Zend_PubSubHubbub - Padraic Brady

<ac:macro ac:name="unmigrated-inline-wiki-markup"><ac:plain-text-body><![CDATA[

<ac:macro ac:name="unmigrated-inline-wiki-markup"><ac:plain-text-body><![CDATA[

Zend Framework: Zend_PubSubHubbub Component Proposal

Proposed Component Name	Zend_PubSubHubbub
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/Zend_PubSubHubbub
Proposers	Pádraic Brady
Revision	0.9 - 07 August 2009 (wiki revision: 5)

Table of Contents

1. Overview
What is PubSubHubbub?
PubSubHubbub Specification Status
Technical Details
Implementation Details within Zend Framework
2. References
3. Component Requirements, Constraints, and Acceptance Criteria
4. Dependencies on Other Framework Components
5. Theory of Operation
6. Milestones / Tasks
7. Class Index
8. Use Cases
9. Class Skeletons

1. Overview

What is PubSubHubbub?

PubSubHubbub is a Publisher-Subscriber protocol which operates over HTTP using REST (via webhook callbacks). It was originally designed by Brett Slatkin and Brad Fitzpatrick from Google. Its purpose is to allow consumers of RSS and Atom feeds to subscribe for updates to a Hub Server for any specific feed (RSS or Atom). This Hub Server will usually be notified by the publisher whenever an update in the feed occurs. In turn, the Hub Server will then notify all subscribers that an update has occurred by making a request to a URI provided by the consumer with an Atom/RSS feed containing the updated or new feed entries in the request body. This protocol therefore implements a push system whereby updates are pushed to the subscribers in real time. This is opposed to the current prevailing practice where subscribers must themselves frequently poll a publisher's feed URI to check whether any changes have occurred. This imposes a time delay between a publisher making a change and any subscriber noticing and retrieving it. The benefits of the push model therefore obviously lie in the immediacy with which subscribers can distribute new content to their users or readers.

PubSubHubbub Specification Status

The PubSubHubbub protocol is not a finalised one. Indeed it is continually evolving to add new features or polish old ones, and this proposal takes that evolution into account using as its basis the current PubSubHubbub Core 0.1 – Working Draft and its reference implementation as used by Google. As with a number of open specifications in recent years, waiting for a final and stable specification has not prevented the continued rise of implementations and real world use cases. To date, PubSubHubbub has been implemented (publicly) by SuperFeedr, FeedBurner, Rabbithub and Google Reader (support is being rolled out gradually) so the protocol is in use right now. There is little justification in holding up adoption when adoptions are already present and PubSubHubbub services are being offered to the public. The specification will evolve over

many months, but this is to be expected. Indeed, this reflects experiences with specifications like OAuth which saw widespread adoption and use in the wild long before a final specification was published. Granted, the risk of not becoming a mainstream protocol remains but the progress to date is encouraging.

Technical Details

It's actually quite difficult to make PubSubHubbub sound technically complex. It's a simple protocol using a web-hooks system (i.e. allows you to subscribe to another site's events using a URI), a coordinating Hub Server (which can be anyone since this is an open decentralised system - you could be your own hub for all PubSubHubbub cares), and the exchange of simple Atom formatted notifications or messages between all parties. Atom is used here as the notification system's substrate which should be distinguished from the feed resources whose updates are of interest and which can be either Atom or RSS.

The remaining technical details concern the protocol's operation over HTTP (nothing special), the discovery of the various URI endpoints (e.g. how consumers locate the Hub(s) servicing a feed so they can subcribe for updates using that Hub's endpoint URI), and security considerations (for future specs/extensions) such as accessing private feeds and ensuring the integrity of messages being exchanged by two parties. Anyone with a passing familiarity of OAuth or OpenID would not be out of their depth since these other open protocols deal with similar topics.

Implementation Details within Zend Framework

The Zend Framework as of version 1.10+ should (assuming successful reviews) package within its stable release an assortment of components which were designed to support a vastly simplified means of reading and writing RSS and Atom feeds, the latter targeting Atom as a general web service substrate. Also included should be the Zend_Crypt component introduced in support of Zend_Oauth. With these three components, implementing PubSubHubbub should be fairly simple though they are only strictly needed for a Hub Server implementation supporting the HMAC extension.

The only real work is in orchestrating the protocol workflow from the perspective of different parties. For example, we can assume there would be a Zend_PubSubHubbub_HubServer class to implement a Hub Server. In a similar vein, we can also assume the presence of Zend_PubSubHubbub_Subscriber and Zend_PubSubHubbub_Publisher for the remaining two parties in the overall protocol. Adding a storage medium would be supported via a Zend_Pubsubhubbub_StorageInterface interface which would allow Storage backends linked to an instance of Zend_Db_Abstract or Zend_Cache.

