



Friday, February 22, 2008


Eve-Online: Review of days 3 to 7 progress


To keep folk informed, I'll be pushing Eve Online blog posts (and other topics) to a new blog separate from this one. Some quick polling showed people assume this is a near pure PHP medium so a new blog for the non-PHP topics is called for. This fits in with my target of 3 blog posts a week which now won't be seen as spamming Planet PHP 

When I left off my last post I was enjoying ownership of a new Caldari Kestrel Frigate in Uitra, in the game called Eve Online, and had just joined a Corporation to further my experience and have a support network available. You can find me in-game as "Maugrim McFiriba", the character I invented in the mid-90s with the now infamous "The Reaper" nickname, and later used as my internet handle all over Yahoo from 1998. Best of luck to other PHP developers trailblazing through Eve right now - I'll add you guys as buddies in-game 

. Email me if you want in on my chosen


Corporation in the next week or two.

This week started off quietly with me outfitting my new Caldari Kestrel. After training the required skills I'm now running with 4 Standard Missile Launchers (1 is held in reserve). To add a little to my "tanking" (the ability of a ship to absorb damage for extended periods without exploding) I've fitted a Shield Hardener, a Small Armor Repairer and a Shield Booster. The first is passive, the final two eat Capacitor power. My naming is not exact - still getting the hang of having module names memorised 

. I figured out recently that I was off the mark about increasing my ship's capacitor max - I actually need to make modules I install require less by training relevant skills. In any case, I can now fit in those tanking modules and 3 Launchers without stressing my capacitor beyond it's design limits. If I disable the Armor Repairer I can put an extra Launcher online (at a safe distance from the enemy since it drains my Capacitor dry when activating!) and increase my fire power by 33%. Assuming my shields can take the return beating with the Shield Booster enabled for short periods, and I don't end up really really needing that Repair module, I can chew on level 1 Pirates from a distance of 20km or so and take out a ship with each grouped launch. I'm sure that won't last long as the Pirates get wind of my new death dealer 

More after the jump.

I've also now moved from Uitra to Perimeter. Moving to Perimeter leaves me a little closer to my Corporation mates, and just three jumps from Jita. Jita remains a lagfest but it's a huge market and some of the prices there are just lower than going elsewhere. You can get almost anything in Jita since it's a natural trading outpost at the confluence of three other star systems. If you can abide the lag at the weekend when it is bustling with commerce. Yesterday I narrowly missed a Corp member's request for a run into Jita to grab a cheap module for 20 million ISK. That's 20,000,000 ISK. As a newb I barely scrape by on a measly 435,000. If I'd had the time I could have earned an easy 1 million commission. Corps are where all the money is at - budding players need to get off the newb training wheels and find themselves a Corporation.

Mid week I received a mail from the Corp mailing list inviting members to a mining op (an operation where we strip mine anything valuable from asteroid fields and sell, sell, sell). I've accepted the invitation so this Saturday I'll be mining for the Corporation. I have my laptop on tap for some light reading/writing/browsing while my ship is stockpiling ore. It's not the most entertaining of professions but the social dynamic is good, and some teamspeak and multitasking works for me 


. Maybe it ranks with WOW's mindless grinds but it serves a purpose in Eve Online and I do have a new ship to buy later. Primarily though it's about standing - I don't want to be the newb who isn't contributing to his Corp's future success.

I do have a **lot** of writing to get started on anyway - more on that in another blog post sometime. It's a 100% taxed op, meaning all proceeds go to the Crimson Industrial and Development. Sunday's op will be for member-profit.

One of the Corp members should be loaning me a Caldari Osprey (Cruiser) for the ops which will be a massive step up from my lowly Kestrel with its meagre cargo capacity. I've spent the last two days training my Caldari Frigate skill to 4, and Caldari Cruiser skill to 3. If the op goes well, and Sunday's secondary non-taxed op does also, I will be shopping around for a new Caldari Caracal (Cruiser) to replace the Kestrel with. Since I'm on a Missile binge, the Caracal is the best Cruiser to step up for. It will be my fourth ship since starting the game, but really the first I will have purchased myself rather than having it awarded. Moving to a Cruiser should afford me more development room for a while and hopefully make the pirates on these Agent missions less of an annoyance. With a Cruiser also comes additional cargo space - I really need to read up on developing decent Salvage skills for those wrecks I leave behind on missions! It will also offer more room to expand my tanking ability, at the expense of speed and manoeuvrability obviously since a Cruiser is far larger than a Frigate.

This week's highlight was when a fellow Corp mate, Sam Savage, asked around for anyone interested in a Heavy Assault Launcher he scavenged from a wreck. I, of course, jumped at the opportunity to get at least one expensive piece of kit for that Caracal Cruiser in stock without personal cost. I met Sam down in 0.7 sec space a few jumps from Perimeter and picked up the Launcher - which is now sitting in storage in Perimeter. I have items scattered across Perimeter, Uitra and Urlen - need to grab everything and move down to the mining op location Friday night so I'm ready to rumble come Saturday afternoon. I think my Clone Beta is still sufficient in case of my imminent death (me; total newb; >1.1 million skill pts). Tip to those just starting, keep the clones updated or you'll lose excess skill points if killed!


With all these Missile Launchers available on the Kestrel I learned eventually (some would say "Doh!") to assign Launchers to multiple targets correctly. I can't believe I spent this long wondering how to switch from launching 4-missile barrages at one target, to 2x2-missile barrages against each of two targets. The solution was of course to target two ships (I need to train Targeting to pick more), activate two Launchers for the current primary target, click the secondary target's graphic in the top right of the UI, and activate another Launcher or two. Hello sitting ducks...! Being able to target more than two ships seems pointless in the Kestrel but the Caracal will have more firepower to distribute across targets.

I also learned another new cultural term "can flipper" referring to another player who attempts to steal the contents of your Canister while mining. Seems to be something players often do to attract the wrath of the owner and provoke them into a PvP confrontation. I suddenly want that Caracal real bad so I can accept a few provocations and get in some PvP experience. I still feel my newbness like a millstone around my neck at times - like when a Drake pops out of a gate marked with the yellow tinge and skull of a wanted player and you wonder if they'll pop off a few rounds at you for getting in their way 

With the move to 0.0 space (non-existent security!) with the rest of the Corporation in the near future (part of a larger scale Alliance migration), I need to get that Caracal rolling and my tanking skills pulled up. I'll ask around the other members if there's any particular specialisation that would be handy for the Corp to have. Squatting in a Cruiser will likely leave me with a lesser role than those pulling 360's in Drakes and I sure don't want to be too close to whatever causes a Drake trouble when it turns up without some preparation and suitable skills to at least make my 5 second survival expectancy count for something useful. Hopefully we smaller pilots can fill in some time doing recon in Frigates, or even Cruisers - though Cruisers may be a bit too big and slow (and expensive) for purely recon roles. I'll ask someone.

