

Saturday, May 26. 2007

## May's phplarchitect Magazine - worthy weekend reading

I usually don't extol the virtues of magazines in blog posts. However, I've been sitting down with a printout of May's phplarchitect Magazine and figured it was well worth some attention. A lot of PHP developers whether newcomers or even old hands just looking to expand their knowledge would love this issue.


The two that stick in my mind are Jim Delahunt's "Unicode and PHP: A Gentle Introduction" and Jeff Moore's "Test Pattern: Model View Controller". Both are something those seeking illumination on these topics should certainly read. Jeff's article presents an overview of the MVC Pattern and injects a lot of clarity into a confusing subject for many PHP developers. This is one of the few times reading about MVC in PHP that there's more than a vague reference to a really important point - that Views may (even should) be empowered to read data from the Model. As Jeff notes the continuing popular trend of forcing an application to shuffle between Controllers just to inject Model data in part of a View is completely unnecessary. In my own humble opinion, over-dependency on Controllers to inject Model data into Views is wasteful of resources by pushing folk into adding more and more Controller classes for every teeny View->Model interaction (more work and files to maintain). I'd be hard pressed to believe it doesn't impose a performance cost too.

Jim Delahunt's article is another noteworthy one. I have a pet peeve about developers and popular online websites which store usernames with Unicode characters, render HTML which declares itself UTF-8, and which results in garbage characters like question marks everywhere. For a long time, my own name with a common European character (a-acute) failed to render on most sites claiming to output UTF-8 - assuming of course the site even allowed me to submit my real name... These days there's a marked improvement. This excellent article digs into Unicode in an easy to access way. If you are still not familiar with Unicode, UTF-8, the other Unicode standards, and why they are essential knowledge when building internationalised web apps then this is the article for you.

What else is in the mag this month? There's a cool overview of the Symfony Framework, Iliia Alshanetsky on Dictionary Attacks, and a few other articles I really enjoyed but won't mention here directly. As PHP magazines go, this is an action packed issue worth being enthusiastic about. You can review and buy the PDF version of May's issue for a handful of Canadian Dollars over at [phplarchitect](http://phplarchitect.com).

Posted by Pádraic Brady in Irishisms, PHP General, PHP Security at 20:48

## Complex Views with the Zend Framework - Part 5: The Two-Step View Pattern

It's been a while since I continued this series. Unfortunately real life workloads are unforgiving of the best of intentions 

Part 5 of our series takes a small time-out from approaching a Composite View solution to reusable Views to take a peek at a simpler approach useful for simpler types of web applications. As we've discussed previously Composite Views allow the nesting of reusable View elements, effectively building a View based on a hierarchy of Views. But often there are simpler solutions to simpler problems. One such solution is the Two-Step View pattern, sometimes called Layouts if implemented in a specific way (as we do below!).

Imagine a simple website. You have hundreds of pages of unique content, but the header and footer for each page is identical. Applying the Composite View approach has the same problem as applying a standard PHP include or `Zend_View::render()` call to including these common elements - the calls themselves are scattered across each and every template. This is the hallmark of a Layout - duplicated includes/renders across multiple views:

```
<?php echo $this->render('standard_header.phtml'); ?>
<p>Here is our unique content! But look, all unique templates
now have the same "standard" headers and footers defined by a
render() call. How do we remove these completely and apply them
automatically instead?</p>
<?php echo $this->render('standard_footer.phtml'); ?>
```

The key is to take the duplicated calls and other duplicated markup and stick them in a Layout template which will encapsulate all Views automatically. Then the only thing our templates contain is unique content!

```
<p>Here is our unique content template! But where have the
standard header and footer includes vanished to?</p>
```

All the Layout file needs to do is provide a "hook", a method placement which signifies where the main View output (which is generated by the current dispatch cycle) should be placed. Since we define two parts the View here we'll refer to them as the "Layout" and the "Main". A Layout might look like:

```
<?php echo $this->render('standard_header.phtml'); ?>
<?php echo $this->main(); ?>
<?php echo $this->render('standard_footer.phtml'); ?>
```

The new `Zps_View::main()` method (`Zps_View` is a subclass of `Zend_View` to which we can add customised behaviour) simply tells the View to render its output at this location in the template. This assumes the default `Zend_View::render()` method now takes a two step approach to rendering (this is where the Two-Step View Pattern comes into play).