2. References

Github zfproposals repository including a stable Publisher and Subscriber, and an in-progress Hub Server implementation
http://github.com/padraic/zfproposals/tree/master
PubSubHubbub Website http://code.google.com/p/pubsubhubbub/
PubSubHubbub Core 0.1 – Working Draft http://pubsubhubbub.googlecode.com/svn/trunk/pubsubhubbub-core-0.1.html
PSH Page: "Why Polling Sucks" http://code.google.com/p/pubsubhubbub/wiki/WhyPollingSucks

3. Component Requirements, Constraints, and Acceptance Criteria

- MUST implement PubSubHubbub Core 0.1 – Working Draft
- MUST implement any confirmed modifications to the protocol awaiting specification
- MUST implement any confirmed Extensions to the protocol awaiting specification
- MUST implement support for Subscribers, Publishers and Hub servers
- MUST maintain specific unit tests over HTTP operations

4. Dependencies on Other Framework Components

Zend_Http_Client
Zend_Crypt (potential)
Zend_Feed_Reader
Zend_Feed_Writer (proposed elsewhere)
Zend_Exception

5. Theory of Operation

6. Milestones / Tasks

- Milestone 1: Publish and have proposal approved with Zend comments to resolve
- Milestone 2: [DONE] Complete initial component with unit tests
- Milestone 3: [DONE] Complete/Maintain testing compliance suite
- Milestone 4: [DONE] Verify that code operates under PHP 5.3/6.0-dev
- Milestone 5: Complete documentation

7. Class Index

- Zend_PubSubHubbub
- Zend_PubSubHubbub_Exception
- Zend_PubSubHubbub_Publisher
- Zend_PubSubHubbub_Subscriber
- Zend_PubSubHubbub_Subscriber_Callback
- Zend_PubSubHubbub_HubServer
- Zend_PubSubHubbub_HubServer_Callback
- Zend_PubSubHubbub_StorageInterface
- Zend_PubSubHubbub_Storage_Filesystem (used primarily in testing)
- Zend_PubSubHubbub_Storage_Memory
- Zend_PubSubHubbub_Notification

Others may be determined during development

8. Use Cases

Issuing a Publisher notification to its configured Hub Servers about updates to its Atom and RSS feeds. Note that Hubbub refers to all feeds URLs (regardless of type) as a "Topic".

```
$feeds = array(
    'http://www.example.org/rss',
    'http://www.example.org/atom'
);
$hubs = array(
    'http://pubsubhubbub.googleapps.com',
    'http://hub.example.com' // self administered backup Hub
);
$publisher = new Zend_Pubsubhubbub_Publisher;
$publisher->setUpdatedTopicUrls($feeds);
$publisher->addHubUrls($hubs);
$publisher->notifyAll();
```

Subscribing with a Hub Server to an Atom feed belonging to a Publisher.

```
$topicsOfInterest = array(
    'http://www.example.org/rss',
    'http://www.example.org/atom'
);
$topicHubs = array(
    'http://pubsubhubbub.googleapps.com',
    'http://hub.example.com'
);
$subscriber = new Zend_Pubsubhubbub_Subscriber;
$subscriber->setCallbackUrl('http://www.example.net/callback');
$subscriber->setStorage(new Zend_Pubsubhubbub_Storage_Filesystem);
$subscriber->addHubUrls($hubs);
foreach ($topicsOfInterest as $topicUrl) {
    $publisher->setTopicUrl($topicUrl);
    $publisher->subscribe();
}
```

When a Hub receives a subscription request, it fires off either a synchronous or asynchronous confirmation request back to the subscribers callback URL.

```
$callback = new Zend_Pubsubhubbub_Subscriber_Callback;
$callback->setCallbackUrl('http://www.example.net/callback');
$callback->setStorage(new Zend_Pubsubhubbub_Storage_Filesystem);
$callback->handle();
if (!$callback->isSuccess()) {
    $errors = $callback->getErrors();
    // do something with errors like logging
}
```

Unsubscribing with a Hub Server from an Atom feed belonging to a Publisher. The confirmation by Hub makes no difference to the Subscriber Callback setup above. In fact unsubscribing is just using a difference (pretty obvious!) method.

```
$topicsOfInterest = array(
    'http://www.example.org/rss',
    'http://www.example.org/atom'
);
$topicHubs = array(
    'http://pubsubhubbub.googleapps.com',
    'http://hub.example.com'
);
$subscriber = new Zend_Pubsubhubbub_Subscriber;
$subscriber->setCallbackUrl('http://www.example.net/callback');
$subscriber->setStorage(new Zend_Pubsubhubbub_Storage_Filesystem);
$subscriber->addHubUrls($hubs);
foreach ($topicsOfInterest as $topicUrl) {
    $publisher->setTopicUrl($topicUrl);
    $publisher->unsubscribe();
}
```

Handling an incoming Hub Server notification to our Subscriber Callback carrying a feed update. Same as above with some additional checks.

```
$callback = new Zend_Pubsubhubbub_Subscriber_Callback;
$callback->setCallbackUrl('http://www.example.net/callback');
$callback->setStorage(new Zend_Pubsubhubbub_Storage_Filesystem);
$callback->handle();
if (!$callback->isSuccess()) {
    $errors = $callback->getErrors();
    // do something with errors like logging
} elseif ($callback->hasFeedUpdate()) {
    $feed = Zend_Feed_Reader::importString($callback->getFeedUpdate());
    // do something with the feed, perhaps update our database for new content/changes
}
```

9. Class Skeletons

TODO

Note: The component remains in development and these skeletons are based on a code snapshot on 07 August 2009. Full available on Github: http://github.com/padraic/zfproposals/tree/master
]]></ac:plain-text-body></ac:macro>
]]></ac:plain-text-body></ac:macro>