In other Eve news from random reading, I'm following sources of intel about the struggle between Tortuga, Goonswarm and their stubborn foes Band Of Brothers (BOB) in 0.0 space. Things do not look good for BOB who in the recent past of Eve were one of the most prominent Alliances. Their fleet is still nothing to be sniffed at according to the intel, but when an Alliance talks about a star system as their Alamo it can't be a good sign. I'm way out on a limb when it comes to politics in the 0.0 sectors, but I hope this isn't a sign that the internal struggles of the 0.0 Alliances are going to cease. Where will that leave me as a budding PvP player? My current Alliance "Eve Evolution" hopefully will offer plenty of

opportunities for fleet ops where I can gain practice and experience before we're pushed into a war of our own. A few fleet training sessions wouldn't go amiss. Some of this might be Alliance pushed - another thing to ask my fellow Corp mates.

Lot of newb questioning this weekend 

Posted by PÃ;draic Brady in PC Gaming, PHP Security at 15:51


Monday, February 18. 2008

My First Two Days Playing Eve Online

It's a lesser known trait of mine that I enjoy playing computer games, specifically strategy games for the PC. The only console I own is a Nintendo Wii - which is saying something since it's the first console I've owned since the Sega Genesis! So this weekend, with all the free time I have, a new broadband connection courtesy of Eircom (after replacing the crappy Netopia router they give you for free; free as in scrap metal), and a little trepidation, I joined Eve Online.

Eve Online is a MMOG set in space. There are approximately 5000 star systems, 200,000+ subscribers, and perhaps 18,000 to 45,000 players online at any one time. Since I was playing at the weekend for extended periods, I noticed the numbers peaked on my GMT clock each evening. The idea behind Eve Online is to enter the vastness of space and make a name for yourself either through combat, mining, trading, production or research. These are not however true alternatives since any player can train any skill imaginable given enough time. So the name of success is called specialisation, not class leveling...

All star systems have a security rating from 1.0 to 0.0. I spent all my time in 1.0 and 0.9 star systems. The word is that going anywhere with a 0.8 rating or less is not something a Rookie should consider for at least a few weeks. Going to 0.5 or lower could charitably be called suicide. Check out YouTube for a few videos of what happens to players who get cocky and impatient and run off to a 0.4 system to mine. It only takes a high skill player with a few missiles...

As for strategic and tactical gameplay - Eve Online rocks. It's a thinking man's dream game. You need to select skills, compare weapons and ammunition types, review Market conditions for the best regional prices (some stations can charge double the average price for items), get used to how ships scale and how to assess which you can take on (which is pretty much nothing since 1.0 sec systems are heavily overwatched by the local race's police forces and only suicides would attack you, or you them 


). Living in 1.0 space is quite safe and a more than a few corporations stay exclusively there. Even a few of the 0.0 Corporations maintain 1.0 sub-Corporations for you to join.

The game itself is beautifully rendered. Back in December CCP release the Trinity client was released which added an overall graphics update with high resolution textures. My PC was never taxed while running it. I have a pretty good gaming rig so I could easily run two clients at the same time (Eve also let's you play three characters on your single account). Th only niggle was collision detection on large objects like stations and planets. While it seems an odd flaw, I suspect it's a simple optimisation. The only annoyance it will serve is trying to reach anything on the opposite side of a station - my advice is to orbit around the station before making a straight-line approach to such objects.


My own experiences from a first weekend after the jump...Day 1: PHP Is Not A Recognised Skill...But Missiles Are!

Sad to say, but I was unable to make a living in Eve Online as a PHP programmer. Aww... After booting up the Eve client I was greeted by an Introduction video detailing some of the back story to Eve Online. It's standard sci-fi fare as Tor would publish in a cut-sized paperback. Yep, a total cliché, but the backstory just set's the scene - it can be as clichéd as it likes. Anyway, does WOW not have the same? Afterwards I input my trial account username and password, before being greeted by the character creation screen.

Creating a character for Eve Online should not be done lightly. Read up online about what kinds of attributes, skills and race/bloodline mixes are best suited to what you want. I wanted to kick other player's asses (PVP/PVE) so I went with a Caldari Achura who had taken up the Soldier profession. I added my 5 attribute points to Perception, Intelligence and Memory (2-2-1). This gave me quite a collection of skills for in the area of Missiles, Gunnery and Calderi Frigates. More besides - but these are great ones to start a Soldier with. Adding a portrait is simple with the client's facial morphing abilities though each race has distinct Earth profiles - so my Calderi Soldier has distinctly Asian features.

The game opens with a tutorial of sorts. It's slightly erratic on two fronts. Firstly it doesn't automatically warp your ship to the tutorial area all the time (so you need to warp to a station, and restart the tutorial whose steps are thankfully easy to fast forward past). Secondly, you need to take instructions literally - for example when they talk about shooting a pirate with a blaster in the first tutorial don't assume to do it now - otherwise the next screen which does say to activate your guns will freeze. Why? Because the pirate is probably dead now after a few shots, and the tutorial doesn't realise you can't activate your guns anymore (no target). That's another route ripe for a restart 

Otherwise the tutorial isn't all that bad - it gives you the basics, introduces you to an Agent who provides missions, get's you up and running.

I spent the next day running missions. During this I made sure to a) have a Clone Beta on hand and move it to any new station I decided to base myself out of, and b) make sure I insured any new ships I received. Keeping all your off-ship cargo stored in a station is a really good idea. I consolidated my assets on my chosen station twice so far. In a few more days I suspect I'll need to consider a semi-permanent central station in the region to consolidate items and other ships in. In truth I spent the whole weekend in about 4 star systems around Uitra which is not bad for a Rookie in a 5000 star system game 

I started out in a Rookie Ibis ship. Basically a tincan with a Railgun and a Mining Laser. As a Soldier, I followed Agent Missions first. After a few I was awarded a new Calderi Condor Frigate which is a bit better than a tincan but only has 2 possible weapon points which I armed first of all with Blasters, but finally with Standard Missile Launchers. Missiles appeared more effective at the time - they deal a lot of damage from long range and even gave single-shot kills. Hybrid Turrets (Guns) are shorter ranged and take more time to kill anything with (as a Calderi at least). Long ranged combat simply makes sense since you're further from the target's own guns and they tend to use lower yield Rockets for long-ranged engagements.

