

Technical Stuff for AF

As those watching the SVN repository are aware, there have been a few changes going on outside the actual AF code. I'm going to run through these quickly.

For those not familiar with the repository layout, the distributable [Astrum Futura](#) source code is located (from the svn root) at /trunk/astrumfutura. Back to the topic at hand...

First of all, I've added a /documentation subdirectory in trunk. Documentation has always been a problem in QS and earlier in SE. For AF we're considering a policy of making each developer who makes a change responsible for updating the documentation. How we enforce that is another matter I'll discuss with my fellow developers.


Documentation is being written in the Docbook XML format and will be versioned in the mentioned documentation directory. To allow future scope for translation efforts, the actual files relevant for the default documentation are stored in /manual/en. The en directory contains a manual.xml file which is basically what determines the layout and contents. Individual pages are stored as separate XML documents in manual/en/module_specs. Common page components and miscellaneous appendixes are stored in manual/en/ref. The remaining directories and files allow the Docbook XML to be processed using xsltproc under linux (or Cygwin on Windows) to generate HTML. Options are open for using Apache fop to generate PDF (not really a concern right now). Directions for generating HTML are included in a README file in the /en subdirectory.

Secondly I have added a /standards subdirectory in trunk. The idea is to determine a final coding standard (see PEAR's PHP_CodeSniffer) and ensure its adhered to. Might seem like a pretty worthless thing to be doing right now but once the code grows to a point where global file formatting changes are time consuming it will prove its worth. The standard being adopted is adapted largely from the PEAR standard. The main difference is that we've adopted a different bracing style (opening brace on new line) and modified a few rules to be either more PHP5 specific, or simply to match the preferred styles of the current developers.

Also added, but not yet utilised for AF specific code is /tests. Since most of the classes we use are already sourced from unit tested libraries like the Zend Framework, Swiftmailer and Quantum Star (that independent game library being segregated to a separate project) this will mainly hold web tests. AF will operate strictly on a REST basis, so all requests are in some way reproducible.

Next to be added (later today) is the /build directory. The core idea here is to automate building a distributable as much as possible. Distributing is often seen as a matter of export/copy/zip. In AF it's a bit more complicated - we need to run all available tests (including all tests for libraries we are sourcing from non-stable sources like SVN trunk) including web tests, build and copy the documentation to its correct location, check the coding standard is clear of errors (ignore warnings), copy all files to a build directory in the correct locations, and finally create distributable archives for zip, gz and bz2. The distributable code must also be tagged in subversion.

I haven't decided on a final build process and am currently trying out a few tools. The one I'll be testing later is Phing (since it's relatively familiar to an Apache ant user).

Did I miss anything? 

Posted by Pádraic Brady at 06:27

Tuesday, November 28. 2006


Astrum Futura: Performance and Optimisation

I thought, as reference, I'd do up a post concerning performance and optimisation in Astrum Futura. Mainly to get my thought processes in text, and take a look at where performance is lacking.


The main driver of performance at the moment, in the absence of reams of game specific code is the Zend Framework. The ZF is currently reaching a v0.60 preview release in subversion (should be release next month) and an interesting discussion took place last week on the ZF mailing lists concerning [this blog post](#) by Paul M. Jones comparing four framework. In a simple minimalistic "Hello World!" application using each framework's most minimum setup possible, and measuring requests per second (on a local PC), the ZF came last.

The main causes of this was the pure OOP approach. Each component has a few Exception classes, each in a separate files, which were loaded (as all ZF files are) using `require_once()`. Add to that the possible use of a static loading method which did weird and wonderful I/O filesystem checks and all that work created a major drag. That may be addressed properly in the future - until a 1.0 release the idea of optimisation is still largely considered premature optimisation. At least one small improvement was made - the use of Reflection has been largely removed from the many Controller classes. Reflection in PHP5 is nice, neat, and very expensive.

The Astrum Futura approach is to wait and see. Until then we've added an `__autoload` function, and are prepared to strip all instances of `require_once()` calls from files we distribute in a release. That in combination with autoloading would ease the burden a fair bit. A release is months away of course, so wait and see is the only reasonable stance right now.

Another future performance driver is the database. This gets interesting however, in that we can't consider the database purely in isolation. By itself we can rely on a mix of query caching (on the few shared hosts who actually bother to check their MySQL configuration), the use of InnoDB and MEMORY (HEAP) storage solutions for tables (again MySQL), and the usual pretension that 4th level normalisation doesn't exist 

In AF, we also have to deal with how database results are handled. In general terms, an AJAX frontend does not typically rely on HTML input. Rather we receive responses in a strict data-only format such as XML or JSON. JSON in particular is the most useful since it's Javascript's native object notation. XML is more complex since it requires using Javascript's DOM to manipulate. One of the focuses for near static data will be utilising caching extensively - not just server side, but also in the client. One of our main concerns, and one of the reasons we're coding to PHP5.2, using MySQL more productively (no MyISAM crap here), and actively seeking ways of caching data, is the fact that the interface is AJAX enabled.

