Zend_Gdata usability improvements - Ryan Boyd

<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_Gdata usability improvements Component Proposal

Proposed Component Name	Zend_Gdata usability improvements
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/Zend_Gdata usability improvements
Proposers	Ryan Boyd Bill Karwin assisting
Revision	1.1 - 10 April 2007: Early design notes. 1.2 - 06 May 2007: Ready for review. Code not yet in incubator. 1.3 - 07 May 2007: Code for App, Gdata and Calendar partially in incubator 1.4 - 09 May 2007: First draft of Spreadsheets code in incubator 1.6 - 12 June 2007: 1.0.0 RC2a of Zend_Gdata released with support for Spreadsheets and Calendar. See public docs for usage (as opposed to examples in this document), as the public docs are going to be more accurate. (wiki revision: 68)

Table of Contents

1. Overview

This proposal describes enhancements to the Zend_Gdata component to make it more usable and consistent with GData client libraries for other languages (e.g. Python, Java).

2. References

- Google Data APIs
- Google Calendar Data API Developer's Guide
- Google Data API Python Client Library project
- Google Data API Python Client Library design overview

3. Component Requirements, Constraints, and Acceptance Criteria

- A developer should be able to perform most functionality by looking only at the PHP documentation of classes, methods and a few examples.
- A developer should not need to know the structure of the XML documents used by the GData services nor the namespaces, etc in order to use the library effectively
- As much consistency as possible should exist between the PHP client library and other GData client libraries, while still maintaining a PHP feel to the library.
- The library should be usable by both experts and beginners in GData and PHP.

4. Dependencies on Other Framework Components

- Zend_Http

5. Theory of Operation

General Operation

- **Retrieving Feeds** is done through the creation of a `Query` and `Service` object. The query object is used to specify the query parameters using standard method calls such as `$query->setUpdatedMin('2007-01-01');`. The service object is called to retrieve the feed based upon the URL generated by the query object using a call such as `$feed = $service->getFeed($query->getQueryUrl())`. The service object uses a `Zend_Http_Client` to retrieve the feed and stores each of the entries as a `DOMElement` instances. The feed implements the Iterator interface to allow for iterating individual entries. As individual entries are iterated upon, the `DOMElements` are converted into individual instances of the appropriate `Entry` class (such as `Zend_Gdata_Entry`). These entry instances use 3 methods – `transferFromDOM()`, `takeChildFromDOM()`, `takeAttributeFromDOM()` – to store data from the DOM as protected members of the appropriate `Entry` instance. All entries and attributes that are children of the Atom `<entry>` are converted into their object equivalents. If elements are not consumed via the `takeChildFromDOM()` method, they become an instance of `Zend_App_Extension_Element`.
- **Retrieving Entries** is done in a very similiar fashion as Retrieving Feeds (as described above), except the `getEntry()` method (or appropriate equivalent) is called on the `Service` object-- and an individual `Entry` of the appropriate type is constructed rather than a `Feed`.
- **Updating Entries** is performed by retrieving the entry from the service, updating the appropriate instance members and saving changes to the entry (causing the associated service to do a `HTTP PUT` to the entry's edit URL on the server). The member elements can be modified using either the magic *`_set (such as $entry->title = $service->newTitle('entry title'); method or by callling pre-defined setters (such as $entry->setTitle($service->newTitle('entry title'));. Note the use of the $service->newTitle method to create instances of the Zend_Gdata_App_Extension_Title class. These factory methods (handled by the magic`* `_call()` method) are provided only as a conveience-- you could choose to instantiate an instance of the `Zend_Gdata_App_Extension_Title` class directly. See the use cases below for more information.
- **Creating Entries** is accomplished by creating an instance of the correct `Entry` class, setting the appropriate instance members and using an authenticated `Service` class to perform a `HTTP POST` to the feed's post URL. The entry instance's members are set using the method described above in 'Updating Entries.'
- **Deleting Entries** is accomplished either by directly passing an edit URL to an authenticated `Service` claass, or by retrieving an individual entry and calling its' `delete()` method – causing a `HTTP DELETE` request to be sent to the edit URL.