1. Render a Layout if one is defined
2. Render the Main template into the Layout

The only funky logic is that the presence of a Layout forces our View object's render method to take a detour so that Layouts are rendered first. This is a pretty simple change to `Zend_View`. Here's our `Zps_View` class with the revised logic.

Please forgive the lack of proper phpDoc comments - Serendipity won't play nice with them.

```

// Zend_View <strong>/
require_once 'Zend/View.php';
// Zps_View_Interface </strong>/
require_once 'Zps/View/Interface.php';
class Zps_View extends Zend_View implements Zps_View_Interface
{
    //
    // The Main Template (i.e. the template file a Controller wishes
    // to render).
    // _mainFile cannot be set by a public setter so it doubles as
    // as a safety valve to prevent unwarranted use of main().
    //
    // @var string
    //
    protected $_mainFile = null;
    //
    // The Layout Template
    //
    // @var string
    //
    protected $_layoutFile = null;
    //
    // Overrides Zend_View::render() to introduce a two step view approach when
    // a Layout template has been defined. The two steps are handled using
    // separate calls to parent::render() which calls the Zend_View render()
    // method without overriding.
    //
    // @param string $name The script script name to process.
    // @return string The script output.
    //
    public function render($name)
    {
        if ($this->hasLayout() && !isset($this->_mainFile)) {
            $this->_mainFile = $name;
            return parent::render( $this->getLayout() );
        }
        return parent::render($name);
    }
    //
    // Set the filename of a Layout template to be used. The existence of a
    // Layout filename will force the over-ridden render() method to detour
    // and render the Layout, only rendering the Main template when a main()
    // call is issued in the Layout template.
    //
    // @param $file string
    // @return void
    //
    public function setLayout($file = 'layout.phtml')
    {
        $this->_layoutFile = $file;
    }
    //
    // Return the filename of the Layout. Layouts are like any

```

```

// other template script and are located in the same place in
// application filesystem.
//
// @return string
//
public function getLayout()
{
    return $this->_layoutFile;
}
//
// Returns true if a Layout has been set for this View.
//
// @return bool
///
public function hasLayout()
{
    return isset($this->_layoutFile);
}
//
// Inform the View object that it should render the Main View, i.e.
// render the template handed to the render() method by a Controller.
// This method is only useful if a Layout is being used, otherwise
// expect an Exception.
//
// @return string
// @throws Zps_View_Exception
//
public function main()
{
    if (isset($this->_mainFile)) {
        return parent::render($this->_mainFile);
    }
    require_once 'Zps/View/Exception.php';
    throw new Zps_View_Exception('Invalid call: There is no primary View template to render');
}
//
// Method to clone this View assuming the sub-View (the clone) is from
// the same application Module as the original.
// Here we are simply getting rid of the inherited public variables which
// represent the ancestor View's model.
// We also disable any Layouts (the inheritance would lead to infinite
// looping otherwise - Apache would bark and die on the spot!)
//
// @return null
//
public function __clone()
{
    foreach(get_object_vars($this) as $key=>$value) {
        $this->__unset($key);
    }
    $this->setLayout(null);
}
}
}

```


So there we go, functional code for allowing Layouts in a Two-Step View approach. Notice how the render() and main() methods interact. Because it's both have very specific uses, main() is only useable within templates when injecting the main template into a Layout.

