend\Debug\Debug::dump()` is best for *ad hoc* debugging during development, requiring no initialization, special tools, or debug environment
  - Add code to dump a variable and then quickly remove the code
- Consider `Zend\Log` when writing more permanent debugging code
  - For example, use the `DEBUG` log level and the `Streamlog` writer to output the string returned by `Zend\Debug\Debug::dump()`

```
// log the contents of variable $data
$logger->debug(
 Zend\Debug\Debug::dump($data, 'Contents of $data', false));
```

- The method `registerExceptionHandler($logger)` intercepts uncaught exceptions, and then passes the exception information to the logger



## Zend\Log Writers

- Writing to Files:

```
$writer = new Zend\Log\Writer\Stream(__DIR__ . '/example.log');
$logger = new Zend\Log\Logger();
$logger->addWriter($writer);
$logger->info('Informational message');
```

- Writing to Databases:

```
$dbAdapter = new Zend\Db\Adapter\Adapter(array(
 'driver' => 'Mysqli',
 'database' => 'zend_db_example',
 'username' => 'developer',
 'password' => 'developer-password'
));
$colMap = array(array('code' => 'priority',
 'level' => 'priorityName', 'msg' => 'message'));
$writer = new Zend\Log\Writer\Db($dbAdapter, 'log_table_name', $colMap);
$logger = new Zend\Log\Logger();
$logger->addWriter($writer);
$logger->info('Informational Message');
```



- There are more writers than shown in the code example:

ChromePhp and FirePhp

FingersCrossed (only log if a certain threshold is met; if so, then log everything that has been logged to that point)

Mail

MongoDB

Syslog

ZendMonitor

## Zend\Log Formatters

- Simple Formatting (`Zend\Log\Formatter\Simple` is the default config):

```
<?php
$format = '%timestamp% %priorityName%(%priority%):%message%'.PHP_EOL;
$formatter = new Zend\Log\Formatter\Simple($format);
```

- Formatting to XML:

```
<?php
$writer = new Zend\Log\Writer\Stream('php://output');
$formatter = new Zend\Log\Formatter\Xml();
$writer->setFormatter($formatter);
$logger = new Zend\Log\Logger();
$logger->addWriter($writer);
$logger->('Informational message');
```



- A Formatter is an object that is responsible for taking an event array describing a log event and outputting a string with a formatted log line.
- `Zend\Log\Formatter\Simple` is the default formatter. It is configured automatically when you specify no formatter.
- `Zend\Log\Formatter\Xml` formats log data into XML strings. By default, it automatically logs all items in the event data array; The code above outputs the following XML

```
<logEntry>
<timestamp>2007-04-06T07:24:37-07:00</timestamp>
<message>informational message</message>
<priority>6</priority>
<pri<priorityName>INFO</priorityName>
</logEntry>
```

## Zend\Log Filters

- Filtering for a Writer Instance :

```
$logger = new Zend\Log\Logger();
$writer1 = new Zend\Log\Writer\Stream($logFile);
$writer2 = new Zend\Log\Writer\Syslog();
$filter = new Zend\Log\Filter\Priority(Zend\Log\Logger::ALERT);
$writer2->addFilter($filter);
$logger->addWriter($writer1);
$logger->addWriter($writer2);

// goes into both files
$logger->emerg('Emergency Message');

// does not go into syslog
$logger->info('Informational Message');
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 18

Slide 19

## M11Ex1: Making Entries to a Log

- See Participant Exercises Instruction booklet for details...



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 18

Slide 20



## (Optional) Zend\Mail

Copyright ©2006-2013, Zend Technologies Inc.

Slide 21

## Zend\Mail\Message

The `Message` class encapsulates a single email message, according to RFCs 822 and 2822 standards

- The class is a value object, used for setting mail headers and content
  - It cannot *send* itself – you would need to use a Transport adapter
  - It cannot *store* itself – you would need to use a Storage adapter
  - It has a series of "add", "set", and "get" methods that allow you to read or compose a mail message
- To create a message, instantiate the class

```
use Zend\Mail\Message; $message = new Message();
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 21

- Note that the `Message` class assumes ASCII encoding for the email
- You can choose other forms of encoding by specifying it – for example, UTF-8  
`$message->setEncoding("UTF-8");`

