

## 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 22:55