

Tuesday, January 29. 2008

Be There, Or Be Square! The Irish PHP User Group Meetings!

Roll up! Roll up!

On Wednesday, 30 January, at 8:00PM GMT upstairs in the Longstone Pub on Dublin's Townsend Street, the Irish PHP User Group will be holding their monthly, last Wednesday of every Month, meeting.

And if there were not enough commas in the above sentence, here's another!

Infiltrating this meeting in search of secret Ninja RAILS users in need of removal, I will be joining Ken Guest and a myriad of other folk who'll help with the "removal". RAILS developers beware...

This marks my long awaited, much discussed (well, at least once) debut at the Dublin venue. If you're a PHP developer of any dubious description feel free to drop in, say hello, and show your support for the Longstone's proprietor with your cash.

If you live outside the Dublin area, additional meetings are held as follows:

Cork: Metropole Hotel on MacCurtain Street

Monaghan: Contact Kae Verens for details.

And remember, we do have a website! And it's way easy to remember!

<http://www.php.ie>

Posted by Pádraic Brady in PHP General, PHP Security at 22:32

Thursday, January 24, 2008

Kicking The Bad Habit Of Being An Overworked Paddy

It's hard to believe we are already almost 1/12 of the distance into 2008. By now all of you have broken your new year resolutions. I know I've broken several at a minimum!

After some months of desperate oft-despairing struggling with work schedules I've finally once and for all conquered my lack of free time. It's an ingenious solution - I'm taking a small break from work before rekindling an interest in financial services in these doubtful times (ask Soci@t@ G@n@ale if you want to know how doubtful, or the US Federal Rserve).

The outcome of this reorganisation of my career direction is twofold. Firstly I get extra bags of cash. Secondly, I get slightly more vacation time. Thirdly, it won't require as much overtime. Fourthly, there's less chance of last-minute-scrumbling which became exceptionally evident over the last few months as the Irish market continues to swell (in defiance of the laws of EU Economics). Of course added together this provides more of my most sought after commodity - personal time.

All that's left is how to use this new-found wealth. In between the extra pub-crawling exercises, engagements as the designated baggage mule on shopping excursions, and the other things an average 20-something is inclined to do, I want to enjoy some travel, take up writing again, and commit some completion time to the open source projects I contribute to.

I've been a very bad boy in that regard in the last six months and at one point I became an absolute nightmare for anyone who needed to contact me by email. It was not my finest hour, and I seriously doubt I escaped with a pristine reputation for being dependable. C'est la vie. A few of these "instances" shall we call them, have since been resolved to my satisfaction so I'm 95% back to nominal form as a powerhouse of innovation, inspiration and ingenuity (see, even my ego is back rockin' at full throttle!). Yep, you can always measure the normality of an Irishman by his level of self-directed sarcasm .

Anyways, enough self-critical analysis - it weakens the ego - since I'm back in fine form after two extremes (a two month vacation, and a four month chaotic period of non-stop work) I have the luxury of directing some of this time where it was always supposed to be: in supplementing my PHP experience with some open source doodling and manic self-promotion . The first target of my ire is a small project with Till Klampaeckel (Seek. Kill. Destroy.). After that is PHPSpec 0.3.0 (Exterminate! Exterminate! Exterminate!). After that is that frickin' promise-but-never-effing-do component for implementing a Yadis service (Off With His Head! Off With His Head!). I swear that thing has been sitting in a personal subversion repo begging for a few final hours of attention!

After that I'm taking a long breather, attending oodles of conferences, and finding something with a lot of words to write.

Posted by Pádraic Brady in Irishisms, Openid and Yadis, PHP Game Development, PHP General, PHP Security, Zend Framework at 19:58

Thursday, January 17, 2008

The PHPSpec Zend Framework App Testing Manifesto: ZF Integration

In the Preamble to this Manifesto, I set out the basic proposal for direct integration of support for applying Behaviour-Driven Development (BDD) to the Zend Framework. The ideal was to move away from line-by-line setup and manipulation of Zend instances towards a more simplified model which is standardised for any PHPSpec specs.

