python-wordpress-xmlrpc Documentation

Release 2.2

Max Cutler

May 18, 2014

Contents

Python library to interface with a WordPress blog's XML-RPC API.

An implementation of the standard WordPress API methods is provided, but the library is designed for easy integration with custom XML-RPC API methods provided by plugins.

A set of *classes* are provided that wrap the standard WordPress data types (e.g., Blog, Post, User). The provided *method implementations* return these objects when possible.

Note: The XML-RPC API is disabled in WordPress by default. To enable, go to Settings->Writing->Remote Publishing and check the box for XML-RPC.

Warning: python-wordpress-xmlrpc 2.0+ is not fully backwards-compatible with 1.x versions of the library.

CHAPTER 1

Getting Started

1.1 Overview

1.1.1 Installation

1. Verify you meet the following requirements:
    - WordPress 3.4+ **OR** WordPress 3.0-3.3 with the XML-RPC Modernization Plugin.
    - Python 2.6+ **OR** Python 3.x
2. Install from PyPI using `easy_install python-wordpress-xmlrpc` or `pip install python-wordpress-xmlrpc`.

1.1.2 Quick Start

Create an instance of the `Client` class with the URL of the WordPress XML-RPC endpoint and user credentials. Then pass an `XmlrpcMethod` object into its `call` method to execute the remote call and return the result.

```
>>> from wordpress_xmlrpc import Client, WordPressPost
>>> from wordpress_xmlrpc.methods.posts import GetPosts, NewPost
>>> from wordpress_xmlrpc.methods.users import GetUserInfo

>>> wp = Client('http://mysite.wordpress.com/xmlrpc.php', 'username', 'password')
>>> wp.call(GetPosts())
[<WordPressPost: hello-world (id=1)>]

>>> wp.call(GetUserInfo())
<WordPressUser: max>

>>> post = WordPressPost()
>>> post.title = 'My new title'
>>> post.content = 'This is the body of my new post.'
>>> post.terms_names = {
>>>   'post_tag': ['test', 'firstpost'],
>>>   'category': ['Introductions', 'Tests']
>>> }
>>> wp.call(NewPost(post))
5
```

Notice that properties of `WordPress` objects are accessed directly, and not through the `definition` attribute defined in the source code.

When a `WordPress` object is used as a method parameter, its `struct` parameter is automatically extracted for consumption by XML-RPC. However, if you use an object in a list or other embedded data structure used as a parameter, be sure to use `obj.struct` or else WordPress will not receive data in the format it expects.

Custom XML-RPC Methods

To interface with a non-standard XML-RPC method (such as one added by a plugin), you must simply extend `wordpress_xmlrpc.XmlrpcMethod` or one of its subclasses (`AnonymousMethod` or `AuthenticatedMethod`).

The `XmlrpcMethod` class provides a number of properties which you can override to modify the behavior of the method call.

Sample class to call a custom method added by a ficticious plugin:

```
from wordpress_xmlrpc import AuthenticatedMethod

class MyCustomMethod(AuthenticatedMethod):
        method_name = 'custom.MyMethod'
        method_args = ('arg1', 'arg2')
```

See *Custom XML-RPC Methods* for more details.

1.2 Examples

1.2.1 Working with Posts

python-wordpress-xmlrpc supports all registered WordPress post types.

Behind the scenes in WordPress, all post types are based on a single “post” database table, and all of the functionality is exposed through the posts methods in the XML-RPC API.

For consistency, the same approach is adopted by python-wordpress-xmlrpc.

Note: Posts will be sent as drafts by default. If you want to publish a post, set *post.post_status = ‘publish’*.