Services and queries

- Remove query parameter functionality from Zend_Gdata, Zend_Gdata_Calendar, Zend_Gdata_spreadsheets, etc and create separate query classes which are used to generate the URL used for querying. This is a more natural model as query parameters can sometimes be used for POSTs. With the current design, if a developer wanted to specify query params for a query, but not for an insert, they would need to either create 2 service objects or clear out the params between the GET and the POST. Moving to having a separate Query class eliminates the need for this and is more in line with the other Google Data APIs client libraries. All service-specific query classes should inherit from Zend_Gdata and provide only those query parameters which are specific to the service.
- Zend_Gdata classes should extend from classes providing basic Atom Publishing Protocol (APP) operations such as getEntry, getFeed, post, put, delete etc. This provides the ability for developers to use the Zend Framework to work with generic APP services offered by other companies, organizations, or sites and provide extensions to the Framework to implement any extensions to APP which are used by those services.

- Zend_Gdata should remain as a class overriding the important client operation methods to handle the GData-specific versions of Atom Publishing Protocol (APP). This includes methods such as getEntry, getFeed, post, delete, put, etc.
- Service classes, such as Zend_Gdata_Calendar, should override Zend_Gdata methods as necessary and implement additional methods such as getCalendarListEntry, getCalendarEventEntry, getCalendarListFeed, getCalendarEventFeed, updateEvent, deleteEvent, insertEvent, etc. These methods would return data model classes of the appropriate type (see data model section below).
- Authentication mechanism (using Zend_Gdata_AuthSub/Zend_Gdata_ClientLogin to return a Zend_Http_Client) should remain. Although different than other GData client libraries, this is a very useful way of specifying credentials, configuring proxies, etc. However, some changes need to be made to the authentication mechanism to improve usability - a developer should not need to know the value of the `Service` parameter used by ClientLogin. Rather, the service class (extending Zend_Gdata) should provide this information.

Inheritance tree:

- **Zend_Gdata_App** - *provides basic APP methods such as `getFeed()`, `getEntry()`, `post()`, `delete()`, `put()`*
  - **Zend_Gdata** - *provides factory `call` method for creating data model classes, allows for GData-specific implementations of Zend_Gdata_App functionality*
    - **Zend_Gdata_Calendar** - *provides methods for getting a Zend_Gdata_Calendar_EventFeed via `getCalendarEventFeed()`, a Zend_Gdata_Calendar_ListFeed via `getCalendarListFeed()`, `insertEvent()` etc.*
    - **Zend_Gdata_Spreadsheets** - *provides spreadsheets-specific functionality such as `getSpreadsheetsFeed()`, `getWorksheetsFeed()`, `updateCell()`, `insertRow()`, etc.*
    - etc.
- **Zend_Gdata_Query** - *provides methods for getting/setting the GData-wide query parameters such as `q`, `updated-min`, etc methods named like `setUpdatedMin()`. Also, provides a method to get the URL represented by the Query object.*
  - **Zend_Gdata_Calendar_EventQuery** - *provides additional query parameters that are specific to querying Calendar event feeds. This includes methods like `setFutureEvents()`, `setRecurrenceExpansionStart()`, etc.*
  - **Zend_Gdata_Calendar_ListQuery** - *provides additional query parameters that are specific to querying Calendar list (meta) feeds.*
  - **Zend_Gdata_Spreadsheets_CellQuery** - *provides additional query parameters that are specific to querying Spreadsheet Cell feeds, such as `setMinRow()`, `setMaxRow()`, etc.*
  - etc.