AJAX gives a nice warm fuzzy feeling when you see it in action. But if used extensively in the wrong way it can create a ton of problems. The simplest explanation is that AJAX should nearly always increase a user's productivity - it's not just eye candy 


. The more productive they are, the fewer server requests they should need for a given task. The disadvantage is that you can easily wind up with a task which rather than taking one or two big page reloads to complete, takes a dozen or more small AJAX updates. If those small updates are slow, and add up to a larger request time and larger bandwidth takeup than the single big request you started with - then something went a bit wrong. Caching both on the server side, and especially on the client side reduces the number of small requests for static data. The fewer the better...

Now, if only these good intentions converted into reality in the near future... 

Posted by Pádraic Brady in Astrum Futura at 07:47

Return of lamsure

lamsure has returned to the blogosphere over on <http://kabal-invasion.blogspot.com>.

For those not familiar with his open source developments, he's a developer on the PHP [Blacknova Traders](#) game project, and is currently planning a PHP library focused to an extent on games, called JOMPT. To add to infamy, lamsure suggested a name for my current game project which eventually won the tie-breaker after a public vote - hence we have "Astrum Futura" 

In the opening post of his new blog, he makes some general observations. He mentions Astrum Futura, and gives a quick opinion on the opening shot in this project. One note is that he'd like to see the "ability to drop-in/override certain backends with alternatives". Presumably this is directed towards the interface between the current Controllers and the ZF which is pretty tight at the moment. As an observation, I'll quickly note the ZF defines a series of Interfaces which make adding alternative backend libraries relatively simple (just need a specific Facade/Adapter for each alternative library). Largely explains why I'm using Zend_View for now instead of Template Lite which I would prefer, among other things.


A second observation is made that a few of the open source PHP games he follows (as do I) have slowed down. I suppose that's an inevitable facet of open source development in general - only the larger popular projects manage to maintain momentum in the absence of the main developers driving progress with their big picture objective setting and the help of cat-o'-nine tails... Legend of the Green Dragon lost a few of its primary developers some time ago, so until the takeover developers pick up momentum and/or get a grip on where the game is going and get an action plan together we'll likely see the current progress inch along.


Final mention goes to JOMPT, the proposed PHP library lamsure has had on the back burner for a while now (I remember it being mentioned way back when). As noted Astrum Futura does sort of hit similar objectives, though our reliance on the Zend Framework has restricted our own goals to a more specific set of functionality - primarily reusable classes that are independent, unit tested, and target specific game elements like Mapping and AI (plus more as the AF project moves along).

Posted by Pádraic Brady at 03:32

Astrum Futura to Subversion

Today marks a milestone in the [Astrum Futura](#) project. After almost a month of internal planning, wide ranging discussions and even a public vote for the project name, I have completed the initial organisation of the source code. This involved selection of appropriate libraries, the inevitable trial and error of test code, creation of a complementary Quantum Library, a few forays into the Zend Framework's components, and some upfront optimisation (nothing premature - simple things like implementing __autoload).

[Astrum Futura](#) is slated as an open source PHP game. Specifically a space strategy (no flash, just oodles of text). The open source PHP game genre has something of a stigma attached to it due to previous (even present 

) poor coding practices. Most games wind up under the management of novice programmers or were essentially completed pre-2001 (i.e. the PHP3 era) and are gathering dust along with security exploits. Others I can't even mention without squirming. But it's not a lost cause - there are a minority of seriously cool developers on the scene (some even raise heads and release stuff like [ADODB Lite](#) and [Template Lite](#) to prove it 

). AF, I hope, is one of those elite group of game projects with a serious effort and a heck of a lot of fun squatting behind it.

The source actually does very little (initial commit, just about functional), at present only allowing account signups and logins (so basic you may even cry). Even then it needs some manual setup using an SQL file and a database config file. But it paints a picture of the final setpiece we'll be working with, and offers something feedback can be provided for. You can checkout/export the current code (revision 23 at time of writing) using your svn client for a looksee from:

<https://svn.sourceforge.net/svnroot/astrumfutura>

We've also setup an svn commit mailing list, astrumfutura-svn@lists.sourceforge.net on Sourceforge. The SF summary page is located at <http://sourceforge.net/projects/astrumfutura>. Additionally, you can see a brief commit summary, and aggregate a commit log RSS feed from <http://cia.navi.cx/stats/project/astrumfutura>. I've aggregated the RSS feed on my blog for reference.

The library list being used includes:

[ADODB-Lite 1.30](#)

[Swiftmailer 2.1.17](#)

[SimpleTest 1.0.1beta](#)

[Zend Framework \(trunk\)](#)

[Quantum Library \(trunk\)](#)

* (trunk) indicates use of current development code

Big thanks go out to my fellow administrator, Lee, and also James for the week-in/week-out drudgery of

planning discussions, brainstorming, and TDD coding. Nah, it's been exciting original stuff 

. Getting this far would have been that little bit more difficult without the help of [Swiftmailer](#)'s developer, Chris Corbyn, who has setup an svn repository for his totally excellent and mind blowing mail library. Fifteen minutes of looking and I was hooked, man. Another goes out to Mark Dickenson who gave developers and performance nuts the gift of ADOdb Lite, later firing a broadside at Smarty with Template Lite.