Normal Posts

First, let’s see how to retrieve normal WordPress posts:

```
from wordpress_xmlrpc import Client
from wordpress_xmlrpc.methods import posts

client = Client(...)
posts = client.call(posts.GetPosts())
# posts == [WordPressPost, WordPressPost, ...]
```

And here’s how to create and edit a new post:

```
from wordpress_xmlrpc import WordPressPost

post = WordPressPost()
post.title = 'My post'
post.content = 'This is a wonderful blog post about XML-RPC.'
post.id = client.call(posts.NewPost(post))

# whoops, I forgot to publish it!
post.post_status = 'publish'
client.call(posts.EditPost(post.id, post))
```

Pages

Out of the box, WordPress supports a post type called "page" for static non-blog pages on a WordPress site. Let's see how to do the same actions for pages:

```
from wordpress_xmlrpc import WordPressPage

pages = client.call(posts.GetPosts({'post_type': 'page'}, results_class=WordPressPage))
# pages == [WordPressPage, WordPressPage, ...]
```

Note two important differences:

1. The `filter` parameter's `post_type` option is used to limit the query to objects of the desired post type.
2. The constructor was passd a `results_class` keyword argument that told it what class to use to interpret the response values.

And here's how to create and edit a page:

```
page = WordPressPage()
page.title = 'About Me'
page.content = 'I am an aspiring WordPress and Python developer.'
page.post_status = 'publish'
page.id = client.call(posts.NewPost(page))

# no longer aspiring
page.content = 'I am a WordPress and Python developer.'
client.call(posts.EditPost(page))
```

Custom Post Types

While the pages example used its own `results_class`, that was a unique situation because pages are special in WordPress and have fields directly in the posts table.

Most custom post types instead use post custom fields to store their additional information, and custom fields are already exposed on `WordPressPost`.

For this example, let's assume that your plugin or theme has added an `acme_product` custom post type to WordPress:

```
# first, let's find some products
products = client.call(posts.GetPosts({'post_type': 'acme_product', 'number': 100}))

# calculate the average price of these 100 widgets
sum = 0
for product in products:
        # note: product is a WordPressPost object
        for custom_field in product.custom_fields:
                if custom_field['key'] == 'price':
                        sum = sum + custom_field['value']
                        break
```

```
average = sum / len(products)

# now let's create a new product
widget = WordPressPost()
widget.post_type = 'acme_product'
widget.title = 'Widget'
widget.content = 'This is the widget's description.'
widget.custom_fields = []
widget.custom_fields.append({
        'key': 'price',
        'value': 2
})
widget.id = client.call(posts.NewPost(widget))
```

Advanced Querying

By default, `wordpress_xmlrpc.methods.posts.GetPosts` returns 10 posts in reverse-chronological order (based on their publish date). However, using the `filter` parameter, posts can be queried in other ways.

Result Paging

If you want to iterate through all posts in a WordPress blog, a server-friendly technique is to use result paging using the `number` and `offset` options:

```
# get pages in batches of 20
offset = 0
increment = 20
while True:
        posts = client.call(posts.GetPosts({'number': increment, 'offset': offset}))
        if len(posts) == 0:
                break  # no more posts returned
        for post in posts:
                do_something(post)
        offset = offset + increment
```

Ordering

If you don't want posts sorted by `post_date`, then you can use `orderby` and `order` options to change that behavior.

For example, in sync scenarios you might want to look for posts by modification date instead of publish date:

```
recently_modified = client.call(posts.GetPosts({'orderby': 'post_modified', 'number': 100}))
```

Or if you want your ACME products sorted alphabetically:

```
products = client.call(posts.GetPosts({'post_type': 'acme_product', 'orderby': 'title', 'order': 'ASC
```

Post Status

Another common scenario is that you only want published posts:

```
published_posts = client.call(posts.GetPosts({'post_status': 'publish'}))
```

Or only draft posts:

```
draft_posts = client.call(posts.GetPosts({'post_status': 'draft'}))
```

You can find the set of valid `post_status` by using the `wordpress_xmlrpc.methods.posts.GetPostStatusList` method.

1.2.2 Working with Taxonomies

Taxonomies in WordPress are a means of classifying content. Out of the box, WordPress has two primary taxonomies, categories (`category`) and tags (`post_tag`). Plugins and themes can specify additional custom taxonomies.