Data model

Inheritance tree:

- **Zend_Gdata_App_Base** - *provides functionality common to feeds, entries and all other data model classes*
  - **Zend_Gdata_App_FeedEntryParent** - *implements functionality common to feeds and entries.*
    - **Zend_Gdata_App_Feed** - *implements functionality required by APP.*
      - **Zend_Gdata_Feed** - *stub to implement additional functionality required by a GData feed.*
        - **Zend_Gdata_Calendar_EventFeed** - *stub to implement additional functionality required by a Calendar event feed.*
        - **Zend_Gdata_Calendar_ListFeed** - *stub to implement additional functionality required by a Calendar list feed.*
        - **Zend_Gdata_Spreadsheets_CellFeed** - *stub to implement additional functionality required by a Spreadsheets cell feed.*
        - ...etc
    - **Zend_Gdata_App_Entry** - *Generic implementations of methods such as `construct()`, `getDOM()`, `takeChildFromDOM()`, `takeAttributeFromDOM()`, `transferFromDOM()`, `get()`, `set()`. Also, provides accessors to generic Atom elements such as `<title>`, `<content>`, etc. Can be extended and overridden to handle specific types of entries.*
      - **Zend_Gdata_Entry** - *Placeholder for any GData-wide entry elements. Not much here at present*
        - **Zend_Gdata_Kind_EventEntry** - *Implements the Event "kind" used by GData serviecs. This includes all elements in the GData namespace (elements prefixed with gd: in the GData documentation).*
          - **Zend_Gdata_Calendar_EventEntry** - *Implements extensions to the basic Zend_Gdata_EventEntry that describe properties specifically related to the Google Calendar implementation of the Event "kind". This includes all elements in the Google Calendar namespace (elements prefixed with gCal: in the Calendar data API documentation).*
        - **Zend_Gdata_Calendar_ListEntry** - *Represents an individual Entry in a Calendar meta feed.*
        - **Zend_Gdata_Spreadsheets_CellEntry** - *Represents a Cell in a spreadsheet. Objects of this type are used when working with the Cell feed in the Google Spreadsheets data API.*
        - ...etc

6. Milestones / Tasks

Describe some intermediate state of this component in terms of design notes, additional material added to this page, and / code. Note any significant dependencies here, such as, "Milestone #3 can not be completed until feature Foo has been added to ZF component XYZ." Milestones will be required for acceptance of future proposals. They are not hard, and many times you will only need to think of the first three below.

- Milestone 1: [DONE] design notes will be published here
- Milestone 2: [IN PROGRESS] Working prototype checked into the incubator implementing the Atom Publishing Protocol(APP) and core GData components (minus batch support and media support)
- Milestone 3: [IN PROGRESS] Implementation checked into the incubator supporting utility classes for working with the Google Spreadsheets data API
- Milestone 4: [IN PROGRESS] Implementation checked into the incubator supporting utility classes for working with the Google Calendar data API
- Milestone 5: Finalized and stable implementation of the core APP and GData components to exist in the incubator, based on any feedback from the Calendar data API and Spreadsheets data API component developers.
- Milestone 6: Unit tests for APP, GData, Spreadsheets, Calendar exist, work, and are checked into SVN.
- Milestone 7: Initial documentation exists.
- Milestone 8: Changes are moved from the incubator to trunk to correspond with xxx release
- Milestone 10: Implementation checked into the incubator supporting utility classes for working with the Google Base data API
- Milestone 11: Implementation checked into the incubator supporting utility classes for working with the Google Picasa data API

7. Class Index

- Zend_Gdata
    - Zend_Gdata_Entry
    - Zend_Gdata_Feed
    - Zend_Gdata_App
    - Zend_Gdata_Query
    - Zend_Gdata_Exception
    - Zend_Gdata_BadMethodCallException
    - Zend_Gdata_AuthException
    - Zend_Gdata_HttpException
    - Zend_Gdata_InvalidArgumentException
        - Zend_Gdata_App_Entry
        - Zend_Gdata_App_Feed
        - Zend_Gdata_App_Extension
            - Zend_Gdata_App_Extension_Element
            - Zend_Gdata_App_Extension_Author
            - Zend_Gdata_App_Extension_Category
            - Zend_Gdata_App_Extension_Content
            - Zend_Gdata_App_Extension_Contributor
            - Zend_Gdata_App_Extension_Date
            - Zend_Gdata_App_Extension_Email
            - Zend_Gdata_App_Extension_Generator
            - Zend_Gdata_App_Extension_Icon
            - Zend_Gdata_App_Extension_Id
            - Zend_Gdata_App_Extension_Link
            - Zend_Gdata_App_Extension_Logo
            - Zend_Gdata_App_Extension_Name
            - Zend_Gdata_App_Extension_Person
            - Zend_Gdata_App_Extension_Published
            - Zend_Gdata_App_Extension_Right
            - Zend_Gdata_App_Extension_Source
            - Zend_Gdata_App_Extension_Subtitle
            - Zend_Gdata_App_Extension_Summary
            - Zend_Gdata_App_Extension_Text
            - Zend_Gdata_App_Extension_Title
            - Zend_Gdata_App_Extension_Updated
            - Zend_Gdata_App_Extension_Uri
    - Zend_Gdata_Extension
        - Zend_Gdata_Extension_AttendeeStatus
        - Zend_Gdata_Extension_AttendeeType
        - Zend_Gdata_Extension_Comments
        - Zend_Gdata_Extension_EventStatus
        - Zend_Gdata_Extension_ExtendedProperty
        - Zend_Gdata_Extension_OriginalEvent
        - Zend_Gdata_Extension_Recurrence
        - Zend_Gdata_Extension_RecurrenceException