Most of the first day was spent learning skills (usually level 1 and 2 for stuff like Learning, Clarity, Mechanics, Science, Navigation). I found my weapon skills were already at levels 3 and 4 but supporting skills for ship systems, shield and armor upgrades, afterburners, propulsion systems and capacitor upgrades were untrained. These are essential to add new ship subsystems like hardened shields, damage control modules, sensors, capacitor upgrades, etc. One of my current concerns is power. Your ships are never fully loaded up with equipment because available power is never enough - something I need to focus on. Processing capacity seems a lot better - but I'd bet that becomes a limiting factor later too. So yeah - train those skills. If not playing for a day or two - pick a big 60hr skill to train while you're offline.

Overall impressions: It was a good start. As I suspected it's not the most exciting game full of thrills and spills. It takes patience and acknowledgment of having to plan and execute a long term strategy of skills training, ship research, corporation membership and cash inflow (Missions, Corp work, Mining, Trading, etc). I intend training skills continually - only takes a quick login to start a new one training (lv1 about 30mins, lv2 about 1.5hrs and lv3 a lot more!). I also intend working my way to a new Calderi Kestrel or Merlin which are the most powerful Calderi Frigates. I'm sticking with Agent Missions for now. My Condor doesn't have the cargo space to go mining for cash and the starting Missions are more profitable for now, as well as being part of an interesting storyline.


Day 2: Suicide Runs, More Missions, Corporation Recruitment Drives

I started Day 2 on a mission to destroy the lair of a Pirate. Outside the Station at Uitra VI Moon 4, a bunch of Rookies were annihilated in an "incident" after opening fire at another player. I also witnessed a wicked looking cargo ship shambling up to the Station for docking and had a quick conversation with the pilot. He was in from a 0.6 system to make a quick sale and get some repairs done. Dangerous places down past 0.8...

After another mission I was awarded a new Calderi Kestrel Frigate. Local chatter advised me it was the finest Calderi

platform for Missiles in the Frigate Class. I was happy enough to load it up with 4 Missile Launchers and another 3000 missiles as ammo. Ammo was seriously being chewed through by now from the more powerful Pirates I was taking Missions against. My ship's power continued to be a problem so I set to on a training track to provide skills I could use to increase capacitor output. Really need to get more supporting systems for my shields and armor in place - but first I need the power capacity to run them!

I spent a block of time spectating the local traffic. Basically just picking out ships, reading their information, and getting a feel for which ones were good, bad or indifferent. Most of the ships were Calderi, but I located an Amarr Industrial which marks a first for another species. I gather I'm so deep into Calderi space it's a rarity to find anything not Calderi yet.

After some bumming around I felt the need for a little exploration so I headed to the Jita System. Or as the local chatter were calling it: LagFest. Turns out Jita is a system at the confluence of three others which makes it a prime trading spot. It was so popular that the CCP servers in Iceland were having a bit of trouble keeping up with the populace 

. It was very laggy, but I purchased a few items at a supremely low price. It's almost a supermarket for the region with low prices and thousands of customers and a *lot*.

After my expedition of a whole 3 jumps from Uitra, I returned to Uitra VI Moon 4. One of the high points of the day was watching two Destroyers (my Kestral being a flea in comparison) start up a firefight. Between the missile barrages, shield flares, and beam weapons flying around a big crowd of Rookies gathered. Into the fray came a few extra ships including a few Frigates. It was an impressive display though one or two Rookies lost all sense of survivability and fired off some Missiles to see what would happen. With some restraint the opponents replied with a single low-yield missile. The antagonisers quickly warped away before the next one!

After some checking of ship, pilot and corporation affiliations of the battle members I realised the whole battle was a deliberate show - a publicity stunt of sorts for a local Corporation. As a recruitment drive it was a good one - Uitra is where most many Calderi Rookies end up so it was their first look at a full scale battle between 5-6 Frigates, Destroyers and Battleships. Such a show of force also underlined what players of 2-3 months can accomplish, which obviously underlines how huge a role patient plays in Eve Online. It takes real time to train up those skills and accumulate the cash for such extravagant ships and equipment.

I finished off the weekend of Eve Online by moving down to the Perimeter Star System a few jumps away. There's a Calderi Naval Assembly Station here and it's closer to my eventual possible home in another week down in Venelin. I've setup a Missile oriented skill to train which will take about 20hrs (sufficient time for me to sleep, go to work, and take my time recovering at home before logging in again for a quick check and a new skill to set training - maintenance mode on weekdays).

Final impressions: Read this <http://atomic-tourist.net/AAR.pdf> for a something interesting about Corporation play out in the real world of Eve Online in 0.0 security space. Long term is the keyword. Jumping into Eve and expecting to crush everyone inside of a few days is pure stupidity. It takes planning, strategy and research to pull off a good game. This undoubtably explains Eve continued disparate subscriber count. At 200,000 it's respectable but not even remotely close to what WOW pulls in. The fact is Eve Online is a different game. It's quite easy to get into, but whether you have a personal makeup capable of blasting away Pirates, mining for a few hours with a good book (obviously you need some in-game entertainment - which may seem counter-productive but then you're missing the point!), or trading. Personally if I take up Mining I intend on taking full advantage of the fact I have a PC which can run Eve, Firefox and Eclipse at the same time, and in the future a MacBook Pro too. Eve is not just about having "fun", it's about accomplishment on a large scale over an extended timeline. Other folk would find this all dreadfully boring and lack the patience and extended mindset to create their own fun and enjoyment from the game.

So the story so far at 2 days? Eve Online is attractive, skills can be trained offline, but it requires patience and long term commitment. The commitment needn't be time consuming either. I could play for a few hours at the

weekend, or a few minutes at a time on a week day, and it wouldn't put a dent into my free time. With skills training offline there's far less pressure to be online all the time, or even a lot of the time. Many corporations clearly state they are looking for players who play "once a month or more" even...

Posted by Pádraic Brady in PC Gaming, PHP Security at 12:35

Zend_Feed: Getting Started With Aggregating RSS/Atom Content

One of the components I spent some time working with recently was Zend_Feed, which was interesting at first, then a little irritating, and eventually compliant. In this entry I explore Zend_Feed from the perspective of someone aggregating RSS and Atom feeds with a view to building a database of uniquely identified content for later presentation in a "Planet" style application.

My first overall assessment is that Zend_Feed needs work. It is a wonderful component that can simplify your life immeasurably, but it's up against competition from third party libraries like [MagpieRSS](#) which do a better job at the one malfeasant facet of blog feeds: invalid, malformed and non-standard RSS and Atom XML. What Zend_Feed needs to breach the threshold of usability is more ability to handle the various problems you meet parsing RSS and Atom to a common range of data. It also badly needs improved documentation with a focus on examples of real use - for example I can't find a single documentation snippet or blog tutorial showing how to get a entry's actual HTML content using Zend_Feed which is problematic, possibly symptomatic, and as you'll see unintuitive. Don't feel too disheartened - it's still a powerful package with sufficient utility to get you well on your way.

