# Zend_Feed_Reader - Pádraic Brady & Jurriën Stutterheim

<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_Feed_Reader Component Proposal

| | |
|---|---|
| **Proposed Component Name** | Zend_Feed_Reader |
| **Developer Notes** | http://framework.zend.com/wiki/display/ZFDEV/Zend_Feed_Reader |
| **Proposers** | Pádraic Brady<br>Jurriën Stutterheim<br>Alexander Veremyev (Zend Liaison) |
| **Revision** | 1.0.0 - 31 July 2008 (wiki revision: 11) |

## Table of Contents



## 1. Overview

Zend_Feed was originally created to offer a natural API akin to SimpleXML to the data contained in RSS and Atom feeds. The primary mechanism driving this is an abstract API to a DOMDocument representation of the XML feed.

Except for several exceptions, Zend_Feed does not attempt to interpret RSS or Atom. It does not understand the formats, has no knowledge of various RSS and Atom versions, makes no attempt to use best practice HTTP retrieval methods, does not validate or filter typed return values (with some exceptions) and does not currently support an internal cache for infrequently modified feeds.

Aside from the above desirable features which are not currently present, Zend_Feed also presents an inconsistent API where method calls, property calls, and return types are so unpredictable that it is a severe impediment to the natural API that leads to a lot of trial and error programming. The simplest examples are easy to spot. If you take an entry from an RSS feed (as $entry) supporting <content:encoded>, <dc:creator>, and <slash:comments> elements the values of each (i.e. a typed result, not a DOMNode object) are retrieved using the following:

- $entry->content(); (method name based on XML namespace prefix of element)
- $entry->creator(); (method name based on tagname omitting XML namespace prefix)
- $entry->'slash:comments'; (property name based on dynamic namespaced tagname)

Not only is this inconsistent, but they are not even equivelant. Attempting the third option on <dc:creator> returns an object instead of the expected typed value, for example. While this would without comparison appear to be a valid interpretation of RSS in some aspects, it comes at the cost of breaking the otherwise valid API and only works when such elements are actually present (of which is there is absolutely no guarantee).

If you look even deeper you also realise another subtle problem. Zend_Feed is incapable of correctly handling XML namespaces automatically. If two feeds use identical XML namespaces but different namespace prefixes (entirely legal XML behaviour), you would need to manually register/unregister the changing XML namespace prefixes for both respectively in order to have consistent parsing. This is particularly highlighted by any RSS 1.0 feeds (which are RDF) where Zend_Feed pre-registers an invalid RSS namespace (there is no RSS XML namespace except for RDF feeds) making parsing all but impossible.

All of the above contribute the one job any programmer wishes to avoid - writing additional support code. When using Zend_Feed every programmer will quickly require a good deal of extra work before it becomes truly useful on anything other than a few feeds whose condition is well known. It also means Zend_Feed will remain incapable of parsing some more exotic or older feeds which require specific knowledge.

## So what is Zend_Feed_Reader?

After working with Zend_Feed for a while, I've built up some helper classes across various projects to make it more useful. In one current project (something akin to the Planetarium open source project) I've decided to refactor the existing source code and formalise it as a single component or, at the discretion of the current Zend_Feed maintainers, integrate it as a general refactoring and expansion of the existing Zend_Feed classes.

Zend_Feed_Reader would be an interpretive and possibly sanitising layer for Zend_Feed which is capable of understanding all RSS and Atom versions and condensing the data across all versions and extensions of both formats into a single Accessor API. I call it interpretive, because it understands RSS and Atom sufficiently to distinguish between versions, and accurately condense the data by querying for preferred data points. Sanitising refers to the possible inclusion of a validation layer (RSS/Atom data is common enough that it could be made internal using a filter chain) since RSS/Atom is, of course, userland data we must validate before use. Though this is of a lesser priority in this proposal and likely not a first version feature.