        - Zend_Gdata_Extension_Reminder
        - Zend_Gdata_Extension_SendEventNotifications
        - Zend_Gdata_Extension_Transparency
        - Zend_Gdata_Extension_Visibility
        - Zend_Gdata_Extension_When
        - Zend_Gdata_Extension_Where
        - Zend_Gdata_Extension_Who
    - Zend_Gdata_Calendar
        - Zend_Gdata_Calendar_ListEntry
        - Zend_Gdata_Calendar_EventEntry
        - Zend_Gdata_Calendar_EventCommentEntry
        - Zend_Gdata_Calendar_ListFeed
        - Zend_Gdata_Calendar_EventFeed
        - Zend_Gdata_Calendar_EventCommentFeed
        - Zend_Gdata_Calendar_EventQuery
        - Zend_Gdata_Calendar_Extension
            - Zend_Gdata_Calendar_Extension_AccessLevel
            - Zend_Gdata_Calendar_Extension_Color
            - Zend_Gdata_Calendar_Extension_Hidden
            - Zend_Gdata_Calendar_Extension_Link
            - Zend_Gdata_Calendar_Extension_Selected
            - Zend_Gdata_Calendar_Extension_Timezone
            - Zend_Gdata_Calendar_Extension_WebContent
    - Zend_Gdata_Spreadsheets
        - Zend_Gdata_Spreadsheets_CellEntry
        - Zend_Gdata_Spreadsheets_ListEntry
        - Zend_Gdata_Spreadsheets_SpreadsheetEntry
        - Zend_Gdata_Spreadsheets_WorksheetEntry
        - Zend_Gdata_Spreadsheets_CellFeed
        - Zend_Gdata_Spreadsheets_ListFeed
        - Zend_Gdata_Spreadsheets_SpreadsheetFeed
        - Zend_Gdata_Spreadsheets_WorksheetFeed
        - Zend_Gdata_Spreadsheets_CellQuery
        - Zend_Gdata_Spreadsheets_ListQuery
        - Zend_Gdata_Spreadsheets_Extension
            - Zend_Gdata_Spreadsheets_Extension_ColCount
            - Zend_Gdata_Spreadsheets_Extension_RowCount
            - Zend_Gdata_Spreadsheets_Extension_Cell
            - Zend_Gdata_Spreadsheets_Extension_ListCell
- etc.

8. Use Cases

UC 1 : Unauthed Calendar Event Retrieval

```
$gdataCal = new Zend_Gdata_Calendar($client);

$eventFeed =
$gdataCal->getCalendarEventFeed('http://www.google.com/calendar/feeds/api.rboyd@google.c
$calendar (below) would be a CalendarEventEntry object
foreach ($eventFeed->entries as $event) {
    $title = $event->title->text;
    if (count($event->where) > 0 && $event->where[0]->valueString != null) {
        $location = $event->where[0]->valueString;
    }
}
```