Slide 22

## Zend\Mail\Message Methods

There are a number of methods available: here are some common ones:

| Methods                     | Meaning                    |
|-----------------------------|----------------------------|
| <code>isValid()</code> ;    | Is the message valid?      |
| <code>setSubject()</code> ; | Set the message subject    |
| <code>setHeaders()</code> ; | Compose headers...         |
| <code>setBody()</code>      | Set the message body       |
| <code>addCc()</code> ;      | Add a "Cc" address         |
| <code>addTo()</code>        | Add one or more recipients |
| <code>addFrom()</code>      | Add a "From" address       |

- There are also "get" methods that correspond to the "add" and "set" methods, such as `getTo()`, `getHeaders()`, ...



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 22

- The full list of methods and related details can be viewed in Sections 20.1.4 in the ZF2 Reference Manual:

<http://packages.zendframework.com/docs/latest/manual/en/zend.mail.html>

## Zend\Mail Message: Zend\Mime

To create multi-part messages, or ones with HTML content, use `Zend\Mime` to create a MIME message object, and then assign it as the body of the Mail message

```
use Zend\Mail\Message,
 Zend\Mime\Message as MimeMessage,
 Zend\Mime\Part as MimePart;

$text = new MimePart($textContent);
$text->type = "text/plain";

$html = new MimePart($htmlMarkup);
$html->type = "text/html";

$image = new MimePart(fopen($pathToImage));
$image->type = "image/jpeg";

$body = new MimeMessage();
$body->setParts(array($text, $html, $image));

$message = new Message();
$message->setBody($body);
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 23

- The Message class will automatically set a "MIME-Version" header, as well as an appropriate "Content-Type" header

## Adding Headers and Content

- Once you have a `Message` instance, you can add headers and content:

```
$message->addFrom("sender@abc.com", "I. M. Sender")
 ->addTo("foobar@example.com")
 ->setSubject("Sending an email from Zend\Mail!");
$message->setBody("This is the message body.");
```

- To add additional recipients as a "Cc" or "Bcc":

```
$message->addCc ("C.C.Recipient@zend.com")
 ->addBcc ("B.C.C.Recipient@zend.com");
```

- To add an alternate reply address:

```
$message->addReplyTo("info@abc.com", "Customer Service");
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 24

- Cc and Bcc are not Latin-based... Cc refers to "carbon copy", a throwback to when typewritten materials were copied using sheets of black "carbon" paper
- Bcc refers to "blind carbon copy", which means this recipient will receive a copy of the email, but their address will not be visible to any other recipients

Slide 25

## Zend\Mail\Transport

`Zend\Mail\Transport` performs the actual delivery of emails

- There are three options:
  - `Sendmail` functionality (uses system resources to deliver)
  - `SMTP` protocol (uses remote server to deliver)
  - `File` transport (creates a mail file for each message sent)
- The `Zend\Mail\Transport` interface defines exactly one method, `send()`
- This method accepts a `Zend\Mail\Message` instance, which it then introspects and serializes in order to send
- Configuration options depend upon the transport used



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 25

- To learn more about the configuration options, see Sections 20.2.3 in the ZF2 Reference Manual:

SMTP Transport:

<http://packages.zendframework.com/docs/latest/manual/en/zend.mail.smtp-options.html>

File Transport:

<http://packages.zendframework.com/docs/latest/manual/en/zend.mail.file-options.html>

## Zend\Mail\Transport

```
// Using PHP's native mail() functionality
use Zend\Mail\Message,
 Zend\Mail\Transport\Sendmail as SendmailTransport;
$message = new Message();
$message->addTo('matthew@zend.com')
 ->addFrom('ralph.schindler@zend.com')
 ->setSubject('Greetings and Salutations!')
 ->setBody("Sorry, I'm going to be late today!");
$transport = new SendmailTransport();
$transport->send($message);
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 26

## Zend\Mail\Transport