To offer a simple example. Assume you are aggregating two RSS feeds. The first is RSS 2.0 with a <content:encoded> element, and the second is an RSS 1.0 feed (RDF) with only an RSS namespaced <description> element. Using Zend_Feed at present you would need to search for any potential content elements one by one, and then decide which one is prefered. With Zend_Feed_Reader, which interprets all feeds and selects a preferred one internally, you could simply call Zend_Feed_Reader_Entry_Rss::getContent() which would return the text from <content:encoded> for the RSS 2.0 feed and from <description> for the RSS 1.0 feed without a content element. No fuss; no custom interpretive code. To boot, Zend_Feed_Reader can (if you need it - not unusual!) tell you what version of RSS is being interpreted since it's version aware.

That is probably the major benefit. One single predictable accessor API.

Possible Features In Brief:

- Understand and interpret all versions of RSS and Atom
- Automatically condense RSS/Atom data into a single preferred Accessor API
- Automatically apply sanitisation on input (with the exception of HTML)
- Allow opt-out use of a predetermined filter/validation chain
- Allow optional setting of custom filters and validators
- Support HTTP Conditional GET - Zend_Feed and HTTP Conditional GET
- Support optional internal caching of all feeds until updated at source
- Support current Zend_Feed querying via proxy
- Expose the underlying parent DOMDocument object more clearly

Please note Zend_Feed_Reader, by definition, makes no changes to any aspect of creating feeds with Zend_Feed.

## 2. References

## 3. Component Requirements, Constraints, and Acceptance Criteria

- MUST understand and interpret all versions of RSS and Atom
- MUST automatically condense RSS/Atom data into a single preferred Accessor API
- MIGHT automatically apply sanitisation on input (with the exception of HTML)
- MIGHT allow opt-out use of a predetermined filter/validation chain
- MIGHT allow optional setting of custom filters and validators

- MUST support HTTP Conditional GET - Zend_Feed and HTTP Conditional GET
- MUST support optional internal caching of all feeds until updated at source
- MUST support current Zend_Feed querying via proxy
- MUST expose the underlying parent DOMDocument object more clearly

## 4. Dependencies on Other Framework Components

Zend_Http_Client
Zend_Feed
Zend_Feed_Abstract
Zend_Cache

## 5. Theory of Operation

Operation can follow one of two strategies. The first is to reuse the existing Zend_Feed API. While this would promote reuse, it doesn't solve the problem of the inconsistent API and would instead become an abstract proxy to the original implementation. The second is to bypass the current API, while maintaining access to it by proxy, and making use of the underlying DOMDocument object representation of a feed to query against (e.g. using XPath). At present this proposal favours the second option since it reduces dependencies and allows greater flexibility by using DOMDocument directly to drive XPath queries (at a performance cost perhaps - nothing is for free  ).

Operation would be quite simple. Zend_Feed_Reader would parent both Feed and Entry subclasses for RSS and Atom which contain sufficient logic to filter data from the underlying feeds. A user seeking the content from a batch of RSS and Atom feeds with unknown versions would only need one method: getContent(). Interpreting all feeds across all versions and locating the closest element match for an entry's content will be handled internally by Zend_Feed_Reader which can select a relevant "content" style element from the current feed when faced with many alternatives. Accessor methods would be available at both the Feed and Entry levels as appropriate. All methods would return literal values i.e. other than sanitisation no attempt would be made to manipulate values.

In addition to the unified API, Zend_Feed_Reader could be supplemented with plugin APIs for common extensions. For example, GeoRSS or Yahoo! Weather. These additional APIs could be derived from plugin calls: "$entry->georss()->getLatitude();".

If a separate component, Zend_Feed_Reader would include a local Zend_Feed_Abstract instance through composition and will encapulate access for all features intended to be internal (e.g. automated validation, caching and conditional fetches). Many of the above will require configuration options (either as an Array or Zend_Config instance).

Please refer to future use cases for additional operational examples.

## 6. Milestones / Tasks

- Milestone 1: Select a strategy of Zend_Feed_Reader or refactoring Zend_Feed
- Milestone 2: Complete initial component with unit tests
- Milestone 3: Assemble a collection of common edge cases for resolution
- Milestone 4: Verify that code operates under PHP 5.3
- Milestone 5: Complete documentation

## 7. Class Index

- Zend_Feed_Reader
- Zend_Feed_Reader_Feed
- Zend_Feed_Reader_Feed_Rss
- Zend_Feed_Reader_Feed_Atom
- Zend_Feed_Reader_Entry
- Zend_Feed_Reader_Entry_Rss
- Zend_Feed_Reader_Entry_Atom

Others may be determined during development

## 8. Use Cases

Use cases are currently being drafted. But here are a few samples. As usual, please refer to my public Zend Framework Proposals respository on Subversion which contains unit tests (as good as if not better than use cases).