Sample usage is pretty simple - I won't delve into any details since you just need two additional pieces of work when instantiating a View object:

1. Create a Layout template (you'll notice a setLayout() default is "layout.phtml" but you're not bound to that convention by any means.
2. Set the Layout on a View object, e.g.

```
$view = new Zps_View;  
$view->setBasePath('/path/to/default/view/directory');  
$view->setLayout('layout.phtml'); // this will force render() to perform a Two-Step  
$view->render('sometemplate.phtml');
```


If you followed my previous posts you'll be able to integrate the Two-Step View/Layout approach pretty easily into your Views. Of course, as usual, a key observation is that in using a Zend\_View subclass be sure not to rely on Zend\_Controller\_Action::initView(). You'll need to override that method in your own application specific Zend\_Controller\_Action subclass.

If the lack of a Zps\_View\_Interface worries you it's just a declaration of all the public methods above. I won't post it here, this entry is long enough 

Any final words? A similar system is also possible using an alternate implementation. Matthew Weier O'Phinney, when I originally started asking about complex views in the Zend Framework, posted a Two-Step View implementation using a dispatchLoopShutdown() plugin. You can read more about this over at the mailing list archives - here's the exact link to Matthew's email.

<http://framework.zend.com/wiki/display/ZFMLGEN/mail/27145>

Finally, this is completely compatible with using a Composite View system. You can imagine creating a component full of widgets or plugin output. Each of these would be aggregated into a Composite View. But that doesn't mean they don't all share one thing - a common layout. So Layouts (Two-Step View) and Composite Views play quite nicely together.

Have fun! 

If anyone comes up more bright ideas throw them into a comment for the hordes of blog readers to consume!

Posted by Pádraic Brady in PHP Game Development, PHP General, PHP Security, Zend Framework at 19:35

Thursday, May 17, 2007

### More Podcasts for PHP

Looks like the [Zend Developer Zone has gotten around to announcing](#) its twice weekly podcast entitled PHP Abstract. I know one or two folk who are stressing themselves out over a microphone for this one



. Good luck

to you both.

Podcasting from DevZone has a start date of June 5th from Cal.

One must be fair, also mention [PHP Architects Pro:PHP podcast](#). I haven't gotten around to hearing Ed Finkler's yet, but I did catch Morgan Tocker from MySQL AB.

I hope my iPod can handle the upcoming load of casts!

Posted by Pádraic Brady in PHP General, PHP Security at 17:35

Wednesday, May 9, 2007

## **The 2007 General Election (when Ireland gets to choose another potentially useless bunch of political leaders)**

In the red corner: Fianna Fail, The Progressive Democrats.

In the blue corner: Fine Gael, Labour, The Green Party.

The Winner? To be decided in a few weeks on a Thursday, so that anyone working from Dublin or attending University is unable to vote in their own constituency. Fianna Fail have smarts... It's going to be another low-turnout election.

I was leafing through the newspapers here today, and looking back on all the drama of the past two weeks. It's a fascinating spectacle. First our current Taoiseach (Prime Minister), Bertie Ahern, is hit by allegations of receiving (well, his partner apparently) a large sum of money from some dubious character. Since that news broke he has steadfastly refused to explain anything, the one thing guaranteed to outrage the electorate.

Then we had the whole thing where criminals in an Irish "maximum security" prison seemed to have developed a fondness for ringing up radio shows with their mobile phones. Our Justice Minister, Michael McDowell, went all out on this one. Somehow he managed to get a spot on the same radio show with 15 minutes of free air time where he was neither interrupted, questioned, nor any after-comments allowed. Now how the guy can swing that kind of deal with Ireland's national broadcaster, RTE, is beyond me.


Finally, we have the whole Health Service in a mess from a Nurse's "work to rule" protest. Just today I heard the Health Service Executive is threatening them with salary deductions of up to 30% over their protest actions. It's so cool to see negotiations going so well.

I wonder what's next? Maybe when Enda Kenny (defacto Opposition Leader, and leader of Fine Gael) stops giggling at this marvellous run of luck for his coalition of parties Gerry Adams will sweep the Polls and we'll end up with Sinn Féin looking for a place in Government. Yep, that's all we need. The Inept Party and the local Terrorist ("But we're now political!") Branch announcing an alliance to lead Ireland into the 21st Century...