Taxonomies

To retrieve a list of taxonomies for a WordPress blog, use `wordpress_xmlrpc.methods.taxonomies.GetTaxonomies`:

```
from wordpress_xmlrpc import Client
from wordpress_xmlrpc.methods import taxonomies

client = Client(...)
taxes = client.call(taxonomies.GetTaxonomies())
# taxes == [WordPressTaxonomy, WordPressTaxonomy, ...]
```

An individual taxonomy can be retrieved by name:

```
category_tax = client.call(taxonomies.GetTaxonomy('category'))
```

Note: Taxonomies can only be created and modified within WordPress using hooks in plugins or themes. The XML-RPC API permits only reading of taxonomy metadata.

Terms

Terms are the individual entries in a taxonomy.

For example, to retrieve all blog categories:

```
categories = client.call(taxonomies.GetTerms('category'))
```

And to create a new tag:

```
from wordpress_xmlrpc import WordPressTerm

tag = WordPressTerm()
tag.taxonomies = 'post_tag'
tag.name = 'My New Tag'
tag.id = client.call(taxonomies.NewTerm(tag))
```

Or to create a child category:

```
parent_cat = client.call(taxonomies.GetTerm('category', 3))

child_cat = WordPressTerm()
child_cat.taxonomy = 'category'
child_cat.parent = parent_cat.id
child_cat.name = 'My Child Category'
child_cat.id = client.call(taxonomies.NewTerm(child_cat))
```

Terms and Posts

Terms are of little use on their own, they must actually be assigned to posts.

If you already have `WordPressTerm` objects, use `terms` property of `WordPressPost`:

```
tags = client.call(taxonomies.GetTerms('post_tag', {...}))

post = WordPressPost()
post.title = 'Post with Tags'
post.content = '...'
post.terms = tags
post.id = client.call(posts.NewPost(post))
```

If you want to add a category to an existing post:

```
category = client.call(taxonomies.GetTerm('category', 3))
post = client.call(posts.GetPost(5))

post.terms.append(category)
client.call(posts.EditPost(post.id, post))
```

But what if you have not yet retrieved the terms or want to create new terms? For that, you can use the `terms_names` property of `WordPressPost`:

```
post = WordPressPost()
post.title = 'Post with new tags'
post.content = '...'
post.terms_names = {
        'post_tag': ['tagA', 'another tag'],
        'category': ['My Child Category'],
}
post.id = client.call(posts.NewPost(post))
```

Note that `terms_names` is a dictionary with taxonomy names as keys and list of strings as values. WordPress will look for existing terms with these names or else create new ones. Be careful with hierarchical taxonomies like `category` because of potential name ambiguities (multiple terms can have the same name if they have different parents); if WordPress detects ambiguity, it will throw an error and ask that you use `terms` instead with a proper `WordPressTerm`.

Advanced Querying

By Count

To find the 20 most-used tags:

```
tags = client.call(taxonomies.GetTerms('post_tag', {'number': 20, 'orderby': 'count', 'order': 'DESC'

for tag in tags:
        print tag.name, tag.count
```

Searching/Autocomplete

To perform case-insensitive searching against term names, use the `search` option for `filter`:

```
user_input = 'wor'  # e.g., from UI textbox
tags = client.call(taxonomies.GetTerms('post_tag', {'search': user_input, 'orderby': 'count', 'numbe

suggestions = [tag.name for tag in tags]
# suggestions == ['word', 'WordPress', 'world']
```

1.2.3 Working with Media

Uploading a File

The `wordpress_xmlrpc.methods.media.UploadFile` method can be used to upload new files to a WordPress blog:

```
from wordpress_xmlrpc import Client, WordPressPost
from wordpress_xmlrpc.compat import xmlrpc_client
from wordpress_xmlrpc.methods import media, posts

client = Client(...)

# set to the path to your file
filename = '/path/to/my/picture.jpg'

# prepare metadata
data = {
        'name': 'picture.jpg',
        'type': 'image/jpg',  # mimetype
}

# read the binary file and let the XMLRPC library encode it into base64
with open(filename, 'rb') as img:
        data['bits'] = xmlrpc_client.Binary(img.read())

response = client.call(media.UploadFile(data))
# response == {
#       'id': 6,
#       'file': 'picture.jpg'
#       'url': 'http://www.example.com/wp-content/uploads/2012/04/16/picture.jpg',
#       'type': 'image/jpg',
# }
attachment_id = response['id']
```