UC 2 : Authed Calendar List Feed Retrieval

```
// Create an authenticate Zend_Http_Client object -- using Google's ClientLogin
$client = Zend_Gdata_ClientLogin::getHttpClient($user, $pass,
Zend_Gdata_Calendar::AUTH_SERVICE_NAME);
$gdataCal = new Zend_Gdata_Calendar($client);

$calFeed = $gdataCal->getCalendarListFeed();

// $calendar (below) would be a CalendarListEntry object
foreach ($calFeed->entries as $calendar) {
    $title = $calendar->getTitle()->text;
    // The color is stored as a member called $_color.  This uses the magic __get
accessor to get the Color
    $color = $calendar->color->value;
    $accessLevel = $calendar->accessLevel->value;
}
```

UC 3 : Authed Calendar Event Query

```
// Create an authenticate Zend_Http_Client object -- using Google's ClientLogin
$client = Zend_Gdata_ClientLogin::getHttpClient($user, $pass,
Zend_Gdata_Calendar::AUTH_SERVICE_NAME);
$gdataCal = new Zend_Gdata_Calendar($client);

// Create Zend_Gdata_Calendar_Query instance using factory method
$query = $gdataCal->newQuery();

// Specify the user/visibility/projection for the feed
// Optional - This defautls to /default/private/full
$query->setUser('default');
$query->setVisibility('private');
$query->setProjection('full');

// Query for all events taking place on 2007-04-04 or later
$query->setStartMin('2007-04-04');

// Send the request for the Zend_Gdata_Calendar_EventFeed using the URL generated by
the Query
$feed = $gdataCal->getCalendarEventFeed($query->getQueryUrl());
foreach ($feed->entries as $entry) {
    echo "\tTitle:" . $entry->title->text . "\n";
    if (count($entry->where) > 0 && $entry->where[0]->valueString != null) {
        echo "\t\tWhere:" . $entry->where[0]->valueString . "\n";
    }
}
```

UC 4 : Creating a Calendar Event

```
// Create an authenticate Zend_Http_Client object -- using Google's ClientLogin
$client = Zend_Gdata_ClientLogin::getHttpClient($user, $pass,
Zend_Gdata_Calendar::AUTH_SERVICE_NAME);
$gdataCal = new Zend_Gdata_Calendar($client);
$event = $gdataCal->newEventEntry();
$event->title = $gdataCal->newTitle('Party');
$event->where = array($gdataCal->newWhere('Down by the river'));

//option 1 - use APP POST method to POST raw XML data.  Response is a
{{Zend_Http_Response}} object
$httpResponse = $gdataCal->post($event->saveXML());

//option 2 - use APP POST method to POST a {{Zend_Gdata_App_Entry}} object.  Response
is a {{Zend_Http_Response}} object
$httpResponse = $gdataCal->post($event);

//option 3 - use Calendar insertEvent method to POST a
{{Zend_Gdata_Calendar_EventEntry}} object.  Response is a
{{Zend_Gdata_Calendar_EventEntry}}.
$returnedEntry = $gdataCal->insertEvent($event);
print "Title of inserted event: " . $returnedEntry->title->text . "\n";
```

UC 5 : Updating a Calendar Event

```
// Create an authenticate Zend_Http_Client object -- using Google's ClientLogin
$client = Zend_Gdata_ClientLogin::getHttpClient($user, $pass,
Zend_Gdata_Calendar::AUTH_SERVICE_NAME);
$gdataCal = new Zend_Gdata_Calendar($client);

// Create and insert a new Zend_Gdata_Calendar_EventEntry
$event = $gdataCal->newEventEntry();
$event->title = $gdataCal->newTitle('Party');
$event->where = array($gdataCal->newWhere('Down by the river'));
$originalEvent = $gdataCal->insertEvent($event);

// Change the title of the inserted entry to 'Updated Party'
print "Title of inserted event: " . $originalEvent->title->text . "\n";
$originalEvent->title->text = 'Updated Party';

// save() will send a HTTP PUT to the server with the updated entry
// The entry, as it exists on the server after the update, will be saved in a
//     Zend_Gdata_Calendar_EventEntry called $updatedEvent
$updatedEvent = $originalEvent->save();

print "New title of inserted event: " . $updatedEvent->title->text . "\n";
```