Anyway, just to moan a little more... I'm hoping Fine Gael have the right stuff to get elected this time around. If they do I'll be watching for some kind of improvement.


1. To get the screwed up, ill managed, Health Service running in some halfway decent manner. It's practically pathetic living in a country which has a second-rate Health Service which thinks it perfectly normal to make people wait months for test result on ailments like cancer. Not like a fast diagnosis has any value, eh?

2. Meeting some kind of Environmental Standard. I've been told Ireland is a signatory to Kyoto. I also know we're never going to meet it, or any other target the EU sets. Fact is, the current government has no motivation to do so - since they're rolling in my tax dollars they think it's cute to do nothing and just pay for whatever carbon surplus the rest of Europe has instead. It would also be nice if the west coast got clean water (anyone intending to visit Galway should carefully watch the local water situation unless you like little critters called cryptosporidiosis).

3. Roads. Big roads which have more than four lanes of traffic. Roads which aren't packed day and night with traffic jams and those frackin' toll booths - tolls! For sitting in a traffic jam! Trains. Yep, trains which go faster than 40 mph. Maybe one of those fancy mainland ones that go like...over 50mph? God, do you guys have ones that go faster??? Is such a miracle possible??? 

If we really make the big time, maybe we'll get a few extra buses which run more frequently than once every 3 hours? I could be reaching too far there...

4. Broadband! Ah, my time honoured complaint for year after year... If you live where I do you have two choices. a) Get a satellite connection for €100+ per month, or b) wait until hell freezes over for the local telecom (well, technically it's national...and a monopoly to boot) install a digital exchange. If all else fails, one can pray that a radio mast pops up within a few kilometers, has a direct line of sight, and doesn't involve building a radio telescope on the roof. Broadband...a man can only dream of ever seeing such a wondrous technology...

5. Proper road signage. I live on a small road you can fit one car up (anyone you meet needs to be very good in reverse 

). According to the local council the speed limit for that road is a humble 80Kph. I keep expecting the Garda- to pull me over for living on the edge doing half that.

Posted by Pádraic Brady in Irishisms at 17:36

## Complex Views with the Zend Framework - Part 4: The View Factory

In parts [1](#), [2](#) and [3](#) I've been taking a look at the [Zend Framework](#) and putting together a broad picture of a potential implementation to add support for complex multi-part web pages. This refers to the practice of building a web page in an application from a number of common reusable elements. An example of such elements include header sections, footers, menu bars, widgets, etc, which surround the main content returned by any client request. In Parts [2](#) and [3](#), I introduced two useful design patterns for this purpose: Composite View and View Helper.

Here in Part 4, I revisit a characteristic of the Composite View pattern - the idea of using multiple Zend\_View objects (or subclasses thereof) to encapsulate each element of the web page. Zend\_View is rather a complex class to instantiate. Not only does it have optional settings, it also requires path information in order to locate it's templates, helpers and filters. To make matters worse, we would like for each concrete instance to have access to it's ultimate parent's Model as given to it by the Controllers which handle requests.

You can guess that there's a long block of code involved in creating a fully functional View object in the Zend Framework (check out Zend\_Controller\_Action::initView() for an idea). Most likely those using the Zend Framework have it all located in either a subclass of Zend\_Controller\_Action, or within the bootstrap index.php file, or rely on the default initView() method. This is fine for one single instance of Zend\_View - but once we get to three, four or more View objects, as needed by the Composite View pattern, this won't work.

As you can guess, we would have to duplicate the same block of setup code wherever a new View object is created. Now since duplication is an obvious code smell, and since the logic concerns object instantiation and configuration, we may identify the problem and its solution - the Factory Pattern.

In this blog entry I'll present a class (the Factory) dedicated to churning out any number of Zps\_View (subclass of Zend\_View) objects using a class called Zps\_View\_Factory. To make things more interesting we'll also centralise all those default settings a View might require into a configuration file which our Factory will be able to use. But first let's look at the UML class diagram of a possible Factory setup.