Enough typing for now... There's a game needs developing.

Posted by Pádraic Brady at 12:38

Tuesday, November 21. 2006

AJAX Chat Application using the Zend Framework

Thanks to Cal Evans, editor of [Devzone](#), Part 1 of my tutorial on creating an AJAX enabled Chat Application using the Zend Framework has been released through [Devzone](#). Part 1 introduces the Zend Framework, describes setting it up with a basic Controller class, and fills in a few getting started issues.

Part 2 will follow next week, so stay tuned.

<http://devzone.zend.com/node/view/id/1234>

Posted by Pádraic Brady in PHP General at 06:22

Astrum Futura: An open source space strategy game project

I said I'd take a stab at describing/elaborating on Astrum Futura... Bear in mind I am a self-professed verbosity addict 

Before I do, I made a small minimalist technical release you can download to look at the source code (since SVN is awaiting an approved Sourceforge project application). The release only handles basic signups and logins, but it's enough to show how all the myriad pieces slot together. See the README file for the brief installation instructions.

[astrum.tar.gz](#)


[astrum.zip](#)

Back on track. Astrum Futura is slated as a next generation PHP web based game. I say next generation for an ensemble of reasons:

1. Written in PHP 5 (intended dep is PHP 5.2.0)
2. Loosely coupled Object Oriented Design
3. Unit Tested using SimpleTest (current CVS)
4. Will implement an AJAX interface using Prototype/Scriptaculous
5. Primary dependency is the Zend Framework (the SVN trunk version)

Notably, I did not say next-generation PHP "application". Don't go quoting me out of context now... 

For an informed professional PHP developer, these aren't new. But for an open source PHP game they almost certainly are. Show of hands; has anyone seen, installed, or worked on a truly professional-like open

source game project in PHP? I can think of only one or two (no names, no assumptions ). The rest are functional, and usually devoid of much professionalism in terms of design, good practice, and maintenance. Again, no names. An unprofessional project does not mean the developers are unprofessional - merely that they have taken shortcuts on a hobby project.

Back to AF, we've completely dropped PHP4 support - probably the one decision most folk will question. Our reasoning is quite simple. PHP5 has a vastly superior OOP model and not using it in this day and age just because shared hosts do not all support it is becoming a sad unreasonable constraint. I suspect PHP5 will see accelerated adoption - so hopefully it's not such a painful constraint except in the short term.

Now for the game description. Astrum Futura will be an open source space strategy played from a browser. Although comparisons with Quantum Star SE are likely inevitable, I'd like to cut off that line of thought immediately. You can quote me on any number of times saying QS is a poorly designed heap of spaghetti code. It always has been.

AF is not another QS/SE clone - we're starting from a blank slate. This doesn't mean all the concepts from QS Evolved are vanishing - just the poorly designed code and limitations on gameplay. AF will implement Fleets, we'll still have clans, planets and funky space opera plotlines. The problem is that QS was stuck in a rut - too many expectations, vague ideas, and dependencies from the Solar Empire era. The name change is more of a reinvention of the whole project - one I believe will lead to a vastly superior game with far more depth and scope. We want more fun, more strategy, more micromanagement (optional), more tactics, more reasons to play and more methods of winning. We want a game worth playing by addicted game junkies who'll have apoplexy if it goes offline for more than a few seconds... Where would the world be without overambitious goals?

Click the magic "more" link for the rest of this entry.

[Continue reading "Astrum Futura: An open source space strategy game project"](#)

Posted by Pádraic Brady at 02:52

Thursday, November 16, 2006

Astrum Futura: A Space Strategy Game for the Web 2.0 Era

That's the last time I ever mention Web 2.0



Following on from my previous posts about Codename Redux, the Redux project and Quantum Star Evolved, I'm pleased to announce the public voting session and a final executive decision by the developers, given a tie in the public vote, has resulted in a decision on a name.

Astrum Futura is a latin phrase meaning "future of the stars" and was suggested a few weeks ago by lamsure (the guy who helped develop Blacknova Traders and the Kabal Invasion). Many thanks to lamsure for the suggestion which beat out close competition in the final vote.

We're currently securing domain names (we have registered the usual .com, .org and .net series) and preparing to migrate from our present temporary domain. We should have forums, and a very basic website in place in a short while - the forums will come first.

I'll take a stab at making a general introduction post in a while...

Posted by Pádraic Brady at 00:09

Monday, November 13, 2006

RESTful Web Services with Zend Framework

Introduction

We've all had over a week to absorb the new Zend Framework 0.20 release. Personally I've been loving it. The Zend Framework meets my requirements with flying colours.

For an upcoming project I intend implementing a RESTful web service. Back a few months ago when tinkering with the idea of a Technorati class for the Zend Framework, a reply from Davey Shafik highlighted the existence of a Zend_Rest_Client proposal. That proposal has since been implemented, and is now available from the Framework's incubator directory. Also available is the focus of this tutorial - Zend_Rest_Server.