This tutorial is intended to cover some basics to get you started. In fact all we create here is a simple command line script to aggregate content frequently (e.g. just set up cron to run it every hour or so) into a database for later presentation.

Setting Up Database And Models

So what common data do I want for each entry in an Atom or RSS feed?

1. Unique Identifier
2. Author
3. Title
4. URL
5. Publish Date
6. Description
7. HTML Content

Based on this, we can predict a decent Model for an Entry table in a database. More on that later, but for now, how to get this data from a feed? The first thing is to have a database or other source to get the URL for each RSS or Atom feed. I'm using a MySQL table as follows:

```
CREATE TABLE IF NOT EXISTS `blog` (  
  `id` int(11) NOT NULL AUTO_INCREMENT,  
  `url` tinytext collate utf8_unicode_ci NOT NULL,  
  `feedurl` tinytext collate utf8_unicode_ci NOT NULL,  
  `title` tinytext collate utf8_unicode_ci NOT NULL,  
  `author` tinytext collate utf8_unicode_ci NOT NULL,  
  `modified` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE  
CURRENT_TIMESTAMP,  
  PRIMARY KEY (`id`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
```

You can insert into this table as you wish. One row per blog you intend aggregating. Here's some sample

data:

url: <http://blog.astrumfutura.org>
feedurl: <http://blog.astrumfutura.com/feeds/index.rss2>
title: Maugrim The Reaper's Blog
author: Pádraic Brady

The other fields update without manual intervention.

A database table for entries may be something like:

```
CREATE TABLE IF NOT EXISTS `entry` (  
  `id` int(11) NOT NULL AUTO_INCREMENT,  
  `blog_id` int(11) NOT NULL DEFAULT '0',  
  `guid` tinytext collate utf8_unicode_ci NOT NULL,  
  `title` tinytext collate utf8_unicode_ci NOT NULL,  
  `url` tinytext collate utf8_unicode_ci NOT NULL,  
  `description` text collate utf8_unicode_ci NOT NULL,  
  `date` datetime NOT NULL DEFAULT '0000-00-00 00:00:00',  
  `creator` tinytext collate utf8_unicode_ci NOT NULL,  
  `content` text collate utf8_unicode_ci NOT NULL,  
  `modified` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE  
CURRENT_TIMESTAMP,  
  `hash` varchar(32) collate utf8_unicode_ci NOT NULL,  
  PRIMARY KEY (`id`),  
  FULLTEXT KEY `search` (`title`,`description`,`content`)  
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
```

Now that we have database tables to work with, we can use Zend_Db to create some Models. We'll store these in the usual `./application/models` directory in keeping with the default Zend Framework directory structure as `Blog.php` and `Entry.php`:

`Blog.php`

```
class Blog extends Zend_Db_Table  
{  
  protected $_name = 'blog';  
}
```

`Entry.php`

```
class Entry extends Zend_Db_Table  
{  
  protected $_name = 'entry';  
}
```

The Aggregator Script Foundation

The aggregator script is a pretty simple one executed by php on the command line. Here's a basic foundation to start from stored to `./scripts/Zend/Aggregate.php`:

```
// get root directory for app
```

```

$root = dirname(dirname(dirname(<u>_FILE_</u>)));
// set the include path for this script
set_include_path(
    $root . '/library' . PATH_SEPARATOR // The ZF
    . $root . '/application/models' . PATH_SEPARATOR // Models
    . $root . '/vendor' . PATH_SEPARATOR // Other
    . get_include_path()
);
// setup autoloading
require_once 'Zend/Loader.php';
Zend_Loader::registerAutoload();
// get initial required non-ZF classes
require_once 'Blog.php';
require_once 'Entry.php';
require_once 'HTMLPurifier.php';
class Zend_Aggregate
{
    // bootstrap this class and commence aggregation
    public static function main()
    {
        // Mini-Bootstrap
        $config = new Zend_Config_Ini(
            dirname(dirname(dirname(<u>_FILE_</u>))) . '/config/config.ini', 'general'
        );
        $db = Zend_Db::factory($config->db->adapter, $config->db->toArray());
        $db->query("SET NAMES 'utf8'");
        Zend_Db_Table::setDefaultAdapter($db);

        // Models
        $blogTable = new Blog;
        $entryTable = new Entry;

        // Aggregator Startup
        $aggregator = new self;
        $aggregator->aggregate($blogTable, $entryTable);
    }

    public function __construct() {}

    public function aggregate(Blog $blogTable = null, Entry $entryTable = null)
    {
        if (!$blogTable) {
            $blogTable = new Blog;
        }
        if (!$entryTable) {
            $entryTable = new Entry;
        }
        $blogs = $blogTable->fetchAll();
        $client = new Zend_Http_Client;
        $client->setConfig(
            array('timeout'=>30)
        );
    }
}

```

```

foreach ($blogs as $blog) {
    try {
        $feed = Zend_Feed::import($blog->feedurl);
    } catch (Zend_Feed_Exception $e) {
        echo "Failed importing feed: {$e->getMessage()}\n";
        exit();
    }
    foreach($feed as $item) {
        $entryData = $this->parseEntry($item, $blog);
        $this->_syncToDatabase($entryData, $blog, $entryTable);
    }
}
protected function parseEntry(Zend_Feed_Entry_Abstract $item, Zend_Db_Table_Row $blog)
{
    protected function _syncToDatabase() {}
}
// execute the main() method
Zend_Aggregate::main();

```

The source code above is not difficult. We use a static main() method call at the end of the file to initiate the process. The main() method creates a new instance of Zend_Aggregate, set's up items like configuration, database connection, and Model instantiation, and finally calls aggregate() on the new object.

Inside, Zend_Aggregate::aggregate() we get a list of all blogs stored in the database. For each of these blogs, we fetch its relevent feed URL. The result of Zend_Feed::import() can be iterated across to get each individual entry or item. On each entry/item we pass it to the Zend_Aggregate::parseEntry() method - which bears overall responsibility for what to do next.

Using Zend_Feed to get common data for RSS/Atom entries

So what would parseEntry() do? Well, it's main role is to parse the feed and assemble a collection of common data we require - see the list above ;). So it is also responsibility for ensuring we get that data in a common format regardless of whether the source was RSS or Atom. Here's the first thing we put in parseEntry():

```
$dom = $item->getDOM();
```

Yes, a lot of the resolution of source comparison and common format will likely require we use DOM. This gets interesting later since Zend_Feed appears to do two completely incoherent things. Firstly, it strips source of several XML namespaces (particularly "content" for RSS), but leaves in most others (e.g. "dc" for RSS). There's an alternative access method for namespaced elements which is plain weird, but which Nick Halstead reminded me of.

Let's hit our list though.