As you can see the Zps\_View\_Factory encapsulates all the steps needed to create View objects. All someone on the outside need do is instantiate a Factory object, call it's createInstance() method with some optional parameters, and receive a configured View object ready for rendering templates. Before we jump into the Factory code though, let's put all our various preferred View settings into a config file for Zend\_Config to chew on.

```
[general]
```

```
; Extracted from config.ini
```

```
; Standard View settings
view.encoding = "UTF-8"
view.escape = htmlentities
view.strictvars = 1
```

With our configuration file we improve the centralisation by putting all our settings information in one editable file. I've neglected to add path information - instead we'll rely on a convention that the View's base directory is of the form: applicationPath/moduleName/views.

Now for the Factory class. Before we go there, I've first added a Zps\_View class. This extends Zend\_View to add a pair of methods for accessing/setting the "main" View. In essence, the View our Controller layer creates. We use this as a common parameter for all Views so that composite Views can access the Controller generated Model (as granted to the "Main View") for any request-specific data a View may require.

As a hint to the future, Zps\_View has a protected \_factory() method which attempts to locate a valid instance of the View Factory from the local Zend\_Registry instance.

```
require_once 'Zend/View.php';
require_once 'Zps/View/Interface.php';
class Zps_View extends Zend_View implements Zps_View_Interface
{
    // Set a main parent View sourced from each Request's controller.
    public function setMainView(Zend_View_Interface $view)
    {
        $this->_mainView = $view;
        return $this;
    }
    // Get the main parent View.
    public function getMainView()
    {
        if (!isset($this->_mainView)) {
            return $this;
        }
        return $this->_mainView;
    }
    // Method to clone this View assuming the sub-View (the clone) is from
    // the same application Module as the original, and therefore is likely
    // a simply duplicate of its parent.
    // Here we are simply getting rid of the inherited public variables which
    // represent the ancestor View's model (isolates Views to prevent coupling).
    public function __clone()
    {
        foreach(get_object_vars($this) as $key=>$value) {
            $this->__unset($key);
        }
    }
    // Fetch an instance of Zps_View_Factory which should be present in
    // in the default instance of Zend_Registry.
    protected function _factory()
    {
        if (!Zend_Registry::isRegistered('ViewFactory')) {
            throw new Zps_View_Exception('Registry does not contain an entry for the ViewFactory');
        } else {
            $viewFactory = Zend_Registry::get('ViewFactory');
        }
        return $viewFactory;
    }
}
```

The Zps\_View\_Interface interface merely enforces the revised interface on the Zend\_View subclass - in our case the new setMainView/getMainView mutator and accessor methods.

Our Factory class isn't all that complex in the end. The configuration file, and some bootstrap sourced Registry values are the main inputs. The rest is simple enough except for the automated handling of Main View references.

```
require_once 'Zps/View/Factory/Interface.php';
require_once 'Zps/View.php';
class Zps_View_Factory implements Zps_View_Factory_Interface
{
    protected $_config = null;
    public function __construct(Zend_Config $config)
    {
        $this->_config = $config;
    }
    public function createInstance($module = null, Zps_View $view = null, array $params = null)
    {
        if (isset($module) && !ctype_alnum($module)) {
            require_once 'Zps/View/Exception.php';
            throw new Zps_View_Exception('Invalid module name; must only contain alphanumeric
characters');
        }
        if (isset($view) && is_null($module)) {
            $subView = clone $view;
        } else {
            if (is_null($module)) {
                $module = 'default'; // assume the default module
            }
            $subView = new Zps_View();
            // This is the conventional ZF directory layout. Some configuration
            // improvements could allow more flexibility here.
            $subView->setBasePath(
                Zend_Registry::get('BasePath') . DIRECTORY_SEPARATOR . $module .
                DIRECTORY_SEPARATOR . 'views'
            );
            if(isset($view)) {
                $subView->setMainView($view->getMainView());
            }
            $subView->setEncoding($this->_config->encoding);
            $subView->setEscape($this->_config->escape);
        }
        // Place holder for parameters set by a parent view on a sub-view which
        // can be used as inputs to certain View Helpers in the sub view. I'll
        // explain later .
        if (isset($params)) {
            $subView->params = $params;
        }
        return $subView;
    }
}
```