**UC-01**

Detect the type and version of a feed (represented by a string of form TYPE-VERSION). The strings are also contained in string constants like Zend_Feed_Reader::TYPE_ATOM_03. Maybe I should split this across two additional methods for ease of use by end users.

```
$feed = Zend_Feed_Reader::import('http://example.com/rss093.xml');
// Zend_Feed_Reader::TYPE_RSS_093
echo $feed->getType(); // "rss-093"
```

**UC-02**

Get some feed information

```
$feed = Zend_Feed_Reader::import('http://example.com/rss093.xml');
echo $feed->getTitle();
echo $feed->getCopyright();
echo $feed->getLink();
echo $feed->getLanguage();
echo $feed->getEncoding();
// ...
```

**UC-03**

Get some entry information

```
$feed = Zend_Feed_Reader::import('http://example.com/rss093.xml');
foreach($feed as $entry) {
    echo $feed->getTitle();
    echo $feed->getAuthor();
    echo $feed->getLink();
    echo $feed->getId();
    echo $feed->getContent();
    echo $feed->getTopics();
    echo $feed->getDate();
    // expose common extensions as plugins
    echo $feed->georss()->getLatitude(); // GeoRSS-Simple
    echo $feed->slash()->getComments); // Slash
    // ...
}
```

**UC-04**

Access underlying Zend_Feed API (RSS)

```
$feed = Zend_Feed_Reader::import('http://example.com/rss093.xml');
foreach($feed as $entry) {
    echo $feed->{'slash:comments'};
    echo $feed->content();
    echo $feed->guid();
}
```

**UC-05**

Enable cache (conditional fetches also automatically enabled). Set a sensible TTL - Zend_Feed_Reader will clear the cache of old content when any feed is updated.

```
$feed = Zend_Feed_Reader::import('http://example.com/rss093.xml');
$feed->setCache($cacheAdapter);
foreach($feed as $entry) {
    echo $feed->content();
    echo $feed->getTitle();
    echo $feed->slash()->getComments();
}
```

## 9. Class Skeletons

**Zend_Feed_Reader**

```
class Zend_Feed_Reader
{

    const TYPE_RSS_090 = 'rss-090';
    const TYPE_RSS_091 = 'rss-091';
    const TYPE_RSS_091_NETSCAPE = 'rss-091n';
    const TYPE_RSS_091_USERLAND = 'rss-091u';
    const TYPE_RSS_092 = 'rss-092';
    const TYPE_RSS_093 = 'rss-093';
    const TYPE_RSS_094 = 'rss-094';
    const TYPE_RSS_10 = 'rss-10';
    const TYPE_RSS_20 = 'rss-20';
    const TYPE_RSS_ANY = 'rss';
    const TYPE_ATOM_03 = 'atom-03';
    const TYPE_ATOM_10 = 'atom-10';
    const TYPE_ATOM_ANY = 'atom';
    const TYPE_ANY = 'any';

    const NAMESPACE_RSS_090 = 'http://my.netscape.com/rdf/simple/0.9/';
    const NAMESPACE_RSS_10 = 'http://purl.org/rss/1.0/';
    const NAMESPACE_RDF = 'http://www.w3.org/1999/02/22-rdf-syntax-ns#';
    const NAMESPACE_ATOM_03 = 'http://purl.org/atom/ns#';
    const NAMESPACE_ATOM_10 = 'http://www.w3.org/2005/Atom';

    const NAMESPACE_DC_10 = 'http://purl.org/dc/elements/1.0/';
    const NAMESPACE_DC_11 = 'http://purl.org/dc/elements/1.1/';
    const NAMESPACE_CONTENT_10 = 'http://purl.org/rss/1.0/modules/content/';
    const NAMESPACE_SLASH_10 = 'http://purl.org/rss/1.0/modules/slash/';
    const NAMESPACE_WFWCOMMENTAPI = 'http://wellformedweb.org/CommentAPI/';
    const NAMESPACE_GEO = 'http://www.w3.org/2003/01/geo/wgs84_pos#';
    const NAMESPACE_YAHOOWEATHER_10 = 'http://xml.weather.yahoo.com/ns/rss/1.0';
    const NAMESPACE_ITUNES_10 = 'http://www.itunes.com/dtds/podcast-1.0.dtd';

    public static function import($url)
    {
    }

    public static function importString($string)
    {
    }

    public static function importFeed(Zend_Feed_Abstract $feed)
    {
    }

    public static function detectType(Zend_Feed_Abstract $feed)
    {
    }
}
```