This morning I merged the experimental branch for the implementation of the Zend Framework specific PHPSpec_Context into the main trunk for release in PHPSpec 0.3.0. Documentation of the Zend Context will be completed within the next day or two and added to the growing PHPSpec Manual at <http://dev.phpspec.org/manual> now available.

The main customisation required to get the Zend Context operational is to set up elements of the FrontController, I've selected two primary methods. The first is to use several static methods on PHPSpec_Context_Zend (addControllerDirectory() and addModuleDirectory()) and a more flexible addFrontControllerSetupCallback() which will accept a custom function or static method reference to execute. The former is handy since you can easily organise your bootstrap file into a static class and segregate FrontController, Dispatcher, Action Helper, etc. elements you are using to customise how the Zend Framework operates. Both can be called from any file included into a Context file (e.g. SpecHelper.php).

Here's a really simple example of a Zend Framework spec. I'll post a more in-depth tutorial next week once the Manual is finalised, and some final cleanups applied (e.g. allowing for Router specification). The only priority for basic operations left is an implementation of a Zend_Factory class to allow you to replace objects directly instantiated within controllers (to maintain the golden standard of specifying/testing controllers isolated from it's dependencies). We'll really start kicking some ZF ass when PHPMock has an initial devel release!

```
PHPSpec_Context_Zend::addControllerDirectory('../src/application/controllers');class DescribeTwitterController extends PHPSpec_Context_Zend{    public function itShouldDisplayHomeTweetsFromDefaultRetrieveAction()    {    $this->get('retrieve');    $this->response()->should->haveText('Tweets From Twitterers Followed:');    }    public function itShouldPostTweetOnSendActionUsingPostData()    {    $this->post('send', array('message'=>'@padraicb: All your specs are belong to us'));    $this->response()->should->haveText('You submitted a new Tweet!');    }    public function itShouldRetrieveMessagesForUserIdInParamsFromRetrieveAction()    {    $this->get('twitter/retrieve/user/padraicb');    $this->response()->should->beSuccess();    }}
```

It's obvious this is bare minimum. There is still a lot of ground to cover incl. integrating PHPMock, so you can literally mock/stub the Twitter service package being used in the controller action itself (Note to Matthew - implement Zend_Service_Twitter already...) and set expectations of how the Controller should use that Twitter service. Adding some basic HTML/XML safe Matchers would also add some fairly substantial help for specifying what the response output should be, or what template any controller action should render.

But the basics, and in short the most time consuming, bit of integration to get something working with the Zend Framework at a level not requiring any devious workarounds is now complete. The weekend should see some really useful additions to this base. Your comments are most welcome as usual whether here, or on the mailing list.

Posted by Pádraic Brady in PHP General, PHP Security at 17:21

Friday, January 11. 2008

The PHPSpec Zend Framework App Testing Manifesto: Preamble

(...or some title designed to attract curious readers so I can captivate them into thinking this will not be a long torturous blog entry) Over the course of the next obsessional phase I go through I'll be attempting to pound the Zend Framework into submission so I can apply Behaviour-Driven Development (BDD) using PHPSpec when I write a Controller. Why? Because I feel like it, and it gives me an excuse to promote one possible incarnation of PHPMock and the PHPSpec Zend Framework extension. See? Perfectly reasonable selfishness!

The other reason is that whether you apply TDD, or the funky new presence of BDD in PHP, applying some form of predictive specification before you rush into PHPEclipse to start typing PHP is a good thing. The alternative is long nights debugging why Controller X and Model Y refuse to produce View Z. The immediate problem with this is that the Zend Framework currently does not offer some means of manipulating Framework operations with a simplified API, instead you have to muck about with `Zend_Controller_Front` and all sorts of other long-winded-named classes whose operation is often mysterious if you've managed to evade tinkering with them while building the Universe's next greatest social networking site.