```
// Using SMTP Protocol
use Zend\Mail\Message,
 Zend\Mail\Transport\Smtp as SmtpTransport,
 Zend\Mail\Transport\SmtpOptions;
$message = new Message();
$message->addTo('matthew@zend.com')
 ->addFrom('ralph.schindler@zend.com')
 ->setSubject('Greetings and Salutations!')
 ->setBody("Sorry, I'm going to be late today!");
// Setup SMTP transport using LOGIN authentication
$transport = new SmtpTransport();
$options = new SmtpOptions(array(
 'name' => 'localhost.localdomain',
 'host' => '127.0.0.1',
 'connection_class' => 'login',
 'connection_config' => array(
 'username' => 'user',
 'password' => 'pass',
),
));
$transport->setOptions($options);
$transport->send($message);
```



Slide 28

## Zend\Mail\Transport

```
// Using ZF2 File Transport
use Zend\Mail\Message,
 Zend\Mail\Transport\File as FileTransport,
 Zend\Mail\Transport\FileOptions;

$message = new Message();
$message->addTo('matthew@zend.com')
 ->addFrom('ralph.schindler@zend.com')
 ->setSubject('Greetings and Salutations!')
 ->setBody("Sorry, I'm going to be late today!");

// Setup SMTP transport using LOGIN authentication
$transport = new FileTransport();
$options = new FileOptions(array(
 'path' => 'data/mail/',
 'callback' => function (FileTransport $transport) {
 return 'Message_' . microtime(true) . '_' . mt_rand() . '.txt';
 },
));
$transport->setOptions($options);
$transport->send($message);
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 28

## Zend\Mail\Transport\Options

There are four basic options for the Zend\Mail\Transport\Smtp mail transport:

- Basic SMTP Transport Usage
- SMTP Transport Usage with PLAIN AUTH
- SMTP Transport Usage with LOGIN AUTH
- SMTP Transport Usage with CRAM-MD5 AUTH

- The configuration options are:

|                   |                                |
|-------------------|--------------------------------|
| <code>name</code> | <code>connection_class</code>  |
| <code>host</code> | <code>connection_config</code> |
| <code>port</code> |                                |



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 29

- The configuration options are:
  - `name`: Name of the SMTP host; defaults to "localhost".
  - `host` : Remote hostname or IP address; defaults to "127.0.0.1".
  - `port` : Port on which the remote host is listening; defaults to "25".
  - `connection_class`: Fully-qualified classname or short name resolvable via Zend\Mail\Protocol\SmtpLoader;  
Typically, this will be "smtp", "plain", "login", or "crammd5"; default is "smtp"  
Typically, the connection class should extend the Zend\Mail\Protocol\AbstractProtocol class, specifically the SMTP variant
  - `connection_config`: Optional associative array of parameters to pass to the `connection_class` in order to configure it; by default, this is empty; for connection classes other than the default, you will typically need to define the "username" and "password" options

Slide 30

## Zend\Mail\Transport\SmtpOptions Methods

There are a number of methods available: here are some common ones:

| Methods                                                    | Meaning                                                     |
|------------------------------------------------------------|-------------------------------------------------------------|
| <code>getName();</code>                                    | Returns the sting name of the local client hostname         |
| <code>setName(string \$name);</code>                       | Set the sting name of the local client hostname             |
| <code>getConnectionClass();</code>                         | Returns a sting indicating the connection class name to use |
| <code>setConnectionClass(string \$connectionClass);</code> | Set the connection class to use                             |
| <code>getConnectionConfig();</code>                        | Get configuration for the connection class                  |
| <code>setConnectionConfig(array \$config);</code>          | Set configuration for the connection class                  |
| <code>__construct(null array Traversable \$config);</code> | Instantiate the class and (optional) configure with values  |



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 30

- The full list of methods and related details can be viewed in Sections 20.3.4 in the ZF2 Reference Manual:

<http://packages.zendframework.com/docs/latest/manual/en/zend.mail.smtp-options>

## Zend\Mail\Transport\SmtpOptions: Code Example

```
// Basic SMTP Transport Usage
use Zend\Mail\Transport\Smtp as SmtpTransport,
Zend\Mail\Transport\SmtpOptions;

// Setup SMTP transport
$transport = new SmtpTransport();
$options = new SmtpOptions(array(
 'name' => 'localhost.localdomain',
 'host' => '127.0.0.1',
 'port' => 25,
));
$transport->setOptions($options);
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 31

Slide 32

## Zend\Mail\Transport\SmtpOptions: Code Example