This newly-uploaded attachment can then be set as the thumbnail for a post:

```
post = WordPressPost()
post.title = 'Picture of the Day'
post.content = 'What a lovely picture today!'
post.post_status = 'publish'
post.thumbnail = attachment_id
post.id = client.call(posts.NewPost(post))
```

Note: If you do not know the mimetype at development time, you can use the `mimetypes` library in Python:

```
data['type'] = mimetypes.read_mime_types(filename) or mimetypes.guess_type(filename)[0]
```

Querying

Use `wordpress_xmlrpc.methods.media.GetMediaLibrary` and `wordpress_xmlrpc.methods.media.GetMedia`
to retrieve information about attachments.

1.2.4 Custom XML-RPC Methods

See the WordPress Codex for details on how to write a WordPress plugin that adds custom XML-RPC method to WordPress.

The following examples will use the sample methods from that codex page.

Anonymous Methods

To use the `mynamespace.subtractTwoNumbers` method, create a class derived from `wordpress_xmlrpc.AnonymousMethod`:

```
from wordpress_xmlrpc import AnonymousMethod

class SubtractTwoNumbers(AnonymousMethod):
        method_name = 'mynamespace.subtractTwoNumbers'
        method_args = ('number1', 'number2')
```

This class can then be used with `Client.call()`:

```
from wordpress_xmlrpc import Client

client = Client('http://www.example.com/xmlrpc.php', 'harrietsmith', 'mypassword')
difference = client.call(SubtractTwoNumbers(10, 5))
# difference == 5
```

Authenticated Methods

If your custom authenticated method follows the common `method(blog_id, username, password, *args)` structure, then you can use `wordpress_xmlrpc.AuthenticatedMethod`:

```
from wordpress_xmlrpc import AuthenticatedMethod

class GetUserID(AuthenticatedMethod):
        method_name = 'mynamespace.getUserID'
```

Again, this class can then be used with `Client.call()`:

```
user_id = client.call(GetUserID())
# user_id == 3
```

Note that you do not have to supply `blog_id`, `username`, or `password` to the class constructor, since these are automatically added by *AuthenticatedMethod*. Custom method classes only require arguments specified by `method_args` and the optional `optional_args`.

CHAPTER 2

Reference

2.1 Client

The `Client` class is the gateway to your WordPress blog's XML-RPC interface.

Once initialized with your blog URL and user credentials, the client object is ready to execute XML-RPC methods against your WordPress blog using its `Client.call()` method.

2.1.1 Client

class **`Client`** (*url*, *username*, *password*[, *blog_id*])

Parameters

- **url** – URL of the blog's XML-RPC endpoint (e.g., http://www.example.com/xmlrpc.php)
- **username** – Username of a valid user account on the WordPress blog
- **password** – The password for this user account

`call` (*method*)

Parameters **method** – `wordpress_xmlrpc.XmlrpcMethod`-derived class

2.1.2 XML-RPC Method Classes

Library to interface with the WordPress XML-RPC API.

See README for usage instructions.

class `wordpress_xmlrpc.`**`XmlrpcMethod`**

Base class for XML-RPC methods.

Child classes can override methods and properties to customize behavior:

Properties:

- *method_name*: XML-RPC method name (e.g., 'wp.getUserInfo')
- *method_args*: Tuple of method-specific required parameters
- *optional_args*: Tuple of method-specific optional parameters
- *results_class*: Python class which will convert an XML-RPC response dict into an object

default_args (*client*)
Builds set of method-non-specific arguments.

get_args (*client*)
Builds final set of XML-RPC method arguments based on the method's arguments, any default arguments, and their defined respective ordering.

process_result (*raw_result*)
Performs actions on the raw result from the XML-RPC response.

If a *results_class* is defined, the response will be converted into one or more object instances of that class.

class `wordpress_xmlrpc.`**AnonymousMethod**
An XML-RPC method for which no authentication is required.

class `wordpress_xmlrpc.`**AuthenticatedMethod**
An XML-RPC method for which user authentication is required.