So let's all gather round and do some tinkering. You can download the code put together here from [restful_tutorial.tar.gz](#) or [restful_tutorial.zip](#)

What is REST?

I'll start with the obligatory REST introduction.

REST stands for Representational State Transfer (see also [Representational State Transfer](#)). In an attempt to simplify understanding REST, consider the world wide web. Using the web, all resources can be requested using either a simple GET or POST request. Each resource has a unique URI. Additionally each resource can be cached since it's (for a period at least) static. It works really well for images, webpages and such.

Applying this concept to a web service means fitting all possible requests into a small number of possible URI resources. If you take an example of a Library Web Service, you may wish to request details for a specific book. As a webservice all we're looking for is the associated data for this book (as XML). A possible URI could be:

`http://www.example.com/book/isbn/xxx` or

`http://www.example.com/book?isbn=xxx` (using a query string)

Simply, a GET request to this URI would return the books data as XML as identified by the ISBN Number. What's important is that this URI always returns the same resource - it's cacheable. It might be updated in time, but the URI is persistent.

If we want the data for a book called "Ulysses", a possible URI could be:

```
http://www.example.com/book?title=Ulysses
```

Of course we could use the Zend Framework's parameter approach using a Zend Controller route. For good measure you can contrast this approach with XML-RPC (see [XML-RPC]) where requests are themselves XML containing a method name and sometimes parameter data, etc. Cutting out such complex XML requests, and returning to basic URI usage (GET, POST, PUT, DELETE) is the main difference between REST and RPC.

This tutorial takes a look at the new Zend_Rest_Server component. In doing so we implement a simple web service, and assess for advantages and disadvantages.

Application Setup

We'll start off by implementing a simple REST server using Zend_Rest_Server. The class is part of the Zend Framework which we'll use as the basis for our mini application. To start we need to setup a basic application using the Zend Framework. The directory structure we'll use is as follows:

```
restful-tutorial/  
  /application  
    /controllers  
    /servers  
  /library  
  /data
```

The Zend Framework itself has the `./library/Zend` directory placed in the application's `./library` directory. The Framework's `./incubator/library/Zend` directory is placed within the application's `./library/incubator` directory.

In the applications root directory we place the bootstrap `index.php` file and and a `.htaccess` file. The `index.php` is as follows.

```
/**
```

Bootstrap file for RESTful Web Service Tutorial

```
*/

/*
  The basics...
  *
  E_NOTICE not used since undefined variable notices
  are reported from Zend_Server classes (see Issue
  ZF
  */
error_reporting((E_ALL|E_STRICT)& ~E_NOTICE);
ini_set('display_errors', 1); //disable on production servers!
date_default_timezone_set('Europe/London');

/*
  Setup the include_path to the ZF library
  The core library classes take precedence
  over the incubator classes
  */
set_include_path(
  './library' . PATH_SEPARATOR
  . './library/incubator/library' . PATH_SEPARATOR
  . './application/servers' . PATH_SEPARATOR
);
include'Zend.php';

/*
  Use Zend::loadClass() to load essentials
  */
Zend::loadClass('Zend_Config');
Zend::loadClass('Zend_Config_Ini');
Zend::loadClass('Zend_Controller_Front');
Zend::loadClass('Zend_Controller_RewriteRouter');
/*
  Grab database configuration
  */
$config = new Zend_Config_Ini('./data/db.ini', 'local');
Zend::register('config', $config);

/*
  Setup RewriteRouter
  */
$route = new Zend_Controller_RewriteRouter();

/*
  On my platform, I need to set the RewriteBase for ZF 0.20
  RewriteBase is assumed to be $_SERVER['PHP_SELF'] after
  removing the trailing "index.php" string
  *
  PHP_SELF can be user manipulated. Avoided using SCRIPT_NAME
  or SCRIPT_FILENAME because they may differ depending on SAPI
  being used.
  */
```

```

$rewrite_base = substr($_SERVER['PHP_SELF'], , -9);
$route->setRewriteBase($rewrite_base);
/*
  @todo add new routes for future REST requests!
*/
/*
  Setup and run the Front Controller
*/
$controller = Zend_Controller_Front::getInstance();
$controller->setRouter($route);
$controller->run('./application/controllers');

```

Our .htaccess file includes:

```

RewriteEngine on
RewriteRule . index.php
php_flag magic_quotes_gpc off
php_flag register_globals off

```

Adding Population Data to Database

Being a web service, we need data to serve! To start I'm creating one table for MySQL with prepopulated data. We'll pretend our REST service allows users to retrieve notes originally stored by a Notelt application. There will be one table - "notes". Here's the SQL for a MySQL database you can import.

```

CREATETABLE`notes`(
  `id` int(11)NOTNULLAUTO_INCREMENT,
  `user` int(11)NOTNULLDEFAULT'0',
  `tag` varchar(32) character SET utf8 NOTNULLDEFAULT'notag',
  `text` varchar(255) character SET utf8 NOTNULL,
  PRIMARYKEY (`id`)
);
INSERTINTO`notes`VALUES(1, 1, 'php', 'Need to compile PHP 5.2.0!');
INSERTINTO`notes`VALUES(2, 1, 'security', 'Check INI options for ext/filter');
INSERTINTO`notes`VALUES(3, 2, 'ajax', 'Add JSON parser to extend String in chat_server.js');
INSERTINTO`notes`VALUES(4, 2, 'zend', 'Discuss bootstrap file with Lee re: error level reporting');
INSERTINTO`notes`VALUES(5, 2, 'php', 'Note in documentation - SimpleXMLElement::addChild()
requires PHP 5.1.3');