First item is getting some sort of unique id for each entry - usually it's a URL reference. RSS and Atom don't agree here, but both have some sort of id. For RSS it's the "guid" element, while Atom has an "id" element. So into Zend_Aggregate::parseEntry() we add:

```

// get a unique id
$guid = "";
if ($item->guid()) {

```

```

    $guid = $item->guid();
} elseif ($item->id()) {
    $guid = $item->id();
} else {
    $guid = $item->link();
}

```

So if we can't find one of those IDs, we'll just use the item's URL which is probably just about enough. In most cases though there will be an ID.

Next up we need a title. Thankfully, both RSS and Atom at least agree on this:

```

// fetch a title
$title = "";
if($item->title()) {
    $title = $item->title();
} else {
    $title = $blog->title;
}

```

In case I was wrong, I'll just insert the blog title instead as a placeholder. Next up is a description. Atom rarely has something for this...in which case we'll just take the entry title.

```

// get a description or similar
$description = "";
if ($item->description()) {
    $description = $item->description;
} else {
    $description = $title;
}

```

If you're following this, note that all single elements can have their enclosed values retrieved by simple calling "\$item->element()" as a method. Calling "\$item->element" (as a property) actually returns another Zend_Feed_Entry_Abstract object which we can use to traverse into element children if required.

Now for some HTML content. The problem with the content is that it comes in two forms. Content as HTML in RSS is encoded (think something like htmlentities()) while the content in Atom is merely enclosed in a CDATA block. So firstly, if it's RSS we can decode it before use. The second part is ensuring the HTML is relatively safe from XSS, and finally that it all follows the same HTML standard. In this mind boggling decision making I've just delegated to [the excellent HTMLPurifier library](#).

Now the interesting thing about content is that in RSS it's enclosed by a "content:encoded" element, i.e. it uses the "content" namespace with a URL of "http://purl.org/rss/1.0/modules/content/". Zend_Feed handles this by stripping out the namespace so we're left with an "encoded" element. Atom on the other hand just has a "content" element to start with (the HTML in a CDATA block).

```

// normalise content
$contentOriginal = "";
$content = "";
if ($item->encoded()) {
    $contentOriginal =
        html\_entity\_decode($item->encoded(), ENT_QUOTES, 'UTF-8');
} elseif ($item->content()) {

```

```

    $contentOriginal = $item->content();
}
// Purify and normalise content to XHTML 1.0 Transitional
$purifier = new HTMLPurifier();
$content = $purifier->purify($contentOriginal);

```

We also retain the original content unchanged. Later we use a md5 hash of it to detect changes (e.g. maybe the author edits their entry) and update our entries on the database.

Next, we need a URL to the specific entry we're parsing here:

```

// fetch entry item link (adjust if href holds it)
$link = "";
if($item->link()) {
    $link = $item->link();
} else {
    $links = $dom->getElementsByTagName('link');
    $link = $links->item()->getAttribute('href');
}

```

Since some RSS feeds like Planet-PHP's include the link in a href attribute, rather than the link element's nodeValue, we need to do a few acrobatics with DOM just in case.

Getting the author or creator's name is another pain in the ass. RSS 2.0 often uses a dc:creator element, with a "dc" namespace with URL of "http://purl.org/dc/elements/1.1/". Unlike with Zend_Feed's previous RSS content:encoded stripping of the namespace - it doesn't do it here at all...:(. How naughty! So we have to parse out the dc elements ourselves assuming they exist, or otherwise search for an author or creator element ourselves, remembering that for Atom it's actually a name element child of author... I feel your confusion ;).

```

// get the author name
$author = "";
$creators = $dom->getElementsByTagNameNS(
    'http://purl.org/dc/elements/1.1/',
    'creator'
);
$creator = $creators->item()->nodeValue;
if($creator) {
    $author = $creator;
} elseif($item->author() && is_string($item->author())) {
    $author = $item->author();
} elseif($item->author->name()) {
    $author = $item->author->name();
} else {
    $author = $blog->author;
}

```

If all else fails, we'll just take the author name from the blog database entry. In the above I'm using the DOM to extract a dc:creator value. Here's another Zend_Feed snippet of a shortcut - if the DOM is not your thing, you can instead get the value of \$creator using:

```

$dccreator = strval($item->{'dc:creator'});
if($dccreator && !empty($dccreator)) {

```

```

    $author = $dcreator;
}

```

I love shortcuts - but here it's almost if not quite worse than using the DOM. You still need to strval() the resulting Zend_Feed_Element value returned, check if it's empty or not, and only then assign it. Still, using this without DOM can be an advantage if DOM is one of those less than familiar extensions. To be honest, I think Zend_Feed really desperately needs a simple get() method to centralise value fetching without all these gymnastics...

Let's not forget a published date!

```

// get a publication date and normalise
$date = "";
$dcdates = $dom->getElementsByNameNS(
    'http://purl.org/dc/elements/1.1/',
    'date'
);
$dcddate = $dcdates->item()->nodeValue;
if($dcddate) {
    $date = $dcddate;
} elseif ($item->pubDate()) {
    $date = $item->pubDate();
} elseif ($item->published()) {
    $date = $item->published();
} elseif ($item->created()) {
    $date = $item->created();
} elseif ($item->updated()) {
    $date = $item->updated();
} elseif ($item->modified()) {
    $date = $item->modified();
}
$date = $this->_normaliseDate($date);

```

Yeah, dates are worse than content sometimes... We'll need to normalise the date from the differing RSS and Atom forms.

```

protected function _normaliseDate($date)
{
    $date = preg_replace("/([0-9])T([0-9])/", "$1 $2", $date);
    $date = preg_replace("/([\+|-][0-9]{2}):([0-9]{2})/", "$1$2", $date);
    $time = strtotime($date);
    if (($time - time()) > 3600) {
        $time = time();
    }
    $date = gmdate("Y-m-d H:i:s O", $time);
    return $date;
}

```

Another method for you to append to the file...

Moving right along, we have two final parseEntry() parts:

```
// get a unique content hash to detect future content changes
```

```
$hash = "";  
$arrayContent = array($title, $contentOriginal, $link);  
$stringContent = implode(' ', $arrayContent);  
$hash = md5($stringContent);
```

```
// put together result object
```

```
$result = new stdClass;  
$result->guid = $guid;  
$result->blog_id = $blog->id;  
$result->title = $title;  
$result->url = $link;  
$result->description = $description;  
$result->date = $date;  
$result->creator = $author;  
$result->content = $content;  
$result->hash = $hash;  
return $result;
```

You could return an array either, I just like objects. ;)

The rest of the tutorial after the jump!

