


Thursday, February 22. 2007

Zend_Service_Yadis Proposal

As part of the overall OpenID support goal, I have just formalised my proposal to support a Yadis Service in the Zend Framework and added to the ZF Wiki. If you wish to review and comment, the url is: [Zend Framework Proposals: Zend_Service_Yadis - Pádraic Brady](#). If you don't have a Wiki account you can add your comments to this blog entry.

The current approach focuses exclusively on the URL. I know there has been lots of progress regarding XRI, i-names, ILI and even an SMTP extension to Yadis however these will be dealt with once the main specification is implemented assuming demand and providers exist. Of course this all assumes the Proposal will even be approved 

. As some are aware the roadmap to Zend Framework 1.0 is very tight, so my main aim is get the proposal to the Laboratory stage along with a few others I'll propose in coming days and weeks.

Post 1.0 should hopefully see these components then reaching the Incubator within Zend Framework releases. In the meantime I'll be setting up a more independent Subversion repository - a lot of the code has been occupying a QGL branch for convenience sake to date.

Edit: The initial source code is now being committed to <http://svn.astrumfutura.org/zendframework/trunk/library/Zend/Service/>, so you can export the entire trunk tree to see how it all fits together. Just be aware only the main service class is remotely finished and awaiting its unit testing.

Posted by Pádraic Brady in PHP General, Zend Framework at 23:17

Yadis: Service Discovery for Identities like OpenID (Part 2)

[Continuation of Part-1...](#)

Service Descriptions - The Yadis Resource Descriptor (XRD)

After all this fuss and running about, it's easy to miss the point of Yadis - getting that final Yadis XRD document that describes the Services associated with a given ID. Sticking with our OpenID example, we know that in order to start authenticating a user on an OpenID Server, we need a few bits of information:

1. The OpenID Server URL; the URL to which HTTP requests will be made.
2. The OpenID Delegate: The ID of the current user on the OpenID Server (not their alias!)
3. The OpenID Service Types offered: For OpenID, this can include Signon 1.0, Signon 1.1 and Simple Registration (sreg) 1.0.

Here's an example XRD document a website might fetch when performing Service Discovery:

```
<?xml version="1.0" encoding="UTF-8"?>
<xrds:XRDS
  xmlns:xrds="xri://$xrds"
  xmlns:openid="http://openid.net/xmlns/1.0"
  xmlns="xri://$xrd*($v*2.0)">
  <XRD>
  <Service priority="0">
```

```
<Type>http://openid.net/signon/1.0</Type>
<Type>http://openid.net/signon/1.1</Type>
<Type>http://openid.net/sreg/1.0</Type>
<URI>http://www.example.com/server</URI>
<openid:Delegate>http://username.example.com/</openid:Delegate>
</Service>
</XRD>
</xrd:XRDS>
```

I'll skip the details of parsing XML - you can use PHP dom or SimpleXML in PHP5. The XML document follows the XRD format, included in a current OASIS specification. At it's most basic, the XRD document (noting these requirements are for Yadis 1.0 only) must contain a single XRD element composed of one or more Service elements. Each Service element defines a Service, as detailed in its child Type elements.

The Type element must always contain a valid URL or XRI (don't worry yet about the XRI bit - it's a proposed addition compatible with the current URI and IRI specifications). The URL should point to a Service specification, and contain a Version. If you check above, our example OpenID Provider is offering Signon 1.0-1.1 and Simple Registration 1.0.

The URI element must contain a valid URL - this is the URL the services described by the Type values are provided from, i.e. it's where a website supporting OpenID logins would send its association and/or authentication requests.

Finally, there is an optional element "openid:Delegate". This namespaced element contains the OpenID URL a user's OpenID Provider knows them as (remember, any other URL can be an alias but the OpenID Provider does not know or care about such aliases).

Conclusion

At the end of this Yadis introduction, I'll refer anyone who's stuck with it to the official Yadis Specification 1.0. It's not a huge document, but has a few nuances I've likely skipped mentioning.
<http://yadis.org/papers/yadis-v1.0.pdf>

Next up I'll jump into the details of OpenID!