UC 6 : Deleting a Calendar Event

```
// Create an authenticate Zend_Http_Client object -- using Google's ClientLogin
$client = Zend_Gdata_ClientLogin::getHttpClient($user, $pass,
Zend_Gdata_Calendar::AUTH_SERVICE_NAME);
$gdataCal = new Zend_Gdata_Calendar($client);

// Create and insert a new Zend_Gdata_Calendar_EventEntry
$event = $gdataCal->newEventEntry();
$event->title = $gdataCal->newTitle('Party');
$event->where = array($gdataCal->newWhere('Down by the river'));
$originalEvent = $gdataCal->insertEvent($event);

// delete() will send a HTTP DELETE to the server
$originalEvent->delete();
```

9. Class Skeletons

SK 1: Zend_Gdata_Calendar_EventEntry

```
class Zend_Gdata_Calendar_EventEntry extends Zend_Gdata_Kind_EventEntry
{

    /**
     * @var string The name of this class- used to determine type of
     * object to return from $entry->save() calls
     */
    protected $_entryClassName = 'Zend_Gdata_Calendar_EventEntry';

    /**
     * @var Zend_Gdata_Extension_SendEventNotifications The GData element
     * which indicates whether notifications should be sent when events are
     * created/updated/deleted.
     */
    protected $_sendEventNotifications = null;

    /**
     * @var Zend_Gdata_Extension_ExtendedProperty An array of arbitrary
     * name-value pairs.  All elements which can occur multiple times
     * are stored as an array
     */
    protected $_extendedProperty = array();


    /**
     * Returns a DOMElement representing the state of this element and all children
     *
     * @param DOMDocument $doc The DOMDocument used to construct DOMElements
     * @return DOMElement
     */
    public function getDOM($doc)
    {
        // 1. Recursively calls parent::getDOM($doc)
```

```
        //    a) $doc is created if it is null
        //    b) Creates root element representing this object
        // 2. Adds children to the root based upon the member variables
        // 3. The getDOM method of each known member variable (such as _title) is
called,
        //    and the result is appended
        // 4. Attributes are added as necessary
    }

    /**
     * Takes an individual child node and determines how it should be handled
     * If Node has no handler defined, it is created as a
     * Zend_Gdata_App_Extension_Element
     *
     * @param DOMNode
     */
    public function takeChildFromDOM($child)
    {
        // 1. Switch statement checks against the node name to determine how to
        //    process the node.  If the node name matches:
        //        a. Creates a new extension element of the appropriate type
        //        b. Calls the extension element's transferFromDOM method
        //        c. Sets the value of the member variable to be equal to the new
        //           extension element, or, if the member variable is an array,
        //           appends the extension element to that array.
        // 2. If there is no match based upon the name of the node passed in,
        //    the parent::takeChildFromDOM($child) is called.  Eventually, if
        //    there are no matches anywhere in the inheritance tree,
        //    a new Zend_Gdata_App_Extension_Element is created and the element
        //    is added to the _extensionElements array.
    }

    /**
     * Takes an attribute node and determines how it should be handled
     * If attribute has no handler defined, it is stored as an array
     * (namespaceURI,name,value) in the _extensionAttributes array
     *
     * @param DOMNode
     */
    public function takeAttributeFromDOM($attribute)
    {
        // 1. Switch statement checks against the attribute name to determine how
        //    to process the node.  If the node name matches:
        //        a. Sets the value of the attribute into the appropriate member
        // 2. If there is no match based upon the name of the attribute passed in,
        //    the parent::takeAttributeFromDOM($attribute) is called.  Eventually, if
        //    there are no matches anywhere in the inheritance tree,
        //    a new array element with name,value is created and inserted
        //    into the _extensionAttributes array
    }

    /**
     * Takes a DOMNode representing the current element and calls takeChildFromDOM()
     * for each child and takeAttributeFromDOM() for each attribute
     *
     * @param DOMNode
     */
    public function transferFromDOM($node)
    {
```

```
        foreach ($node->childNodes as $child) {
            $this->takeChildFromDOM($child);
        }
        foreach ($node->attributes as $attribute) {