Putting It All Together

The last part of the class is inserting and updating entries - which are called from the previous stub `_syncToDatabase()` method. Here's the full `Aggregate.php` file for your reading.

```
./scripts/Zend/Aggregate.php
```

```
$root = dirname(dirname(dirname(<u>_FILE_</u>)));  
set_include_path(  
    $root . '/library' . PATH_SEPARATOR  
    . $root . '/application/models' . PATH_SEPARATOR  
    . $root . '/vendor' . PATH_SEPARATOR  
    . get_include_path()  
);  
require_once 'Zend/Loader.php';  
Zend_Loader::registerAutoload();  
require_once 'HTMLPurifier.php';  
require_once 'Blog.php';  
require_once 'Entry.php';  
class Zend_Aggregate  
{  
  
    public static function main()  
    {  
        $config = new Zend_Config_Ini(  
            dirname(dirname(dirname(<u>_FILE_</u>))) . '/config/config.ini', 'general'  
        );  
        $db = Zend_Db::factory($config->db->adapter, $config->db->toArray());  
        $db->query("SET NAMES 'utf8'");  
        Zend_Db_Table::setDefaultAdapter($db);  
    }  
}
```

```

$blogTable = new Blog;
$entryTable = new Entry;

$aggregator = new self;
$aggregator->aggregate($blogTable, $entryTable);
}

public function __construct()
{}

public function aggregate(Blog $blogTable = null, Entry $entryTable = null)
{
    if (!$blogTable) {
        $blogTable = new Blog;
    }
    if (!$entryTable) {
        $entryTable = new Entry;
    }
    $blogs = $blogTable->fetchAll();
    $client = new Zend_Http_Client;
    $client->setConfig(
        array('timeout'=>30)
    );
    foreach ($blogs as $blog) {
        try {
            $feed = Zend_Feed::import($blog->feedurl);
        } catch (Zend_Feed_Exception $e) {
            echo "Failed importing feed: {$e->getMessage()}\n";
            die();
        }
        foreach($feed as $item) {
            $entryData = $this->parseEntry($item, $blog);
            $this->_syncToDatabase($entryData, $blog, $entryTable);
        }
    }
}

public function parseEntry(Zend_Feed_Entry_Abstract $item, Zend_Db_Table_Row $blog)
{
    // for those times when Zend_Feed lets us down...
    $dom = $item->getDOM();

    // get a unique id identifying this entry online
    $guid = "";
    if ($item->guid()) {
        $guid = $item->guid();
    } elseif ($item->id()) {
        $guid = $item->id();
    } else {
        $guid = $item->link();
    }

    // fetch a title

```

```

$title = "";
if($item->title()) {
    $title = $item->title();
} else {
    $title = $blog->title;
}

// get a description or similar
$description = "";
if ($item->description()) {
    $description = $item->description;
} else {
    $description = $title;
}

// normalise content
$contentOriginal = "";
$content = "";
if ($item->encoded()) {
    $contentOriginal =
        html_entity_decode($item->encoded(), ENT_QUOTES, 'UTF-8');
} elseif ($item->content()) {
    $contentOriginal = $item->content();
}
// Purify and normalise content to XHTML 1.0 Transitional
$purifier = new HTMLPurifier();
$content = $purifier->purify($contentOriginal);
// fetch entry item link (adjust if href holds it)
$link = "";
if($item->link()) {
    $link = $item->link();
} else {
    $links = $dom->getElementsByTagName('link');
    $link = $links->item()->getAttribute('href');
}

// get the author name
$author = "";
$creators = $dom->getElementsByTagNameNS(
    'http://purl.org/dc/elements/1.1/',
    'creator'
);
$creator = $creators->item()->nodeValue;
if($creator) {
    $author = $creator;
} elseif($item->author() && is_string($item->author())) {
    $author = $item->author();
} elseif($item->author->name()) {
    $author = $item->author->name();
} else {
    $author = $blog->author;
}

```

```

// get a publication date and normalise
$date = "";
$dcdates = $dom->getElementsByTagNameNS(
    'http://purl.org/dc/elements/1.1/',
    'date'
);
$dcdate = $dcdates->item()->nodeValue;
if($dcdate) {
    $date = $dcdate;
} elseif ($item->pubDate()) {
    $date = $item->pubDate();
} elseif ($item->published()) {
    $date = $item->published();
} elseif ($item->created()) {
    $date = $item->created();
} elseif ($item->updated()) {
    $date = $item->updated();
} elseif ($item->modified()) {
    $date = $item->modified();
}
$date = $this->_normaliseDate($date);

// get a unique content hash to detect future content changes
$hash = "";
$arrayContent = array($title, $contentOriginal, $link);
$stringContent = implode(' ', $arrayContent);
$hash = md5($stringContent);

// put together result object
$result = new stdClass;
$result->guid = $guid;
$result->blog_id = $blog->id;
$result->title = $title;
$result->url = $link;
$result->description = $description;
$result->date = $date;
$result->creator = $author;
$result->content = $content;
$result->hash = $hash;
return $result;
}
protected function _syncToDatabase(stdClass $entryData, Zend_Db_Table_Row $blog, Entry
$entryTable)
{
    $select = $entryTable->select()->where('guid = ?', $entryData->guid);
    $row = $entryTable->fetchRow($select);
    if ($row && $row->hash !== $entryData->hash) {
        $entryData->id = $row->id;
        $this->_updateEntry($entryData, $blog, $entryTable);
    }
    $this->_insertEntry($entryData, $blog, $entry);
}
}

```

```

protected function _insertEntry($entryData, Zend_Db_Table_Row $blog, Entry $entry)
{
    $data = get_object_vars($entryData);
    $entry->insert($data);
}

protected function _normaliseDate($date)
{
    $date = preg_replace("/([0-9])T([0-9])/", "$1 $2", $date);
    $date = preg_replace("/([\+\-][0-9]{2}):([0-9]{2})/", "$1$2", $date);
    $time = strtotime($date);
    if (($time - time()) > 3600) {
        $time = time();
    }
    $date = gmdate("Y-m-d H:i:s O", $time);
    return $date;
}

}
Zend_Aggregate::main();

```

To use the script, you just need to navigate on the command line to `./scripts/Zend` and run: `"php Aggregate.php"`. When the execution completes, you'll have a database of ten entries from my blog. The rest, as they say, is up to your imagination. The current script really needs to be refactored into a small script component containing much of the `Zend_Aggregate::main()` code while the rest of the `Aggregate.php` file should be moved to `./library` as a Zend Framework parallel class. This way you can refactor and reuse `Zend_Aggregate` elsewhere in an application.

In Summary

So what are some things to remember?