```

Adding URIs For Client Requests

Our web service will expect clients to make GET requests for this data. Oddly enough, users will not need all the data. To allow users append conditions to their requests, while maintaining a RESTful service we will design a number of possible urls to handle such conditions.

Rather than using a query string, it'll be difficult and rely on parameters set as part of the URI. These will be parsed from the URI by `Zend_Controller_RewriteRouter`.

The basic conditions we might expect to meet are retrieving notes based on:

user

tag

user and tag

id

We can create URI paths for these. In the example urls below, `service` stands for a Zend Framework `ServiceController` class, and `note` stands for a `NoteAction()` method on that Controller. The remainder of the url constitutes a parameter list.

`http://www.example.com/service/note/user/XXX`

`http://www.example.com/service/note/tag/XXX`

`http://www.example.com/service/note/user/XXX/tag/XXX`

`http://www.example.com/service/note/id/XXX`

We could also allow a client to grab all notes for all users, or all notes for all tags - but we'll restrict ourselves to the above possibilities. By keeping the possible URIs as simple as possible we make them easy to remember - since we're using the Zend Framework we can avoid using a query string and instead rely on the internal routing process which parses that part of the URI above our parent directory (which holds `index.php`) into controller, action and parameters.

The Zend Framework's core MVC components allow us to define and place requirements on additional routes using `Zend_Controller_RewriteRouter` and `Zend_Controller_Router_Route`. If we take the first URI, we can assemble a route of the form:

```
$routes = array();
$route['byuser'] = new Zend_Controller_Router_Route(
    / Set route format /
    ':controller/:action/user/:user',
    / Set applicable defaults /
    array('controller'=>'service', 'action'=>'book'),
    / Set variable requirements (Regex) /
```

```

    array('user'=>'ld+')
);

```

Using the above as a template we can quickly create new routes for each URI we intend supporting in our RESTful web service. We can replace the previous index.php @todo comment with the following:

```

/*
    @todo add new routes for future REST requests!
*/
$routes = array();
$routes['compat'] = $route->getRoute('default');
$routes['byuser'] = new Zend_Controller_Router_Route(
    ':controller/:action/user/:user',
    array('controller'=>'service', 'action'=>'note'),
    array('user'=>'ld+')
);
$routes['bytag'] = new Zend_Controller_Router_Route(
    ':controller/:action/tag/:tag',
    array('controller'=>'service', 'action'=>'note'),
    array('tag'=>'lw{3,32}')
);
$routes['byusertag'] = new Zend_Controller_Router_Route(
    ':controller/:action/user/:user/tag/:tag',
    array('controller'=>'service', 'action'=>'note'),
    array('user'=>'ld+', 'tag'=>'lw{3,32}')
);
$routes['byid'] = new Zend_Controller_Router_Route(
    ':controller/:action/id/:id',
    array('controller'=>'service', 'action'=>'note'),
    array('id'=>'ld+')
);
$route->addRoutes($routes);

```

ServiceController Class

Ignoring IndexController, we'll run our service from a NoteAction() method on a ServiceController class, i.e. URI after our root url will be /service/note.

With our valid routes defined, we can now implement the ServiceController class. The Controller will currently only require a single method - NoteAction(). The new Action will access the parameters parsed from the URI, and setup a Zend_Rest_Server instance to handle the current request.

The actual work entailed in gathering the data needed to build a response is offloaded on a standalone class we'll call App_Rest_Server_Note. In the absence of an application name, App_ is as good a class prefix as any. This class will hold all the specific methods Zend_Rest_Server may need to call.

The ServiceRestController (saved as ServiceController.php to ./application/controllers):

class ServiceController extends Zend_Controller_Action

```
{
    public function IndexAction()
    {
        / For anything other than a call to service/note
        redirect to IndexController
        */
        $this->_redirect('/');
    }

    public function NoteAction()
    {
        /*
        Fetch Parameters and Parameter Keys
        We don't need the controller or action!
        */
        $params = $this->_getAllParams();
        unset($params['controller']);
        unset($params['action']);
        $paramKeys = array_keys($params);

        /*
        Whitelist filter the Parameters
        */
        Zend::loadClass('Zend_Filter_Input');
        $filterParams = new Zend_Filter_Input($params);

        /*
        Build a request array, with method name to call
        on handler class for REST server indexed with
        'method' key.
        *
        Method name is constructed based on valid parameters.
        */
        $paramKeysUc = array();
        foreach($paramKeys as $key)
        {
            $paramKeysUc[] = ucfirst($key);
        }
        $methodName = 'getBy' . implode("", $paramKeysUc);
        $request = array(
            'method'=>$methodName
        );

        /*
        Filter parameters as needed and add them all to the
        $request array if valid.
        */
        foreach($paramKeys as $key)
        {
            switch($key)
```

```

    {
        case 'tag':
            $request[$key] = $filterParams->testAlnum($key);
            break;
        default:
            $request[$key] = $filterParams->testDigits($key);
    }
    if (!$request[$key])
    {
        // need better handling of filter errors for a real webservice...
        throw new Exception($request[$key] . ' contained invalid data');
    }
}

/*
  Setup Zend_Rest_Server
  Use App_Rest_Server_Note as handler
  */
require_once 'Zend/Rest/Server.php';
require_once 'App_Rest_Server_Note.php';

$server = new Zend_Rest_Server;
$server->setClass('App_Rest_Server_Note');
$server->handle($request);
}
}