Blog ID, username and password details will be passed from the *Client* instance to the method call.

2.2 WordPress Objects

2.2.1 WordPressPost

class WordPressPost
Represents a post, page, or other registered custom post type in WordPress.

- id
- user
- date (*datetime*)
- date_modified (*datetime*)
- slug
- post_status
- title
- content
- excerpt
- link
- comment_status
- ping_status
- terms (*list* of `WordPressTerms`)
- terms_names (*dict*)
- custom_fields (*dict*)
- enclosure (*dict*)
- password
- post_format
- thumbnail

- sticky
- post_type

WordPressPage

class WordPressPage

Derived from `WordPressPost`, represents a WordPress page. Additional fields:

- template
- parent_id
- parent_title
- order (*int*)
- post_type = 'page'

2.2.2 WordPressPostType

class WordPressPostType

Metadata for registered WordPress post type.

- name
- label
- labels (*dict*)
- cap (*dict*)
- hierarchical
- menu_icon
- menu_position
- public
- show_in_menu
- taxonomies (*list*)
- is_builtin
- supports (*list*)

2.2.3 WordPressTaxonomy

class WordPressTaxonomy

Metadata for registered WordPress taxonomy.

- name
- label
- labels (*dict*)
- hierarchical
- public

- show_ui
- cap (*dict*)
- is_builtin
- object_type (*list*)

2.2.4 WordPressTerm

class `WordPressTerm`

Represents a term (e.g., tag or category) in a WordPress taxonomy.

- id
- group
- taxonomy
- taxonomy_id
- name
- slug
- description
- parent
- count (*int*)

2.2.5 WordPressBlog

class `WordPressBlog`

Represents a WordPress blog/site.

- id
- name
- url
- xmlrpc
- is_admin (*bool*)

2.2.6 WordPressAuthor

class `WordPressAuthor`

Minimal representation of a WordPress post author.

- id
- user_login
- display_name

2.2.7 WordPressUser

class **`WordPressUser`**
Basic representation of a WordPress user.

- id
- username
- password
- roles
- nickname
- url
- first_name
- last_name
- registered
- bio
- email
- nicename
- display_name

2.2.8 WordPressComment

class **`WordPressComment`**
Represents a WordPress comment.

- id
- user
- post
- post_title
- parent
- date_created (*datetime*)
- status
- content
- link
- author
- author_url
- author_email
- author_ip

2.2.9 WordPressMedia

class **`WordPressMedia`**

Represents a WordPress post media attachment.

- id
- parent
- title
- description
- caption
- date_created (*datetime*)
- link
- thumbnail
- metadata

2.2.10 WordPressOption

class **`WordPressOption`**

Represents a WordPress blog setting/option.

- name
- description
- value
- read_only (*bool*)

2.3 Methods

See *Examples* for guidance on how to use the following method classes.

2.3.1 methods.posts

class `wordpress_xmlrpc.methods.posts.`**`GetPosts`**`(`[*filter*, *fields*]`)`

Retrieve posts from the blog.

Parameters:

***filter*: optional *dict* of filters:**

- *number*
- *offset*
- *orderby*
- *order*: 'ASC' or 'DESC'
- *post_type*: Defaults to 'post'
- *post_status*

Returns: *list* of `WordPressPost` instances.

class `wordpress_xmlrpc.methods.posts.`**`GetPost`**(*post_id*[, *fields*])
Retrieve an individual blog post.

Parameters: *post_id*: ID of the blog post to retrieve.

Returns: `WordPressPost` instance.

class `wordpress_xmlrpc.methods.posts.`**`NewPost`**(*content*)
Create a new post on the blog.

Parameters: *content*: A `WordPressPost` instance with at least the *title* and *content* values set.

Returns: ID of the newly-created blog post (an integer).

class `wordpress_xmlrpc.methods.posts.`**`EditPost`**(*post_id*, *content*)
Edit an existing blog post.

Parameters: *post_id*: ID of the blog post to edit. *content*: A `WordPressPost` instance with the new values for the blog post.