Posted by Pádraic Brady in PHP General at 02:55

Yadis: Service Discovery for Identities like OpenID (Part 1)

Yesterday, I posted about my planned work on writing a PHP5 OpenID library. As I mentioned towards the end, I would be blogging about specific topics required for such a library. This first entry is an examination of Yadis, a Specification which is relied upon by OpenID. Of course Yadis is a topic of its own, since it is also used by Light-Weight Identity (LID), sXip Identity, and mIDm. You can bet more will join in once Yadis becomes more firmly established.

This post is mainly to introduce Yadis, and what my upcoming effort towards a `Zend_Service_Yadis` would entail performing. Part 2, later, will briefly examine the format of a Yadis XRD document and how it should follow the Yadis Specification 1.0.

So what is Yadis?

The purpose of Yadis is to specify a standard for enabling any party ("the Relying Party") to obtain an XML document containing relevant details of a specific Service ("the Yadis Resource Descriptor") which describes Service details such as Type, Url, and other optional data to be included in Service requests. If that's mumbo jumbo, it's easily explained with an example.

Let's take OpenID. In OpenID, the Yadis ID is simply your OpenID URL. If you have a Livejournal Blog, for example, you can use your blog URL as an Identity for other websites which support OpenID authentication. These will fire off a request to your Identity Provider to check you out. The problem is that the website doesn't know where to send the request - it needs to discover your Provider's authentication service, a process called "Service Discovery". Service Discovery is what Yadis is all about since it defines a standard HTTP GET process, and a standard Service data format (XML XRDS syntax) anyone can use.

Service Discovery

The Yadis Specification 1.0 describes all the steps a Relying Party needs to perform in order to obtain a description of the services associated with an ID. In our example, an OpenID URL will be associated with an OpenID Server URL which can be used by any website to authenticate a user.. Service Discovery describes the steps in finding out where that OpenID Server is located, i.e. its URL. In case you're wondering, an OpenID URL could be subdomain on your OpenID Provider's website, or even your own personal website URL - this is very different from the Service URL.

Performing discovery relies on finding a Yadis Resource Descriptor which details the Services associated with any Yadis ID. This document will be XML, using the Extensible Resource Descriptor (XRD) format (an OASIS specification). It's a bit tricky to find the end YRD XRDS document, so bear with me as I describe the process.

The first step is to make a simple HTTP GET request to the ID (if a URL). For example, our OpenID URL. The response now needs to be examined. There are broadly three types of valid responses detailed in the specification:

1. An HTML document with a "head" element that includes a "meta" element with a `http-equiv` attribute of "X-XRDS-Location". The "content" attribute of this element should contain a URL pointing to the Yadis ID's associated Yadis XRD Document. For example:


```
<meta http-equiv="X-XRDS-Location" content="http://youraccount.example.com/xrds" />
```

2. Any document, so long as the response-headers contain an "X-XRDS-Location" response-header whose value should contain a URL pointing to the Yadis ID's associated Yadis XRD Document.

3. A document of MIME media type, "application/xrds+xml". We'll hit gold when we receive this one! It means we just received the XRD document we were looking for. The MIME type will typically be contained in the "Content-Type" response-header.

The Bread Crumb Trail

In the above, two of the valid responses to an initial HTTP GET request to a Yadis ID (the OpenID URL in our example) don't provide the needed Yadis XRD document, but provide a means of figuring out its actual location. Typically Service Discovery under Yadis will require at least two HTTP GET requests - the first to find the XRD location, the second to actually fetch it.

The first valid response is often used when an OpenID URL is an alias. You can actually create any number of aliases (and hence any number of aliasing OpenID URLs) based on a personal URL using this method since the aliases can simply inform a website (via either a response-header or "meta" element) where to look for the underlying Service Provider's XRD document. Of course the underlying XRD is specific to only one account - that URL or ID which your aliases are each aliasing. Last time I mention "alias" in case I get dizzy... 


In short, the Yadis Specification ensures any Relying Party can follow the valid responses from one to another until they finally receive the required Yadis document. Of course, if someone throws a spanner in the works you should ensure an application doesn't follow a circular trail until PHP bumps its max execution limit. The Yadis Specification is a bit vague, but in general if you make more than 4 HTTP requests then I'd suggest the Yadis process should be considered to have failed.