- RSS is commonly malformed - the parsing above will likely need extra treatment at some point
- `Zend_Feed` only strips out the "content" namespace; everything else needs a `DOMElement::getElementsByTagNameNS()` operation or the shorter `$entry->{'ns:element'}` syntax property call
- Since `Zend_Feed` strips some namespacing, RSS (not Atom) content is grabbed using `Zend_Feed_Entry_Abstract::encoded()`, not `content()`.
- Dates always need to be normalised to a common format
- Always hash content for later change detection before you normalise it!
- At present some Atom feeds may not be importable into `Zend_Feed` (I'm trying to figure out why...)
- Use `HTMLPurifier` for HTML normalising and filtering
- Manually check feeds for screwups before adding to the database (if possible) or find someone who can

add better RSS/Atom parsing rules ;) (not me!)

Posted by Pádraic Brady in PHP General, PHP Security, Zend Framework at 15:55

Wednesday, February 6, 2008

The Zend Framework and Microformats: Zend_Microformat Proposed

I sometimes get stick for complaining about the Zend Framework, but it's only because I care 

To show some more adoration, I've proposed Zend_Microformat on the ZF Proposal Wiki at [Zend_Microformat - Pádraic Brady](#). I've also started a proposal page for Zend_Microformat_Xfn.

I've had Microformats on the back burner as a suitable proposal since last Summer, and since it still seems like a long overdue proposal to round out the Zend Frameworks web services offerings, I'm shooting it out there.

If you're not familiar with Microformats, they are a collection of data encodings generally included in websites using highly simple syntaxes in plain old HTML, XHTML or XML. One of the most prevelant these days is the XHTML Friends Network (XFN) microformat which is extremely common these days and can provide a wide field of collectable data about social network users and their network of friends and colleagues. Others you may be familiar with include hCard and hCalendar.

You can read more about Microformats at <http://microformats.org/about/>.

The new proposal is now open for community comments, and I'll spend some of tomorrow adding more use cases showing how the Zend_Microformat component could be used in practice for both parsing and generating Microformatted data.

Posted by Pádraic Brady in PHP General, PHP Security, Zend Framework at 00:53

The Zend Framework, Dependency Injection and Zend_Di

A while ago I wrote this neat little subclass of Zend_Loader to add a midgen of Dependency Injection to the Zend Framework application I was then building. After emailing the lists to propose it be fully adopted into Zend_Loader some time ago, I realised someone had proposed a novel new component called Zend_Di, and that small change was since accepted by Federico into his DI buster. This is a quick overview with very simple use cases - read the full proposal for even more gory details about your object innards using DI...

http://framework.zend.com/wiki/display/ZFPROP/Zend_Di++Dependency+Injection+Container

But what is Dependency Injection (DI)? And why should you care?

Dependency Injection is both the ultimate bane and blessing in PHP programming. If you're an experienced object oriented programmer, chances are you already know what the term means, and why it's an all-consuming obsession. If you don't, then here's an overview.

When you start mucking about with objects you eventually realise that the best way to get objects working together towards a unified objective, is through composition. In other words, objects use other objects, which in turn can use others. You end up not with a restrictive inheritance tree, but a pool of objects injected into each other to build up an overall purpose. The problem of course is how to inject one object into another!

We can identify a few methods:

1. Pass objects in via a constructor
2. Pass object in via a setter
3. Let some external object manage it

The first two are prevalent in the Zend Framework and for good reason - it helps ensure source code can be maintained in a highly decoupled state. Which make it easier to subclass the Zend Framework to death ;), and modify it's components before use.

The third is often confused with the Singleton and Registry design patterns. In short, people sometimes think that banging all dependencies into a Registry and then retrieving from within objects is the only extent of dependency injection required. Now it's quite true a Registry goes a long way, but let's remember the Registry has to get into the object before you used it. FYI - you probably passed in through a setter (perhaps as a Front Controller user parameter) or are using the Registry class as a Singleton Registry (calling Zend_Registry::get() statically for example). Basically, the object's dependency becomes the Registry... And let's assume the Registry is only useful for objects used very frequently by all Controllers. And then let's assume mocking objects from a static scope is less then exciting...

A neat example I like to use is that of a Controller. Let's say you wanted to create a Controller Action which fires off an email to your address. We can make a few simple assumptions:


1. Only a few Controller Actions need Email support
2. An instance of Zend_Email won't be in a Registry
3. We'll assume Zend_Mail's transport was setup previously (via static calls on Zend_Mail)

Here's the sample Controller (accessed from <http://example.com/email/developer>):


```

class EmailController extends Zend_Controller_Action
{
    public function developerAction()
    {
        $mail = new Zend_Mail;
        $mail->setBodyText('This is the text of the mail.');
```

Small problem, how do we mock Zend_Mail?

It's an increasingly common practice in the real world to apply Test-Driven Development (or to a much lesser barely existing extent in PHP, Behaviour-Driven Development (BDD) ) , before writing implementation code.

Part of those practices is to isolate the system under test (SUT), or in BDD parlance to identify the behaviour being specified.

In our case, it's simply that the Controller should send an email. Do we need to actually send an email? Well, if you really want to test Zend_Mail and don't trust Simon Mundy...  . Otherwise we should either stub, or

mock, Zend_Mail out of the system.

How about?

```

class EmailController extends Zend_Controller_Action
{
    public function developerAction()
    {
        $mail = Zend_Di::create('Zend_Mail');
```

Hey?! Where's my Zend_Mail "new" keyword vanished?

The above is a viable use case for the new Zend_Di proposal. In short, Zend_Di offers a level of indirection, whereby you use an external system (Zend_Di) to create objects while your testing framework (or PHPSpec) can access the same external system to implant a replacement Mock Object when executing tests. In this simple case, it's basically a proxy to Zend_Loader::loadClass() and a little Reflection.

If you can bang Zend Framework ops into PHPUnit, it works there too. For now indulge a PHPSpec developer:


```

class DescribeEmailController extends PHPSpec_Context_Zend
```

```

{
    public function itShouldSendEmailToDeveloperOnDeveloperAction()
    {
        $mockMail = PHPMock::mock('Zend_Mail');
        Zend_Di::replaceClass('Zend_Mail', $mockMail);
        $mockMail->shouldReceive('setBodyText')->with('This is the text of the mail.')->once()->ordered();
    ;
        $mockMail->shouldReceive('setFrom')->with('somebody@example.com', 'Some Sender')->once()
->ordered();
        $mockMail->shouldReceive('addTo')->with('somebody_else@example.com', 'Some Recipient')->
once()->ordered();
        $mockMail->shouldReceive('setSubject')->with('TestSubject')->once()->ordered();
        $mockMail->shouldReceive('send')->withNoArgs()->once()->ordered();
        $this->get('developer');
        $mockMail->verify();
    }
}

```

We're cutting it fine by omitting configuration of the email details, but you get the point. To write a Controller test or spec, you really need to isolate the Controller by mocking its dependencies to ensure the Controller behaves as expected, and interacts with Zend_Mail as expected. We already know ZF has unit tests for Zend_Mail. Now you could take the route of functional or acceptance testing, but since that's for a different purpose (client req's met?) it's not that useful in a TDD or BDD session when the View doesn't even exist yet and this is not a final Controller version 

But that's just Zend_Di with a little indirection. What about this?!

class FormatController extends Zend_Controller_Action

```

{
    public function boldemphasisAction()
    {
        $text = $this->getRequest()->text;
        $formatter = new Text_Bold(new Text_Emphasis($text));
        $this->view->text = $formatter->toString();
    }
}

```

Decorating some text with HTML/CSS is just a simple idea... But how to mock this entire construct in a controller? Or even more relevant, how to change the precise Text_* classes being used without editing the code every time it needs modification?