Returns: *True* on successful edit.

class `wordpress_xmlrpc.methods.posts.`**`DeletePost`**(*post_id*)
Delete a blog post.

Parameters: *post_id*: ID of the blog post to delete.

Returns: *True* on successful deletion.

class `wordpress_xmlrpc.methods.posts.`**`GetPostStatusList`**
Retrieve the set of possible blog post statuses (e.g., "draft," "private," "publish").

Parameters: None

Returns: *dict* of values and their pretty names.

Example:

```
>>> client.call(GetPostStatusList())
{'draft': 'Draft', 'private': 'Private', 'pending': 'Pending Review', 'publish': 'Published'
```

class `wordpress_xmlrpc.methods.posts.`**`GetPostFormats`**
Retrieve the set of post formats used by the blog.

Parameters: None

Returns: *dict* containing a *dict* of all blog post formats (*all*) and a list of formats *supported* by the theme.

Example:

```
>>> client.call(GetPostFormats())
{'all': {'status': 'Status', 'quote': 'Quote', 'image': 'Image', 'aside': 'Aside', 'standard
 'supported': ['aside', 'link', 'gallery', 'status', 'quote', 'image']}
```

class `wordpress_xmlrpc.methods.posts.`**`GetPostTypes`**
Retrieve a list of post types used by the blog.

Parameters: None

Returns: *dict* with names as keys and `WordPressPostType` instances as values.

class `wordpress_xmlrpc.methods.posts.`**`GetPostType`**(*post_type*)
Retrieve an individual blog post type.

Parameters: *post_type*: Name of the blog post type to retrieve.

Returns: `WordPressPostType` instance.

2.3.2 methods.pages

class `wordpress_xmlrpc.methods.pages.`**`GetPageStatusList`**
Retrieve the set of possible blog page statuses (e.g., "draft," "private," "publish").

Parameters: None

Returns: *dict* of values and their pretty names.

Example:

```
>>> client.call(GetPageStatusList())
{'draft': 'Draft', 'private': 'Private', 'publish': 'Published'}
```

class `wordpress_xmlrpc.methods.pages.`**`GetPageTemplates`**
Retrieve the list of blog templates.

Parameters: None

Returns: *dict* of values and their paths.

Example:

```
>>> client.call(GetPageTemplates())
{'Default': 'default', 'Sidebar Template': 'sidebar-page.php', 'Showcase Template': 'showcas
```

2.3.3 methods.taxonomies

class `wordpress_xmlrpc.methods.taxonomies.`**`GetTaxonomies`**
Retrieve the list of available taxonomies for the blog.

Parameters: None

Returns: *list* of `WordPressTaxonomy` instances.

class `wordpress_xmlrpc.methods.taxonomies.`**`GetTaxonomy`**(*taxonomy*)
Retrieve an individual taxonomy.

Parameters: *taxonomy*: name of the taxonomy

Returns: `WordPressTaxonomy` instance.

class `wordpress_xmlrpc.methods.taxonomies.`**`GetTerms`**(*taxonomy*[, *filter*])
Retrieve the list of available terms for a taxonomy.

Parameters: *taxonomy*: name of the taxonomy

***filter*: optional *dict* of filters:**

- *number*
- *offset*
- *orderby*
- *order*: 'ASC' or 'DESC'
- *hide_empty*: Whether to return terms with count==0
- *search*: Case-insensitive search on term names

Returns: *list* of `WordPressTerm` instances.

class `wordpress_xmlrpc.methods.taxonomies.`**`GetTerm`**(*taxonomy*, *term_id*)

Retrieve an individual term.

Parameters: *taxonomy*: name of the taxonomy

term_id: ID of the term

Returns: `WordPressTerm` instance.

class `wordpress_xmlrpc.methods.taxonomies.`**`NewTerm`**(*term*)

Create new term.

Parameters: *term*: instance of `WordPressTerm`

Returns: ID of newly-created term (an integer).

class `wordpress_xmlrpc.methods.taxonomies.`**`EditTerm`**(*term_id*, *term*)

Edit an existing term.

Parameters: *term_id*: ID of the term to edit.

term: A `WordPressTerm` instance with the new values for the term.