            $this->takeAttributeFromDOM($attribute);
        }
    }

    /**
     * Retrieves the DOM represented by this entry and returns the XML representation
     * of the DOM.
     *
     * NOTE: THIS METHOD IS ACTUALLY IN AN ANCESTOR - Zend_Gdata_App_Base, but is
     * documented here for illustration purposes
     *
     * @returns string - XML representation of this entry
     */
    public function saveXML()
    {
        $element = $this->getDOM() to get the DOM representation
        return $element->ownerDocument->saveXML($element);
    }

    /**
     * Saves any changes made in this entry back to the server by doing a
     * HTTP PUT to the edit URL defined in the entry.
     *
     * NOTE: THIS METHOD IS ACTUALLY IN AN ANCESTOR - Zend_Gdata_App_Base, but is
     * documented here for illustration purposes
     *
     * @returns mixed - entry of the appropriate type
     * @throws Zend_Gdata_HttpException - thrown for any error codes returned by
server
     * @throws Zend_Gdata_Exception - thrown when an edit link cannot be found in the
entry
     */
    public function save()
    {
        // 1. Grabs edit link using $this->getLink('link');
        // 2. Grabs Zend_Http_Client object from Zend_Gdata_App::getHttpClient
        // 3. PUTs or POSTs (with 'X-HTTP-Method-Override' header) to the edit URL,
depending
        //    on value of Zend_Gdata_App::getHttpMethodOverride
        // 4. Creates a new Zend_Gdata_App_Entry or instance of a subclass (as
appropriate) to
        //    represent the response from the server.
    }

    /**
     * Deletes the object on the server represented by this entry by
     * sending a HTTP DELETE to the edit URL
     *
     * NOTE: THIS METHOD IS ACTUALLY IN AN ANCESTOR - Zend_Gdata_App_Base, but is
     * documented here for illustration purposes
     *
     * @throws Zend_Gdata_HttpException - thrown for any error codes returned by
server
     * @throws Zend_Gdata_Exception - thrown when an edit link cannot be found in the
entry
```

```
     */
    public function delete()
    {
        // 1. Grabs edit link using $this->getLink('link');
        // 2. Grabs Zend_Http_Client object from Zend_Gdata_App::getHttpClient
        // 3. DELETEs or POSTs (with 'X-HTTP-Method-Override' header) to the edit URL,
depending
        //    on value of Zend_Gdata_App::getHttpMethodOverride
    }

    /**
     * Zend_Gdata_App_Entry-specific version of the __get magic method
     * Calls the appropriate accessor method for the member which the code is
attempting
     * to access.
     *
     * NOTE: THIS METHOD IS ACTUALLY IN AN ANCESTOR - Zend_Gdata_App_Base, but is
     * documented here for illustration purposes
     *
     * @param string    $name - the member attempting to be found
     * @returns mixed   - the value attempting to be found
     * @throws Zend_Gdata_NoSuchMethodException
     */
    public function __get($name)
    {
        // 1. Calls the appropriate accessor method
        // 2. If no accessor is defined, throws a Zend_Gdata_NoSuchMethodException
        // eg calling $this->title should attempt to call $this->getTitle() and throw
a
        //    Zend_Gdata_NoSuchMethodException if getTitle cannot be found.
    }

    /**
     * Zend_Gdata_App_Entry-specific version of the __set magic method
     * Calls the appropriate setter method for the member which the code is attempting
     * to set.
     *
     * NOTE: THIS METHOD IS ACTUALLY IN AN ANCESTOR - Zend_Gdata_App_Base, but is
     * documented here for illustration purposes
     *
     * @param string    $name  - the member attempting to be found
     * @param mixed     $value - the value to set the member to
     * @returns mixed          - the value of the set member
     * @throws Zend_Gdata_NoSuchMethodException
     */
    public function __set($name, $value)
    {
        // 1. Calls the appropriate setter method
        // 2. If no setter is defined, throws a Zend_Gdata_NoSuchMethodException
        // eg calling $this->title = 'foo' should attempt to call
$this->setTitle('foo')
    }

    /**
     * Gets the _extendedProperty member
     *
     * @returns array Array of Zend_Gdata_Extension_ExtendedProperty elements
     */
    public function getExtendedProperty()
```

```
{
    return $this->_extendedProperty;
}

/**
 * Sets the _extendedProperty member
 *
 * @returns Zend_Gdata_Kind_EventEntry Provides a fluent interface
 */
public function setExtendedProperty($value)
{
    $this->_extendedProperty = $value;
    return $this;
```

```
    }
}
```