```

The NoteAction() method isn't hugely complicated. It grabs an array of all parameters obtained from the URI. Once we have the parameters, we need to filter them since they are after all user input data. Once filtered, the filtered data is built into a new \$request array. We additionally add an extra 'method' entry to this array containing the method to call on the handler class.

With parameters filtered, we then instantiate a Zend_Rest_Server object and hand the \$request array to it. We also provide the Server instance with the name of the class it should use to handle the request and gather the data required for (a response).

There are some complications however. Our original assumption on the service would be that the parameters from the client can vary. Usually we could do something simple in the handler method on App_Rest_Server_Note like use func_get_args() to cope with variable numbers of parameters. Unfortunately, Zend_Rest_Server is very strict - you must have one public method for all possible parameter combinations on the handler class. In addition, the naming of such parameters must be identical to the names of the original parameters (i.e. see the variable requirements on our preset routes in index.php).

The only way to get around this, is adding additional code to compensate.

Before we go further, we also need a default IndexController class:

```
class IndexController extends Zend_Controller_Action
```

```
{  
    public function IndexAction()  
    {  
        $this->_forward('Index', 'noRoute');  
    }  
    public function noRouteAction()  
    {  
        echo'The Notes web service is accessible from /service/note/.<br/>',  
        'The URI requires additional parameters to be ',  
        'passed appended to the URI in the form:<br/>',  
        '/service/note/param1/value1<br/>The accepted parameters are: ',  
        'tag, user, and id.';  
    }  
}
```

App_Rest_Server_Note Handler

To gather the data needed by Zend_Rest_Server to create a response, we will create an App_Rest_Note_Handler class. Because of the above limitations, we must have a specific public method for all possible parameter combinations. To avoid code duplication (our service only runs a simple SQL query with a varying WHERE condition), these simply delegate to a private getNotes() method.

The App_Rest_Server_Note class (saved as App_Rest_Server_Note to ./application/servers)

```
/*  
    Server Class set in Zend_Rest_Server::setClass()  
    *  
    Since Zend_Rest_Server does not support variable parameter counts, I have to create specific public methods to handle the possible parameters to be passed. In addition the parameter variable names must be identical to the variable name used when setting up these routes in the bootstrap index.php file. These all pass off the end task to the private getNotes().  
*/
```

```
class App_Rest_Server_Note
```

```
{  
    /*  
        /service/note/user/xxx  
    */  
    public function getByUser($user)  
    {  
        return$this->getNotes(array('user'=>$user));  
    }  
    /*  
        /service/note/tag/xxx /  
    */  
    public function getByTag($tag)  
    {
```

```

    return $this->getNotes(array('tag'=>$tag));
}
//service/note/user/xxx/tag/xxx /
public function getByUserTag($user, $tag)
{
    return $this->getNotes(array('user'=>$user, 'tag'=>$tag));
}
//service/note/id/xxx /
public function getById($id)
{
    return $this->getNotes(array('id'=>$id));
}

/*
Use the passed parameters to build a WHERE
string for the necessary SQL query, run the
query and return the data for Zend_Rest_Server
to XMLify.
*/
private function getNotes(array $params)
{
    /*
    Create PDO class and connect to database
    */
    $config = Zend::registry('config');
    require_once 'Zend/Db.php';
    $dbParams = array(
        'host'=>$config->host,
        'username'=>$config->username,
        'password'=>$config->password,
        'dbname'=>$config->name
    );
    $db = Zend_Db::factory($config->type, $dbParams);
    /*
    Build WHERE part of the query from current
    parameters. This uses :key placeholders for
    binding values to the query string
    */
    $where = array();
    foreach($params as $key=>$value)
    {
        $where[] = $key . ' = :' . $key;
    }
    $whereString = implode(' and ', $where);

    /*
    Perform select query
    */
    $result = $db->query('select from notes where ' . $whereString, $params);
    $data = $result->fetchAll();

    /*
    Since Zend_Rest_Server cannot handle multidimensional arrays

```

```

    like a set of results. We need to create a custom XML response
    using SimpleXML
    */
    return$this->getXML($data);
}
}

```

Once you get over the public method handoffs (the `getNotes()` method needs to know the name of parameters so it has both keys and values when building the SQL). The rest is simple. We run the query after establishing a database connection, and proceed to build an XML response.