[Part 2 continues by examining the Yadis XRD Document format...](#)

Posted by Pádraic Brady in PHP General, Zend Framework at 23:34

PHP Magazine: PHP Game Poll

http://www.php-mag.net/magphpde/magphpde_news/psecom.id.26905.nodeid.5.html

The International PHP Magazine has started a new poll asking readers to vote for their favourite PHP game. Judging by the list it looks like the top 10 Sourceforge registered open source games. I listed these a while back and noted phpDiplomacy in particular was doing really well. Somehow though I doubt the voting will be brisk. Should I tell them the QGL is not actually a game but a PHP game-oriented library? 

Posted by Pádraic Brady in PHP Game Development at 16:46

OpenID library for the Zend Framework?

A few months ago I was requested to add OpenID support (a growing trend) to a small website. Some of the members of the society whose website this is, were running blogs and Flickr sites and thought it would be "neat". Being a fan of "neat" myself, I figured it would be reasonable to add, and indeed it was. I was using the JanRain PHP-Openid library seeing as it is likely the only full implementation in PHP so far.


Here in the present, everyone I work with is all too aware of OpenID. It's been growing up, especially with the current 1.2 set of proposals, and of course support for OpenID is bound to be spurred higher by recent support announcements by [Mozilla for Firefox 3](#), and [Microsoft for Windows Vista](#). Also, the open Yadis specification is useful outside of OpenID for other things.

Loving PHP5 and being a long time OOP fanboy, I've decided to take on a small personal project - writing a strictly PHP5 library to support Yadis and OpenID respectively. Partly to learn the teeny details surrounding OpenID/Yadis, and also to add a pure PHP5 library which is strictly object oriented. Of course such an

undertaking is a complete pain in the ass since OpenID requires a lot of functionality from cryptographic algorithms (HMAC/Diffie Hellman), big integer math (bcmath/gmp/big_int support) to simple things like making HTTP requests from PHP.

Enter the Zend Framework. The ZF has become central to no few of my projects. I also use other frameworks, but I just find the ZF hits the sweet spot a lot. Since the Zend Framework has a lot of independent components, relying on it lets me take advantage of components like Zend_Service_Abstract, Zend_Uri and Zend_Http_Client among others. If you've ever looked under the hood of the JanRain PHP-Openid library you'll notice they implement **all** of this without a single dependency on PEAR or other external classes. I don't want to repeat that practice here.

I guess I can be accused of "Not Invented Here" syndrome. I'm willing to plead guilty on that. In my defence, there exists only the one (AFAIK) php library dedicated to OpenID. It's written in PHP4 and mixes procedural functions into it's otherwise OOP approach. I found it unnecessarily complex and duplicative - a lot of its functionality is readily available from PEAR or other sources. But outside my personal impressions, I love tinkering with and learning the nitty-gritty of new ideas. I do hope the library will prove acceptable enough to eventually be added to the Zend Framework. Regardless of that suggestion I intend plundering the ZF for all its worth, and release everything under the New BSD license anyway.

In the meantime, as a bit of a brain dump exercise I'll be posting a lot more about the technologies underlying OpenID. It's a well documented area, but I doubt everyone reading this blog has the time to read through all the associated specifications and their recent proposal updates. Should be a fun journey - for me anyway 

Posted by Pádraic Brady in PHP General, PHP Security, Zend Framework at 06:48

Quantum_Db: The QGL Data Access Object Implementation

Back in 2005, the Quantum Star SE project caught the Object Relational Mapping (ORM) bug and I took a first stab at implementing something that was simple, fast and easy to use. Until this stage I was a regular user of the Data Mapper pattern (among others) but since switching to PHP and developing open source projects found it to be largely unnecessary. The root problem is akin to hammering nails with a sledgehammer - too much complexity being implemented for a simple underwhelming task.

Here in 2007, with the QGL in development I found myself re-assessing my previous experiences and finding it attractive to develop something less complex and hopefully faster than a full scale ORM solution. Taking some previous work I did, I opened a branch in the QGL subversion repository and set to work. The results are slowly coming to fruition - the DAO, Row and part of the Driver families are shaping up going by the unit tests.