Never fear, there are established solutions. One of the easiest to find is archived on the fw-general mailing list for the framework, a relatively straightforward post by the Master Of MVC Operations, Matthew Weier O'Phinney, using PHPUnit and an extended `PHPUnit_TestCase` class. Here's the URL in case you missed it:
How to use PHPUnit in a MVC project

For PHPSpec I hope to simplify even further. Moving from `Zend_Controller_*::someConfusingMethod()` x 5 to `PHPSpec_Context::doItAlreadyOrI'llKillSomething()`. The basis of this theoretical (unless you've found the top secret location of the subversion repo) discussion is that simple is better, and easier is way better still. To get there one needs to compact Zend Framework operations into a handful of easy to remember methods and practices. Unfortunately it does also require some Zend Framework fun by establishing a few ground rules on how you write Controller code. We'll get there much later, because it's one detail not necessary to cover in an opening sortie and it's not incredibly troublesome (I think).

To refresh your memory, PHPSpec as a BDD framework is designed to kick your ass into high-TDD gear without needing 2+ years of prior experience to get there. It's also designed to aid your thought approach by using a readable API. Besides being easy to learn, it also enforces several well-established best practices in TDD and xUnit Patterns. Here's a quickie shot:

```
class DescribeZendFramework extends PHPSpec_Context { public function itShouldMakeWebDevelopmentBetter() {
    $zend = new Zend_Framework;    $this->spec($zend->isBetter())->should->beTrue(); }}
Stored as ZendFrameworkSpec.php, you could run this spec by hitting it's location on the command line (there is a
HTML Runner option) and running:
phpspec ZendFrameworkSpecNot rocket science. Back to the topic of applying BDD... If one thinks a bit, any ZF request
is composed of only a few key components. There's the Request (POST/GET/COOKIE/URI Params), the Controller, the
Action on the Controller, and finally the Response. In between are bits and pieces which while important in a complex
application we needn't dwell on yet.
```

In a perfect world, we could wave a magic wand and drop big hints like:

```
class DescribeIndexController extends PHPSpec_Context_Zend{ public function
itShouldDisplayHelloWorldOnIndexAction() {    $this->get('index');    $this->response()->should->match("/Hello
World"); }}
The basic foundation above is simple enough. The Controller is derived from the class name, the Action is passed to the
relevant request method (get() for GET), there's a response upon which specifications can be defined.
```

In an imperfect world, it's not quite as simple as this. In applying TDD or BDD a big annoyance is ensuring we only test isolated specifics. In a typical Controller Action one can easily expect a few things:

1. Models
2. Misc Objects (e.g. `Zend_Mail`)

3. View Rendering

The biggest problem in there are Models. Models are incredibly irritating objects that insist on writing to databases. They are also as common as dirt in Controllers. Not exactly good company when applying TDD or BDD. The suggested best practice approach is to Mock or Stub these demons out of their evil ways, and turn them into predictable automatons (we can separately spec/test the actual Model classes independently). In other words, we need to Mock/Stub the dependencies of the class being tested/spec'd so it's isolated - should sound familiar as an intonation of Unit Testing practitioners worldwide.

That's the problem in a nutshell. Controllers tend to directly instantiate new objects without the benefit of good Dependency Injection. Makes them very difficult to test discretely without falling back on the double-edged sword of pure functional or acceptance testing. And that double-edged sword is quite popular in PHP right now.

Enter something like a Factory. Using a Factory, our specs/tests can potentially intercept objects used in a Controller and replace them with Mocked or Stubbed copies whose behaviour is controller by us. The simplest style would be to use a general object Factory to replace both `require_once` and `new` keywords (i.e. introduce a new Controller convention whereby we don't use the "new" keyword if avoidable in Action code). Something whose API is similar to:

```
$mailer = Zend_Factory::create('Zend_Mail');
```

A suitable `Zend_Factory` class would inherently know the `require` path for `Zend_Mail`, and would return a new `Zend_Mail` object (assume optional constructor params in the API are supported). Couple this with some backend Registry and one could allow `PHPSpec/PHPUnit` to play god with:

```
Zend_Factory::replaceClass('Zend_Mail', new Zend_Mail_Mocked);
```