Building an XML Response

It's here we find another `Zend_Rest_Server` limitation. First of all, the `Server` class can create an XML response itself, but it cannot yet handle a multidimensional array. To return the results, we therefore must create our own XML response. The referenced `getXML()` method for this follows.

```

private function getXML(array $data)
{
    /*
    Create XML reponse.
    Ignores special XML characters <>'"&
    */
    $xmlString = '<?xml version="1.0" standalone="yes"?><response></response>';
    $xml = new SimpleXMLElement($xmlString);
    $dataCount = count($data);
    for($i=0;$i<$dataCount;++$i)
    {
        $note = $xml->addChild('note');
        foreach($data[$i] as $key=>$value)
        {
            $note->addChild($key, $value);
        }
    }
    $xml->addChild('status', 'success');
    return $xml;
}

```

This method can build the expected response. If no data actually exists, we'll get an empty `<response>` element. In addition we add a `<status>` element for success. `Zend_Rest_Server` will use a similar element for any failures and include a failure message.

As of 0.20, `Zend_Rest_Server` does not support JSON, YAML or other formats for a response. We're restricted to XML until support for those formats are added.

Trying It Out

With all pieces in place, including the database (edit db.ini for your details) we can try a test url:

`http://yourserver/service/note/user/2`

The response resulting from this requests (just use your browser - we're not implementing a service client):

Posted by Pádraic Brady in PHP General at 07:26

Filter Extension Issues - A Storm in a Teacup?

< ?xml version="1.0" standalone="yes"? >

<response >

<note>

A few posts are floating around the net from [Ben Ramsey](#) and [Christopher Kunz](#) about fla...(nahh!, change that term!), erm...unfinished aspects? potential issues? something?, in both the ext/filter extension and the Zend Framework's Zend_Filter_Input. This small flurry of potential feedback resulted in a [scorching post from Pierre](#) asking PHP users and developers who have not done their homework to "be humble and keep quiet".

Personally I haven't tested ext/filter. I should no doubt be shot like a rabid dog for daring to post about this for that reason alone. Commenting idly without any knowledge of an extension is obviously illegal somewhere, possibly China. Honestly, the inflammatory post was a bit much. Where's the feedback going to come from if the first few guys to blog in a large public forum about possible issues are shot down so abruptly? Sure, it's past the release date but, hey, that's when the average developer will start worrying about these things. It's the point at which the developer community at large will finally start evaluating this new addition to see how it works.

It is a bit worrying to see publicity like this so soon, and with such a public negative response following like a mighty slap from Ye Gods above. I know of only two people who have actually used ext/filter so far (both think it's great). That's what happens until it arrives enabled by default (?) in a major PHP release and developers suddenly realise it's here to stay. It'll be interesting to see what issues are raised - but it could be weeks before enough developers are using it to catch anything potentially missed (no offence! don't shoot!) by the filter developers.

Well, the whole back and forth exchange was entertaining if nothing else. Ben notes in his blog he has issues to put together and communicate to the ext/filter team so I'll be interesting in seeing where these lead. I rate this one as a **Teacup Storm** until something turns up on the internals list or an issue tracker where the developers on the ground can maul them to their hearts content.

On the Zend_Filter mention, well, 0.20 is a preview release. Issues like locale specific implementations of common filters was always going to be a long wait - I doubt the Irish telephone format is essential anyway

Posted by Pádraic Brady in PHP General at 12:21

Thursday, November 9, 2006

Just because the GPL has a loophole...

Posted by Pádraic Brady in PHP Game Development at 01:28

Tuesday, November 7, 2006

Response to the Unit Testing Experiment

Posted by Pádraic Brady in PHP General at 03:40

Rob Allen's Updated Zend Framework Tutorial

. 0.15 didn't even have an RFC compliant email regex at the time. In fact neither does 0.20 yet! Getting one together requires a reference implementation which obviously raises potential copyright and license questions about the source. Hopefully they'll rope someone with an implementation into signing a CLA soon.

Disclaimer: The above commentary is stuffed with personal comments. Live with it. Pretty please with sugar on top?

A few weeks ago I was re-introduced to Merchant Empires by my fellow Redux Project Administrator, Hades. ME as I remember it from a few years ago (shortly before the open source fuss) was a very complex and cool game. Unfortunately I was at the time starting a trainee contract (work hard, no overtime pay, peanuts for salary) and never found the time to play more.

In digging around recently by way of research I was once again reminded why open source games are vulnerable endeavors often falling victim to questionable folk who have the scruples of the average ferret. In particular, the GNU GPL has no method of enforcing redistribution of modified source code (since it's hosted, not distributed as a compiled executable).

The most popular Merchant Empires game at the moment is hosted by a group of people on the advancedpowers.com domain. My guess is that its quite a profitable game. They offer "Gold Memberships" for \$6 to raise funding. The benefit of a Gold Membership is mainly that you avoid the in-game ads which spin more money from non-member players.