```
// SMTP Transport usage with PLAIN AUTH
use Zend\Mail\Transport\Smtp as SmtpTransport,
Zend\Mail\Transport\SmtpOptions;

// Setup SMTP transport using LOGIN authentication
$transport = new SmtpTransport();
$options = new SmtpOptions(array(
 'name' => 'localhost.localdomain',
 'host' => '127.0.0.1',
 'connection_class' => 'plain',
 'connection_config' => array(
 'username' => 'user',
 'password' => 'pass',
),
));
$transport->setOptions($options);
```



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 32

## Zend\Mail\Transport\SmtpOptions: Code Example

```
// SMTP Transport usage with LOGIN AUTH
use Zend\Mail\Transport\Smtp as SmtpTransport,
Zend\Mail\Transport\SmtpOptions;

// Setup SMTP transport using LOGIN authentication
$transport = new SmtpTransport();
$options = new SmtpOptions(array(
 'name' => 'localhost.localdomain',
 'host' => '127.0.0.1',
 'connection_class' => 'login',
 'connection_config' => array(
 'username' => 'user',
 'password' => 'pass',
),
));
$transport->setOptions($options);
```



## Zend\Mail\Transport\SmtpOptions: Code Example

```
// SMTP Transport usage with CRAM-MD5 AUTH
use Zend\Mail\Transport\Smtp as SmtpTransport,
Zend\Mail\Transport\SmtpOptions;

// Setup SMTP transport using LOGIN authentication
$transport = new SmtpTransport();
$options = new SmtpOptions(array(
 'name' => 'localhost.localdomain',
 'host' => '127.0.0.1',
 'connection_class' => 'crammd5',
 'connection_config' => array(
 'username' => 'user',
 'password' => 'pass',
),
));
$transport->setOptions($options);
```



Slide 35

## M11Ex3: Mail Lab

- See Participant Exercises Instruction booklet for details...



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 35

Slide 36

## Zend Framework 2: Advanced Topics

**Proposed topics to be covered in our advanced Zend Framework 2 class:**

- **Advanced features:** Event Manager, Dependency Injection, MVC, ...
- **Performance & Security:** Caching, Encryption, Navigation, Pagination, ...
- **Web Services & External Authentication:** LDAP, REST, ...
- **Code Generation & Annotations:** Reflection, Scanning, Annotations, ...
- **JavaScript Integration & Form Enhancements:** Captchas, Ajax, JSON, ...
- **Internationalization:** Locale, Translate, ...



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 36

## Acceptance of Conditions for Use of Code

ZEND grants you a nonexclusive license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

THE ZEND COURSES' MATERIALS ARE PROVIDED "AS IS" AND SUBJECT TO ANY STATUTORY WARRANTIES WHICH CANNOT BE EXCLUDED, ZEND, ITS OFFICERS, DIRECTORS, EMPLOYEES, PROGRAM DEVELOPERS AND TRAINING PARTNERS MAKE NO WARRANTIES OR CONDITIONS EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT, REGARDING THE COURSES, ZEND COURSES' MATERIALS OR PROGRAMS PROVIDED, IF ANY.

UNDER NO CIRCUMSTANCES IS ZEND, ITS OFFICERS, DIRECTORS, EMPLOYEES, PROGRAM DEVELOPERS OR TRAINING PARTNERS LIABLE FOR ANY OF THE FOLLOWING, EVEN IF INFORMED OF THEIR POSSIBILITY:

LOSS OF, OR DAMAGE TO, DATA;  
DIRECT, SPECIAL, INCIDENTAL, OR INDIRECT DAMAGES, OR FOR ANY ECONOMIC CONSEQUENTIAL DAMAGES;  
OR  
LOST PROFITS, BUSINESS, REVENUE, GOODWILL, OR ANTICIPATED SAVINGS.  
ACCURACY OR COMPLETENESS OF THE ZEND COURSES' MATERIALS.

SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF DIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, SO SOME OR ALL OF THE ABOVE LIMITATIONS OR EXCLUSIONS MAY NOT APPLY TO YOU.



The PHP Company

Copyright © 2006-2013, Zend Technologies Inc. | 37

Slide 38



Questions?

Copyright © 2006-2013, Zend Technologies Inc.

Slide 39



# Thank You for Attending!

Copyright © 2006-2013, Zend Technologies Inc.