Returns: *True* on successful edit.

class `wordpress_xmlrpc.methods.taxonomies.`**`DeleteTerm`**(*taxonomy*, *term_id*)

Delete a term.

Parameters: *taxonomy*: name of the taxonomy

term_id: ID of the term to delete.

Returns: *True* on successful deletion.

2.3.4 methods.comments

class `wordpress_xmlrpc.methods.comments.`**`GetComments`**(*filter*)

Gets a set of comments for a post.

Parameters:

***filter*: a *dict* with the following values:**

- *post_id*: the id of the post to retrieve comments for
- *status*: type of comments of comments to retrieve (optional, defaults to 'approve')
- *number*: number of comments to retrieve (optional, defaults to 10)
- *offset*: retrieval offset (optional, defaults to 0)

Returns: *list* of `WordPressComment` instances.

class `wordpress_xmlrpc.methods.comments.`**`GetComment`**(*comment_id*)

Retrieve an individual comment.

Parameters: *comment_id*: ID of the comment to retrieve.

Returns: `WordPressPost` instance.

class `wordpress_xmlrpc.methods.comments.`**`NewComment`**(*post_id*, *comment*)

Create a new comment on a post.

Parameters: *post_id*: The id of the post to add a comment to. *comment*: A `WordPressComment` instance with at least the *content* value set.

Returns: ID of the newly-created comment (an integer).

class `wordpress_xmlrpc.methods.comments.`**`NewAnonymousComment`**(*post_id*, *comment*)
Create a new comment on a post without authenticating.

NOTE: Requires support on the blog by setting the following filter in a plugin or theme:

> add_filter('xmlrpc_allow_anonymous_comments', '__return_true');

Parameters: *post_id*: The id of the post to add a comment to. *comment*: A `WordPressComment` instance with at least the *content* value set.

Returns: ID of the newly-created comment (an integer).

class `wordpress_xmlrpc.methods.comments.`**`EditComment`**(*comment_id*, *comment*)
Edit an existing comment.

Parameters: *comment_id*: The idea of the comment to edit. *comment*: A `WordPressComment` instance with at least the *content* value set.

Returns: *True* on successful edit.

class `wordpress_xmlrpc.methods.comments.`**`DeleteComment`**(*commend_id*)
Delete an existing comment.

Parameters: *comment_id*: The id of the comment to be deleted.

Returns: *True* on successful deletion.

class `wordpress_xmlrpc.methods.comments.`**`GetCommentStatusList`**
Retrieve the set of possible blog comment statuses (e.g., "approve," "hold," "spam").

Parameters: None

Returns: *dict* of values and their pretty names.

Example:

```
>>> client.call(GetCommentStatusList())
{'hold': 'Unapproved', 'approve': 'Approved', 'spam': 'Spam'}
```

class `wordpress_xmlrpc.methods.comments.`**`GetCommentCount`**(*post_id*)
Retrieve comment count for a specific post.

Parameters: *post_id*: The id of the post to retrieve comment count for.

Returns: *dict* of comment counts for the post divided by comment status.

Example:

```
>>> client.call(GetCommentCount(1))
{'awaiting_moderation': '2', 'total_comments': 23, 'approved': '18', 'spam': 3}
```

2.3.5 methods.users

class `wordpress_xmlrpc.methods.users.`**`GetUser`**(*user_id*[, *fields*])
Retrieve an individual user.

Parameters: *user_id*: ID of the user *fields*: (optional) *list* of fields to return. Specific fields, or groups 'basic' or 'all'.

Returns: `WordPressUser` instance.

class `wordpress_xmlrpc.methods.users.`**`GetUsers`**([*filter*, *fields*])
Retrieve list of users in the blog.

Parameters:

***filter*: optional *dict* of filters:**

- *number*
- *offset*
- *role*

fields: optional *list* of fields to return. Specific fields, or groups 'basic' or 'all'.

Returns: *list* of `WordPressUser` instances.

class `wordpress_xmlrpc.methods.users.`**`GetProfile`**

Retrieve information about the connected user.