What's disappointing is that this is a closed project, with no open source releases. They claim otherwise in their documentation...

"The APME Project started in 2001 as a fork of the Merchant Empires code base originally created by Bryan Burton, with the intention of creating a distributable and truly Open Source version of the game."

Fat chance.

Its already been noted that requests for the source code are ignored. Its nothing more than a group of people reaping the benefits of another's work which was placed under an open source license. Bryan Burton gets his single mention on the website in the above paragraph. There is no other hint he might hold the copyright to a portion of the current source code. Attribution is important, but I do note Bryan never added a copyright notice anywhere in the game's output - pity.

Of course the folk above probably think nothing of what they are doing. After all they have the source code on a private server - aren't they allowed to change it and turn a profit doing it? Are they?

It's actions like this that make me hope the day comes when an open source license is allowed to impose forced distribution of modified source code for web hosted applications in PHP (GPL 3 hopefully). Popular PHP web apps which decide to go open source have severe disadvantages in this respect. It's not always obvious, but it's a massive problem particularly in PHP games when installations are edited to remove copyrights, links to the open source, etc. A final quote from Merchant Empires real founder, Bryan Brunton...

"It's quite ironic that a group of players took the source to the game, created their own project labeled "Open Merchant Empires", and then proceeded, after a short period of time, to not release any source or modifications to the original source."

Ferrets, Bryan, ferrets...

Blank Filler Post for id#244 RSS export to new blog at 22 Jan 2007

I'm a few days late in getting around to reading it... Rob Allen has updated his excellent tutorial for the Zend Framework 0.20 release over on <http://www.akrobat.com/zend-framework-tutorial>.

As he notes in his [blog entry](#), he only needed to make some small changes (including his updated Zend_Config use) to migrate the tutorial from the 0.15 to 0.20 version of the framework. Version 1.1 of the tutorial is a must read if you're looking for an easy to access introduction to the Zend Framework.

Rob is the guy to blame if you ever have any Zend_Config issues

Posted by Pádraic Brady in PHP General at 03:30

I'm not titling this one!

.

I haven't posted much general entries for a while, I have a stockpile sitting on my desktop. Yep, I have a directory for nothing but random notes and nuggets of wisdom. Comes in useful sometimes if I feel like looking back on the interesting things I've read or posted about previously.

Anyways.

This one is just a quick update on progress in the Redux project. It's kept a very low profile with no forums or wiki's publicised. To review, Redux is a temporary name for a new game project. This project aims to create a space strategy in PHP which runs from a browser and utilises AJAX (heavily).

Now to the progress. Starting a new project in my experience follows a predictable curve. The first stage is the initial Enthusiasm. Everyone starts throwing around ideas, brainstorming, posting little code extracts, exploring ideas and possible technical additions. We're currently well into the second stage - Exploration.

The Exploration phase has a lot less random ideas, and a lot more focused detailed discussions. The topics I've been involved in during the current phase are largely about the Navigation and Mapping details for the project. We've covered generating coordinates, displaying maps, using AJAX to navigate, map formats (squares and hexagonal styles), the underlining mathematics of square vs hexes, and even an element of AI in the form of the A* pathfinding algorithm.

Amid all this discussion, I've been busy on Phase 3 in parallel - Planning and Initial Documentation. I have this perverse love of documentation - give me an idea, and I'll merrily write it half to death

. In this phase I've started a basic library which is focused on writing generic classes useful in future game projects we might undertake. Mapping for example is completely generic, only the specific details in The code to date is very early, but its covered by unit tests as a rule.

We've also drawn up some Coding Standards, a working Coordinate Navigation test with AJAX (see [TesterMap](#) test) and more.

As part of Phase 3, I've invested a chunk of free time to writing a tutorial to introduce the inner workings of the future source code. This is a 7 page document which introduces the Zend Framework 0.20, AJAX, PHP5's SimpleXML, the JSON response format, Prototype (the AJAX lib integrated with Ruby On Rails) and other general concepts. It builds a very simple application from scratch and should be a very informative introduction to the larger game application which uses the exact same structure, libraries and practices. As a bonus it also comprises a useful chunk of code for future use . Two birds with one stone...

I'll have it released online in the near future. It's not a project specific tutorial - other folk would find it useful I think.

Phase 4 is still to be reached: Implementation. This is where all the abstract ideas, code snippets, and initial documentation builds up enough confidence for folk to actually start coding the game in an initial form without being too worried their making a complete mess of it

Posted by Pádraic Brady in PHP Game Development at 02:27

Wednesday, November 1. 2006

Zend Framework 0.20 Preview Released

. I won't dig into the project goals and such again - but suffice it to say we're aiming for a pure OOP solution that's as easy to maintain as possible. It's very likely the PHP requirement will be deliberately high, i.e. PHP 5.2.0. This limits potential public users of course, but this project would not be finished for quite some time. Better to get a major PHP revision utilised and adapted to now than months down the line when it's a popular hosted version (and of course when PHP6 will be grabbing more attention!).

Just in time for An Samhain (that's Halloween for non-Irish folk

Posted by Pádraic Brady in PHP General at 02:19