So what is Quantum_Db (in practice)? It will be a family of classes which form a Data Access Object (DAO) Framework when complete. This post is largely a ramble about current progress and intentions - so if looking for a complete solution you're in the wrong place as the current code is strictly in-progress. For reference though, the source code is under the New BSD License. It's incomplete, but available from its subversion branch at:

http://quantumstar.svn.sourceforge.net/viewvc/quantumstar/branches/Quantum_Db/ and the forums can be found for the QGL in general at <http://forums.astrumfutura.com/viewforum.php?f=25>.

In this specific implementation of Quantum_Db, the DAO class (Quantum_Db_Access) is a Singleton which attempts to offer relatively basic CRUD methods capable of use on any valid database table record. A large part of its responsibility is generating CRUD type SQL. So you could have a User record, which could be saved by passing it to a DAO save() method. It's not quite as clear cut, since to reduce the mass of code a DAO implementation often needs the "record" is encapsulated in a container called Quantum_Db_Row. This Row class can represent a single table record, but also doubles as a data container for setting up SQL conditions (think of WHERE, IN and INSERT clauses).

One of the goals intentionally targets simplicity, so there is no Propel-style Criteria object to mess about with. There will be in the future, but for simple needs, a simple DAO framework is sufficient as an initial start. A simplistic use case is adding a new User record. Some code showing this task:

```
require_once 'Models/User.php';  
// Fields are public properties handled by __get/__set in the  
// Quantum_Db_Row class. The User class is actually a simple  
// subclass defining valid fields and primary key(s).  
//  
// We do not specify an Id value for new Users. The DAO class will  
// assign this assuming Id is an auto-incrementing value for MySQL  
// or a Sequence for PostgreSQL.  
$user = new User;  
$user->name = 'Pádraic';  
$user->email = 'padraic.brady@example.com';  
$user->password = sha1('password');  
$user->country = 'ie';  
// Call on DAO to save this record to the database. Obviously any  
// database errors will result in Exceptions thrown by the DAO so  
// the database table rules (primary key, unique key, etc.) all apply.  
$user->save();
```

The first thing you can note is that the DAO class (called Quantum_Db_Access is not directly called. Instead all Row objects carry a reference to the Quantum_Db_Access singleton so Quantum_Db_Row::save() proxies to Quantum_Db_Access::save() - we only need one generic DAO object. If one desperately needed the DAO they could just place a call to Quantum_Db_Access::getInstance().

```
$dao = Quantum_Db_Access::getInstance();  
$dao->save($user);
```

In fact, you can equally access the underlying database extension driver to run queries directly using the following. Of course Singletons have their own evil uses, so ideally you'd pass around the object reference instead of littering source code with static getInstance() calls which explicitly refer to the class name.

```
// Example of accessing the driver to drop the User table  
$driver = Quantum_Db_Access::getInstance()->getDriver()->exec("DROP TABLE user");
```

Retrieving records is more interesting. It's relatively simple to extract a record based on a primary key:


```
$user = new User;  
$user->getByPk(1);  
var_dump($user->asArray()); // echo array of User (Id=1) data
```

It's much more complex to fetch rows of a very specific nature. Even worse, without a Criteria object using the MySQL IN clause is a challenge. However for those where the SQL values depend solely on simple field values a neat solution is "named queries". A Named Query (check the excellent introduction at: <http://www.tonybibbs.com/article.php/PropelDAO> regarding something similar for Propel) is an SQL string containing value placeholders (e.g. for use in a PDO::prepare()) stored in some central file. This has a few benefits and disadvantages.


The main benefit is the centralisation - like OOP, centralising logic (in this case non-generated SQL strings) reduces duplication and encourages reuse. Allowing the named queries to be indexed (for example, within an XML hierarchy) by a name and model allows them to be easily located and prepared by the Data Access Object. The disadvantages are of course that it's still limited. For example, use of a MySQL "SELECT id, name, email FROM user_table WHERE id IN(1,2,3,4)" isn't likely a storable named query since the IN() range of id values is too variable. But for many queries it's a good fit in the absence of some form of Criteria class.

Enough of the DAO class however. After beating it to death, there are still pieces to be completed when it comes to allowing complex SQL clauses. The main point is that for simple use cases it is a good match - and simple use cases are extremely common in many of my open source endeavors.