**Zend_Feed_Reader_Entry_Interface (Atom, Dc and RSS Entry classes implement this)**

```
interface Zend_Feed_Reader_Entry_Interface
{
    public function __construct(Zend_Feed_Entry_Abstract $entry, $entryKey, $type =
null);

    public function setXpath(DOMXPath $xpath);

    public function getAuthors();

    public function getAuthor($index = 0);

    public function getContent();

    public function getDateCreated();

    public function getDateModified();

    public function getDescription();

    public function getId();

    public function getLink($index = 0);

    public function getLinks();

    public function getPermalink();

    public function getTitle();

    public function getType();

    public function toArray();

    public function getDomDocument();
}
```

**Zend_Feed_Reader_Feed_Interface (Atom, Dc and RSS Feed classes implements this)**

```
interface Zend_Feed_Reader_Feed_Interface
{
    public function __construct(Zend_Feed_Entry_Abstract $entry, $entryKey, $type =
null);

    public function setXpath(DOMXPath $xpath);

    public function getAuthors();

    public function getAuthor($index = 0);

    public function getContent();

    public function getDateCreated();

    public function getDateModified();

    public function getDescription();

    public function getId();

    public function getLink($index = 0);

    public function getLinks();

    public function getPermalink();

    public function getTitle();

    public function getType();

    public function toArray();

    public function getDomDocument();
}
```

**Zend_Feed_Reader_Feed_Abstract**

```
abstract class Zend_Feed_Reader_Feed_Abstract implements Iterator
{
    /**
     * Enter description here...
     *
     * @var Zend_Feed_Abstract
     */
    protected $_feed = null;

    /**
     * Enter description here...
     *
     * @var array
     */
    protected $_data = array();
```

```
    /**
     * Enter description here...
     *
     * @var DOMDocument
     */
    protected $_domDocument = null;

    /**
     * Enter description here...
     *
     * @var DOMXPath
     */
    protected $_xpath = null;

    /**
     * Enter description here...
     *
     * @var array
     */
    protected $_entries = array();

    /**
     * Enter description here...
     *
     * @var int
     */
    protected $_entriesKey = 0;

    /**
     * Enter description here...
     *
     * @param Zend_Feed_Abstract $feed
     * @param string $type
     */
    public function __construct(Zend_Feed_Abstract $feed, $type = null)
    {
    }

    /**
     * Enter description here...
     *
     * @return string
     */
    public function getType()
    {
    }

    /**
     * Enter description here...
     *
     * @return int
     */
    public function count()
    {
    }

    /**
     * Enter description here...
     *
```

```
     */
    public function rewind()
    {
    }

    /**
     * Enter description here...
     *
     * @return Zend_Feed_Reader_Entry_Interface
     */
    public function current()
    {
    }

    /**
     * Enter description here...
     *
     * @return unknown
     */
    public function key()
    {
    }

    /**
     * Enter description here...
     *
     */
    public function next()
    {
    }

    public function valid()
    {
    }

    /**
     * Enter description here...
     *
     * @return array
     */
    public function toArray() // untested
    {
        return $this->_data;
    }

    /**
     * Enter description here...
     *
     */
    abstract protected function _registerDefaultNamespaces();

    /**
     * Enter description here...
     *
     */
```

```
    abstract protected function _indexEntries();
}
```

]]></ac:plain-text-body></ac:macro>
]]></ac:plain-text-body></ac:macro>