Now any call to `Zend_Factory::create('Zend_Mail')` would actually return a registered Mock of `Zend_Mail`. Granted the API ignores multiple objects and non-`Zend` classes for now (an API potential discussion for another day if this goes beyond theory) but you get the point. Replacing "new Class" with "`Zend_Factory::create('Class')`" in a Controller Action would facilitate mocking by introducing a healthy dose of Dependency Injection. Let's kick another example using `PHPSpec` and `PHPUnit` (note: `PHPUnit` is undergoing development so the API is questionable as to its stability) where the `PHPSpec_Context_Zend` extension defines additional helper methods on top of the normal ones attached to `PHPSpec_Context`. This is supposed to be the revelatory part of the blog post so pay attention and critique it to death in the comments.

```
require_once 'Zend/Mail.php'; class DescribeIndexController extends PHPSpec_Context_Zend{ public function before() { $this->mail = phpmock('Zend_Mail'); $this->replaceClass('Zend_Mail', $this->mail); } public function itShouldSendAnEmailUsingPostData() { // setup the Mocked Zend_Mail instance (no real email sent of course) $this->mail->shouldReceive('setBodyText')->with('This was a test email')->once(); $this->mail->shouldReceive('setFrom')->with('anon@example.net')->once(); $this->mail->shouldReceive('setTo')->with('padraic@example.com')->once(); $this->mail->shouldReceive('setSubject')->with('Testing...1...2...3')->once(); $this->mail->shouldReceive('send')->withNoArgs()->once(); // POST request to IndexController::mailAction() $this->post( 'mail', array( 'email'=>'padraic@example.com', 'subject'=>'Testing...1...2...3', 'body'=>'This was a test email' ) ); // and because no PHPUnit integration with PHPSpec yet...ugly append $this->spec($this->mail->verify())->should->beTrue(); // we're ignoring the Response for now..oh ok then $this->response()->should->beSuccess(); $this->response()->should->match("/successfully sent/"); }}
```

Screw "new `Zend_Mail`", let's mock the devil 'imself mo chara! What works for `Zend_Mail` would feasibly also work for Models. So in theory (at a personal perspective) I like my Manifesto Preamble. Having a simplified approach to applying BDD or TDD to `Zend` Framework controllers (someone else can take on `Symfony` if they wish) is something I'd very much like to have.

Now we have no implementation, just a BDD derived specification of what the future implementation should do once complete. To round off the example, here's a possible implementation:

```
class IndexController extends Zend_Controller_Action{ public function mailAction() { // ignore security for brevity; normally we'd be damned sure $post // was the result of filtered/validated data $post = $_POST; $mailer = Zend_Factory::create('Zend_Mail'); // presto; delivers PHPSpec defined Mock $mailer->setBodyText($post['body']); $mailer->setFrom('anon@example.net'); $mailer->setTo($post['email']); $mailer->setSubject($post['subject']); $mailer->send(); $this->getResponse()->setBody('Email successfully sent.');
```

Of course this is total drivel without a) a stable `PHPUnit`, and b) `PHPSpec_Context_Zend`. I'll take a quick

Blog Export: Maugrim The Reaper's Blog, <http://blog.astrumfutura.com/>

proof-of-concept stab later after I investigate the wonders of GIT to see if it kicks SVK up the rear... In the meantime is this drivel for real, or does it have potential? Commentors who use the phrase "Mad Irishman" will be booted...

More when the PHPSpec 0.3.0 branch is active.

Posted by Pádraic Brady in PHP General, PHP Security at 23:14

Wednesday, January 9, 2008

PHPSpec 0.2.0 Released

With the beta release behind us, I'm happy to announce the immediate availability of PHPSpec 0.2.0 from the project's pear channel at pear.phpspec.org or you can download directly from <http://pear.phpspec.org/get>.

The beta release received an encouraging response and several individuals have ensured this release is up to par. I would like to thank Kubo Atsuhiko for submitting several patches which have resolved some final issues with Exception handling, and Unicode. Also Modmac who submitted the newly adopted HTTP Runner method of executing specs. And also Takagi Masahiro who has translated the PHPSpec Manual to Japanese.