Well...

./config/di/format/boldemphasis.php

```

return array(
    'Formatter' => array(
        'class' => 'Text_Bold',
        'arguments' => array(
            '__construct' => 'Emphasis'
        )
    ),
);

```


```
'Emphasis' => array(
    'class' => 'Text_Emphasis'
);
```

And back to our controller...

class FormatController extends Zend_Controller_Action

```
{
    public function boldemphasisAction()
    {
        $config = new Zend_Config( require '/path/to/config/di/format/boldemphasis.php' );
        $di = new Zend_Di_Container($config);
        $formatter = $di->loadClass('Formatter')->newInstance();
        $this->view->text = $formatter->toString();
    }
}
```

Now if we ever intend boldemphasisAction() to perform a dozen other formatting steps, we can just stick them into the DI config file without editing the actual code in the controller. For a simple example, the usefulness is limited - but in a more complex web of objects you can see the benefit more clearly. Especially if using a similar web of objects numerous times with few differences.

Besides this sudden burst of flexibility, how does it help in testing? And indeed how do we ensure not to overuse it for testing (the trick question plaguing Spring... 

). Obviously I've introduced more dependencies into the Controller (i.e. Zend_Config is directly instantiated). Secondly, how can we influence the Zend_Di_Container to substitute mock objects for the real ones it would normally instantiate?

The first is an easy:

```
Zend_Di::loadClass('Zend_Config',
    require '/path/to/config/di/format/boldemphasis.php');
```

Which is really the only right answer for a simple use case. You see, Zend_Di doesn't get mocked. It's a Dependency Injection container, which we use as part of the overall testing platform. The only thing we have to change, is the configuration it uses...so that it instantiates Mock Objects or Stubs we create in our test cases (or spec) instead. Perhaps using (in a test):

```
Zend_Di::replaceClass('Zend_Config', $config);
```

Where \$config is a modified configuration for Zend_Di containing references to some Mock objects for the Text_* classes we wish to substitute into the Controller.

As for overuse - DI works wonders in small measures. There will always be a tipping point where the benefits of a DI container are outweighed by convenience, a point usually close to where mocking an object provides little real benefit. Personally, I think DI containers work wonders for Controller development in a TDD or BDD environment, even better with a good mocking framework available!

Posted by Pádraic Brady in PHP General, PHP Security, Zend Framework at 16:10

Zend_Yaml; Gone the way of the Dodo...

After my previous blog entry about escaping the trap of being overworked to the point I was failing to maintain a respectable sense of responsibility for getting things done in my open source life, I've been going through all those open source projects and cutting the dead weight. It should pay dividends before the weekend with such achievements as completing development of PHPSpec_Context_Zend, PHPMock 0.1, and committing Zend_Service_Yadis before I start writing its belated unit tests.

One of the victims of this review has been Zend_Yaml. Earlier this morning I found an odd comment on the Zend_Translate_Yaml proposal by Thomas Weidner that he was erasing his proposal on the basis that there had been no progress on Zend_Yaml. Now enough is enough, so I read through the past comments and found Zend's review. The review dated 2 January 2008 stated:

- Work with Padraic Brady to ensure that Zend_Translate_Yaml's syntax will be supported by Zend_Yaml.
- If Padraic is unable to move Zend_Yaml forward, create Zend_Yaml with support for parsing the syntax subset necessary for Zend_Translate_Yaml; future development could then take care of any other portion of the syntax we wish to support. To remove any similar confusion and perhaps stem the tide of emails over YAML, I have erased Zend_Yaml from the proposal page once and for all, and here's why.


On the 15th November 2007, I received an email indicating that Zend_Yaml had been reviewed, and some concerns had been raised. Notably, the component required adherence to the YAML 1.1 specification and it was pointed out this would not perform well unless an extension existed supporting YAML 1.1. On this basis it was recommended the proposal await a future YAML 1.1 extension.

I replied that I agreed a full parser was no longer viable, esp. since in between the original proposal and it's eventual review, the Syck extension for YAML had been accepted into PECL which was a huge push towards YAML support in PHP. I suggested a lightweight subset of YAML be supported.

I did not receive a response, so the status from my perspective was that Zend_Yaml had been effectively turned down until support for YAML 1.1 was available via an extension for PHP. Since that was unlikely to happen soon - Zend_Yaml was pretty much dead in the water.


So for those not paying attention - Zend_Yaml has been out of the picture since November 2007.

In December 2007, Eric Coleman posted a question about the status of Zend_Yaml and Zend_Service_Yadis. ZSY is easy to explain - overworked Paddy fumbled the ball horribly and rather disappointingly on that one so it's currently released on PEAR, but not yet in the ZF. Wil Sinclair responded with the following on December 20th:

We just haven't scheduled a team meeting to go over them yet. As you've probably noticed, we've been pretty busy working on the 1.5 release. [...] but the fact is they are still there because we've got a lot on our plates. If Paddy were a squeakier wheel they would probably get through faster. 

I got a bit "squeaky" after reading it, though honestly I'm only getting around to applying a little oil to remove squeaks this month, and pointed out Zend_Yaml was already reviewed, and I had responded with suggestions. Predictably, there was nothing further. Which goes right back to a blog post I made sometime ago about [the Zend Framework's idea of a proposal process](#).

Now this is twice my name has been dropped as the ignoramous holding up Zend's YAML support. I fully confess to being an irresponsible ass over Zend_Service_Yadis which has sat in PEAR for months without a Zend Framework migration as I struggled between work and other stuff, but pointing the finger at the author

of a rejected proposal nobody else at Zend seem to realise has been rejected is a bit tiresome 

So guys, just to clarify things...I've deleted the proposal. This will ensure everyone figures out it's been rejected and I'm not holding it up, and that using it as a review point for future YAML using proposals is thereby pointless. Maybe someone can now re-review Thomas' Zend_Translate_Yaml proposal and give it a second lease on life before it too joins the Great Dodo in the sky...

Posted by Pádraic Brady in PHP General, PHP Security, Zend Framework at 13:02