To backup the Data Access Class Quantum_Db also implements a Driver and Results class family. Again simplicity is key, so it's being written under the presumption that PHP5's PDO extension is available. The Quantum_Db_Driver_Interface file actually mirrors the PDO interface for this reason. Of course we can't forget mysqli (the most popular PHP5 database extension) or pgsql and the rest. Quantum_Db_Driver defines an interface to enable additional extension type support. Personally I only intend adding a light abstraction over ADOdb-Lite - it's not worth duplicating Mark's already existing library.

The Results class, the one part no code exists for yet - don't be worried I develop using TDD and unit testing so it's perfectly normal 


. Quantum_Db_Results will aggregate the results of queries into a class implementing PHP5's SPL Iterator. To be specific, the Iterator methods will be tied to each underlying Driver's fetch* methods. This should allow continued use of Lazy Loading during iterations, and allow continued use of foreach() etc.

Here ends my ramble for now. For those working on the QGL or similar code maybe it will provide a few of my thoughts on the topic - assuming you can summarise the rambling of course 

Posted by Pádraic Brady in Astrum Futura, PHP Game Development, PHP General at 22:25


OpenID support for PHP openssl extension? Yes, please.

I almost missed the post by Wez Furlong over on <http://netevil.org/node.php?nid=949> regarding his patch for openssl to make it easier to implement OpenID support in PHP applications.

I spent some time not long ago playing around with the JanRain library for PHP (ported from its Python big brother I think - wasn't paying much attention to its history 

) and wished I could more easily mock up a slimmer version to offer as an alternative in something like the new Zend Framework [Zend Auth](#) component without using a particularly specialised or broad based library (JanRain's or other) for the purpose.

Having OpenID support bundled with openssl is something I look forward to seeing in an openssl extension release near me especially since one of the items that seems to cause problems in some varying implementations is the big number math - something a widely installed extension supporting them (like openssl!) will help rectify.

Wez's post also comes with code examples of the patch in use which demonstrate the patched openssl at work. The patch also offers support for Typekey - its not an OpenID monopoly out there 

Posted by Pádraic Brady in PHP General at 00:58

Thursday, February 8, 2007

lamsure Blogs about a new project

lamsure appears to found a sense of the dramatic. So [his new blog entry](#) very successfully piques my interest!

Most of all, the end result should be a fun, entertaining, and groundbreaking game. Its massive, it will scale, it will be internet based, and it will be both similar and totally different from other games I've worked on and spoken about.

It's going to be a commercial project I gather so no open source goodies to examine, but these days I'm often less interested in the code, then the approach to execution. I guess once you've been using PHP since 2000 you eventually find that your reading leans heavily to theory and the experiences of other developers facing a unique challenge (whether PHP, Ruby or other). I for one look forward to reading more!

Posted by Pádraic Brady in PHP Game Development at 19:16

Friday, February 2, 2007

QGL under the New BSD License

The Quantum Game Library ("QGL") is a PHP5 library aimed at creating a bunch of reusable classes for use in PHP games. We're still in early days but with two dedicated developers, lots of feedback and suggestions we've already put together a few starting solid components and have others in progress.


Over the last few months the QGL has evolved from a minor offshoot of the Astrum Futura project into a fully independent library. I guess everyone on the team saw a generic game library as something really useful; we're all game developers on varying projects.

To reflect the library status, as well as to widen the net for potential users and developers we have decided to move from the GNU General Public License (which rules Astrum Futura) to the far more liberal [New BSD License](#). We've also provisionally determined that copyrights should be centralised to a degree, this currently is not in operation since the number of contributors remains manageable but may be (pending contributor feedback obviously) in the future.

Given licensing is now decided, file header updates in progress (if gradual), and some useful components are hitting stable, it's likely we'll make a public preview release in the near future. Given our focus on doing things right, this will only occur once documentation is completed and our unit tests verified for good measure.

As a summary (feel free to request details on the [QGL Forum Section](#)) the current near-stable component list includes:

- Quantum_Db
- Quantum_Coordinate
- Ai_Pathfinding
- Quantum_Map_Measure
- Quantum_Map_Grid
- Quantum_Turing

This is far from complete, and there are literally dozens of possible additions. Those in planning are already shaping up to offer some cool second iteration functionality. That's not even including the planned ext/qgl PHP extension... 

Posted by Pádraic Brady in Astrum Futura, PHP Game Development at 18:47