SK 2: Zend_Gdata_App_Feed

```
class Zend_Gdata_App_Feed extends Zend_Gdata_App_FeedEntryParent implements Iterator
{
    /**
     * The classname for individual feed elements.
     *
     * @var string
     */
    protected $_entryClassName = 'Zend_Gdata_App_Entry';

    /**
     * The element name for individual feed elements (Atom <entry>
     * elements).
     *
     * @var string
     */
    protected $_entryElementName = 'entry';

    /**
     * Cache of feed entries.
     *
     * @var array
     */
    protected $_entry;

    /**
     * Index representing current position in $_entries.  Used
     * by Iterator implementation
     *
     * @var int
     */
    protected $_entryIndex;

    /**
     * Creates individual Entry objects of the appropriate type and
     * stores them in the $_entry array based upon DOM data.
     *
     * @param DOMNode $child The DOMNode to process
     */
    protected function takeChildFromDOM($child)
    {
        $absoluteNodeName = $child->namespaceURI . ':' . $child->localName;
        switch ($absoluteNodeName) {
        case Zend_Gdata_Data::lookupNamespace('atom') . ':' . 'entry':
            $newEntry = new $this->_entryClassName(
                null,
                $child);
            $newEntry->setHttpClient($this->getHttpClient());
            $this->_entry[Zend_Font - Karol Babioch] = $newEntry;
            break;
        default:
```

```
            parent::takeChildFromDOM($child);
            break;
        }
    }

    /**
     * Make accessing some individual elements of the feed easier.
     *
     * Special accessors 'entry' and 'entries' are provided so that if
     * you wish to iterate over an Atom feed's entries, you can do so
     * using foreach ($feed->entries as $entry) or foreach
     * ($feed->entry as $entry).
     *
     * @param  string $var The property to access.
     * @return mixed
     */
    public function __get($var)
    {
        switch ($var) {
            case 'entries':
                return $this;
        }
    }

    /**
     * Returns Entry elements from this feed
     *
     * @return array Zend_Feed_App_Entry array
     */
    public function getEntry()
    {
        return $this->_entry;
    }


    /**
     * Get the number of entries in this feed object.
     *
     * @return integer Entry count.
     */
    public function count()
    {
        return count($this->_entry);
    }


    /**
     * Required by the Iterator interface.
     *
     * @return void
     */
    public function rewind()
    {
        $this->_entryIndex = 0;
    }

    /**
     * Required by the Iterator interface.
     *
```

```
     * @return mixed The current row, or null if no rows.
     */
    public function current()
    {
        return new $this->_entryClassName(
            null,
            $this->_entry[$this->_entryIndex]);
    }

    /**
     * Required by the Iterator interface.
     *
     * @return mixed The current row number (starts at 0), or NULL if no rows
     */
    public function key()
    {
        return $this->_entryIndex;
    }


    /**
     * Required by the Iterator interface.
     *
     * @return mixed The next row, or null if no more rows.
     */
    public function next()
    {
        ++$this->_entryIndex;
    }


    /**
     * Required by the Iterator interface.
     *
     * @return boolean Whether the iteration is valid
     */
    public function valid()
    {
        return 0 <= $this->_entryIndex && $this->_entryIndex < $this->count();
```

```
    }
}
```

SK 3..infinity: Zend_Gdata_*

```
/**
 * Refer to prototype in incubator for lots of other code skeleton/implementation
examples
 */
```

10. Testing Strategy

Unit tests will be created for both offline and online operation. The offline unit tests will test the basic data model functionality – parsing XML, converting it to the appropriate class instances and members, etc. These tests should be run and pass as a dependency of running the online "unit" tests. These tests *may* use mock XML data. The online "unit" tests will explicitly test the service components (HTTP, etc) in addition to implicitly testing the data model.

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