Here's the Factory being instantiated from an extract of the bootstrap file (bootstrap.php or index.php depending on setup) and being added to the Registry. You could instantiate it from anywhere really but this is the lowest central point independent of any Controllers which has direct access to the configuration file.

```
$ApplicationDir = dirname(<u>__FILE__</u>);
// Include path setup (assuming no facility to edit php.ini)
set_include_path(
    $ApplicationDir . PATH_SEPARATOR
    . $ApplicationDir . '/extends' . PATH_SEPARATOR // holds ZF subclasses
    . get_include_path()
);
// Load configuration
require_once 'Zend/Config/Ini.php';
$Config = new Zend_Config_Ini($ApplicationDir . '/config/config.ini', 'general');
$ConfigView = $Config->view;
// Create the View Factory
require_once 'Zps/View/Factory.php';
$ViewFactory = new Zps_View_Factory($ConfigView);
// Create the Registry
require_once 'Zend/Registry.php';
$Registry = Zend_Registry::getInstance();
$Registry->set('Configuration', $Config);
$Registry->set('BasePath', $ApplicationDir);
$Registry->set('ViewFactory', $ViewFactory);
```

Last of all, here's an extract from a possible subclass of Zend\_Controller\_Action. Since we have subclassed Zend\_View the default initView() method can't be used because it is tightly coupled to the class name "Zend\_View" (so it won't work with a subclass called "Zps\_View"). With our Factory in place however, it's a few short lines.

```
require_once 'Zend/Controller/Action.php';
class Zps_Controller_Action extends Zend_Controller_Action
{
    // Instance of Zend_Registry passed by Zend_Controller_Front as an
    // invocation argument.
    protected $registry = null;
    // init() method called by Zend_Controller_Action constructor to setup
    // this Action (or its subclasses). Used here to assign FC params as
    // class properties and perform general preparation.
    public function init()
    {
        $this->_initRegistry();
        $this->initView();
        $this->_initResponseHeaders();
    }
    protected function _initRegistry()
    {
        if (!$this->getInvokeArg('Registry') instanceof Zend_Registry) {
            require_once 'Zps/Controller/Exception.php';
            throw new Zps_Controller_Exception('Registry invokation argument is not an instance of type
            Zend_Registry');
        }
    }
}
```


```

    $this->registry = $this->getInvokeArg('Registry');
}
// Over-rides Zend_Controller_Action::initView()
public function initView()
{
    $viewFactory = $this->registry->get('ViewFactory');
    $this->view = $viewFactory->createInstance(
        $this->getRequest()->getModuleName()
    );
    return $this->view;
}
// Configure the default Response header "Content-Type" value.
protected function _initResponseHeaders()
{
    $characterSet = $this->registry->get('Configuration')->view->encoding;
    $this->getResponse()->setHeader('Content-Type', 'text/html; charset=' . $characterSet);
}
}

```

Last words? Using the Composite View pattern we need a way of managing the creation of a multitude of View objects. Rather than relying on the typical coupling `Zend_Controller_Action::initView()` offers, it's better to centralise the code which manages object instantiation in a dedicated Factory class. This will centralise the code, more easily allow for subclassing/editing, and prevent code duplication.

Creating the Factory within the bootstrap just maintains a single instance of the Factory in the Registry for reuse. Although it could be a static method or Singleton within `Zps_View` itself, both of these options might add some inconvenient coupling to a specific class name (much the same problem that the default `Zend_Controller_Action::initView()` method has since it explicitly refers to "Zend\_View" and is therefore in need of being overridden in any `Zend_View` subclass if you intend relying on it).

Let's try for an overall Composite View system in Part 5! 

Posted by Pádraic Brady in PHP General, PHP Security, Zend Framework at 15:08