PHPSpec is the first Behaviour-Driven Development (BDD) framework for PHP. It provides a low level framework for writing specs; executable examples of how the code you're designing should behave. It seeks to offer an intuitive easy to learn means of learning and applying BDD: a methodology which has evolved from the practice of Test-Driven Development, Domain Driven Design and Acceptance-Testing Driven Development.

The changelog for the 0.2.0 Stable release follows:

- Added support for HTML reporting output
- Added support for executing specs from a HTTP Runner. Thanks to Modmac.
- Committed three patches courtesy of KUBO Atsuhiko fixing several issues, include one reporting bug.
- Included Japanese translation of the manual courtesy of TAKAGI Masahiro.
- Refactored Runner logic into PHPSpec_Runner class.
- Added API Docs to documentation directory.
- Cleaned up and documented Console and HTTP options in a new manual chapter.
- Added support for coloured console output when using a *nix console (not supported on Windows).

While I'm waiting for a home page to emerge, here's some relevant links:

PEAR Channel:
pear.phpspec.org

The Manual:
<http://dev.phpspec.org/manual>

The Mailing List:
<http://groups.google.com/group/phpspec-dev>

The Google Code Home:
<http://code.google.com/p/phpspec/>

Posted by Pádraic Brady in PHP General, PHP Security at 13:52

Monday, January 7, 2008

PHPSpec 0.2.0beta Released

It's been a longer than expected road to 0.2.0, but I've finally gotten around to releasing the first beta of PHPSpec 0.2.0.

PHPSpec is a Behaviour-Driven Development framework designed from the ground up to offer a BDD tool for PHP5. It's functionality and use is heavily influenced by similar frameworks in Java, Smalltalk, .NET and Ruby. This beta release marks the start of a short review process with the aim of making a general public stable release within the next 7 days.

PHPSpec is currently hosted on the PHPSpec PEAR channel at: pear.phpspec.org. Installation is as simple as:

```
pear channel-discover pear.phpspec.org
```

```
pear install phpspec/PHPSpec-beta
```

Additional installation options (for those who don't like PEAR) and instructions are available in the PHPSpec Manual which is currently hosted at <http://dev.phpspec.org/manual>. I made the effort to keep the manual as informative as possible so it should be relatively easy to get started with PHPSpec, and discovered what Behaviour-Driven Development (BDD) means if you haven't caught one of my earlier explanatory blog articles on the topic.

This beta omits several items which didn't make the scheduled cut off point of Sunday evening which I'll complete for the GA release. The main ones are a PCRE regular expression matcher, improved Equality matcher, and something to predicate expected Exceptions. Besides these three missing bits, the goals originally set for 0.2.0 have been met and in some cases exceeded as time allowed.

PHPSpec has been one of those projects I found a real need for in my own development work so most of the design and functional decisions have tended towards a personal preference. Just a reminder that as a 0.2.0 release the API will remain stable for the life of the 0.x point releases but remains open to improvement based on user feedback and suggestions. We're also currently investigating adding a HTML UI and optionally running specs in their own unique PHP process. In addition, PHPSpec has led to two other small libraries in progress - namely PHPMock and PHPMutagen. It's been an interesting open source project to say the least, and I hope it's well received by the community.

Any questions/comments may be directed to the PHPSpec Mailing List (and Google Group) over on: <http://groups.google.com/group/phpspec-dev>.

Posted by Pádraic Brady in PHP General, PHP Security at 17:14

ext/snarl: PHP interface to the Windows Snarl Notification Tool API

It's far from rocket science, but this small extension I wrote over the Holidays allows PHP to send messages to the Snarl Notification Tool. Snarl was inspired by Growl for Mac OS and despite it's reliance on a C/C++ API (versus a network API as with Growl) isn't too hard to work with.

My intention for the API is to facilitate Window's use of a soon to be announced tool for autotesting in PHP using PHPSpec. The basic idea is that rather than switching to a console to rerun selected tests or specs for each change a developer makes, a behind the scenes PHP process automatically detects modifications to files and reruns only selected tests/specs relevant for those files, and reports results in a notification popup. The difference between this and the usual method is really simple - by avoiding switching to the console and/or running all specs including those unrelated to the change (which feasibly can take anything from a few seconds to a few minutes), you can shave time off your development process. In a nutshell, in many cases it's just more efficient and stays out of your way.

I hope to propose a completed version of the extension to PECL shortly. For the moment I've only implemented a single method to display a timed notification of some event. Obviously the extension itself is not specific to PHPSpec use - anywhere PHP would be useful in creating a desktop notification on Windows.

The source code is currently in subversion at <http://code.google.com/p/php-snarl/> on Google Code.

I haven't uploaded a compiled .dll extension yet but I'll get around to it soon. Usage will be relatively simple. The extension defines a PHP5 class called (predictably) Snarl:

```
$snarl = new Snarl;$snarl->showMessage('Title', 'Notice! Your PC is about to explode!', 5,  
'C:/icons/warning.png'); There's other parts to the Snarl API but showing timed notice messages is the most common  
use case. It's likely PHPSpec will in the future bundle similar dependencies for Mac OS Growl and KDE Notify (possibly  
depending on its API for KDE4) as part of a small abstraction library to make use across platforms easier.
```

Posted by Pádraic Brady in PHP General, PHP Security at 16:37

Tuesday, January 1, 2008

Happy New Year!

It's that time of the year again (the first day) when the idea of 2008 still seems shiny new and exciting to behold on your calendar. Go on, admit it, seeing 2008 gives you an anticipatory thrill.

It's now about two years since I shed the pseudonym Maugrim and I now have a 3 in 5 chance that any particular website won't die when I enter a-acute characters for my real name into a web form. Now that's real progress! Let's get it up to 4 in 5 this year!

Looking back it's been a fairly good year. I became far more involved in open source projects including my own meagre offerings. I've seen the rise of the Zend Framework and it's gradual evolution into a praise worthy framework - even its adoption of some of my own proposals. I finished my qualifications as a Chartered Accountant here in Ireland. I bit the bullet and kickstarted three projects to hopefully push and prod some improved TDD/BDD practices (PHPSpec, PHPMock and PHPMutagen). I became the Irish Representative on the OpenID Europe Foundation and did a ton of work to be finished soon on writing a PHP5 library for OpenID and the Yadis Protocol. As usual I typed too much on the Devnetwork Forums - looking for another Devnet Award!. I even refreshed my C knowledge which I've barely used since 2000 and started writing a small wrapper for the Windows Snarl API. Oh, and I learned Ruby which was quite an experience and I even managed to avoid Rails as a largely pointless exercise. Ruby without Rails is quite cool enough. I also had a few articles published on Zend's Devzone.

My outlook for 2008 is bright. There's a few projects in open source which will become stable and enter their maintenance mode (see above). I'll be working with the Zend Framework more on large projects, including a reentry to PHP game development. I have high hopes of attending a few conferences during the year since I should have a less hectic year. Maybe one day I'll even give in and create a Facebook account .

The coolest stuff for PHP has no doubt been already covered. PHP 5.3 is incoming with namespaces which will be welcome. PHP6 continues progressing which I'm really looking forward to since I absolutely hate all the UCS4 gymnastics I currently indulge in for parsing characters. Zend Framework should hit another major release version. Things seem on track on a much improved testing/specing landscape in PHP - PHPUnit 4 with PHPMock & Hamcrest, SimpleTest for PHP5, and my own and Travis' PHPSpec for Behaviour-Driven Development. I'm sure Ruby users will also realise why a Windows Snarl API is applicable here too . And PHPT for PEAR2 is also coming along very nicely under Travis Swicegood. I'm also looking forward to seeing how Adobe AIR influences web apps this year. After using Spaz (a pretty good poster child for AIR) by Ed Finkler there's a lot of potential there for desktop integration to web application APIs.

Here's to 2008!

Posted by Pádraic Brady in PHP General, PHP Security at 16:28