Parameters: None

Returns: instance of `WordPressUser` representing the user whose credentials are being used with the XML-RPC API.

class `wordpress_xmlrpc.methods.users.`**`EditProfile`**`(`*user*`)`

Edit profile fields of the connected user.

Parameters: *user*: *WordPressUser* instance.

Returns: *True* on successful edit.

class `wordpress_xmlrpc.methods.users.`**`GetUsersBlogs`**

Retrieve list of blogs that this user belongs to.

Parameters: None

Returns: *list* of `WordPressBlog` instances.

class `wordpress_xmlrpc.methods.users.`**`GetAuthors`**

Retrieve list of authors in the blog.

Parameters: None

Returns: *list* of `WordPressAuthor` instances.

class `wordpress_xmlrpc.methods.users.`**`GetUsers`**`(`[*filter*, *fields*]`)`

Retrieve list of users in the blog.

Parameters:

***filter*: optional *dict* of filters:**

- *number*
- *offset*
- *role*

fields: optional *list* of fields to return. Specific fields, or groups 'basic' or 'all'.

Returns: *list* of `WordPressUser` instances.

class `wordpress_xmlrpc.methods.users.`**`GetUser`**`(`*user_id*[, *fields*]`)`

Retrieve an individual user.

Parameters: *user_id*: ID of the user *fields*: (optional) *list* of fields to return. Specific fields, or groups 'basic' or 'all'.

Returns: `WordPressUser` instance.

class `wordpress_xmlrpc.methods.users.`**`NewUser`**`(`*user*`)`
Create new user on the blog.

Parameters: *user*: A `WordPressUser` instance with at least *username*, *password*, and *email*. *send_mail*: (optional) Send a confirmation email to the new user.

Returns: ID of the newly-created blog user (an integer).

class `wordpress_xmlrpc.methods.users.`**`EditUser`**`(`*user_id*, *user*`)`
Edit an existing blog post.

Parameters: *user_id*: ID of the user to edit. *user*: *WordPressUser* instance.

Returns: *True* on successful edit.

class `wordpress_xmlrpc.methods.users.`**`DeleteUser`**`(`*user_id*[, *reassign_id*]`)`
Delete a blog user.

Parameters: *user_id*: ID of the blog user to delete. *reassign_id*: ID of the blog user to reassign this user's posts to.

Returns: *True* on successful deletion.

2.3.6 methods.media

class `wordpress_xmlrpc.methods.media.`**`GetMediaLibrary`**`(`[*filter*]`)`
Retrieve filtered list of media library items.

Parameters:

***filter*: *dict* with optional keys:**

- *number*: number of media items to retrieve
- *offset*: query offset
- ***parent_id*: ID of post the media item is attached to.** Use empty string (default) to show all media items. Use *0* to show unattached media items.
- *mime_type*: file mime-type to filter by (e.g., 'image/jpeg')

Returns: *list* of `WordPressMedia` instances.

class `wordpress_xmlrpc.methods.media.`**`GetMediaItem`**`(`*attachmend_id*`)`
Retrieve an individual media item.

Parameters: *attachment_id*: ID of the media item.

Returns: `WordPressMedia` instance.

class `wordpress_xmlrpc.methods.media.`**`UploadFile`**`(`*data*`)`
Upload a file to the blog.

Note: the file is not attached to or inserted into any blog posts.

Parameters:

***data*: *dict* with three items:**

- *name*: filename
- *type*: MIME-type of the file
- *bits*: base-64 encoded contents of the file. See xmlrpclib.Binary()
- *overwrite* (optional): flag to override an existing file with this name

Returns: *dict* with keys *id*, *file* (filename), *url* (public URL), and *type* (MIME-type).

2.3.7 methods.options

class `wordpress_xmlrpc.methods.options.`**`GetOptions`**(*options*)

Retrieve list of blog options.

Parameters: *options*: *list* of option names to retrieve; if empty, all options will be retrieved

Returns: *list* of `WordPressOption` instances.

class `wordpress_xmlrpc.methods.options.`**`SetOptions`**(*options*)

Update the value of an existing blog option.

Parameters: *options*: *dict* of key/value pairs

Returns: *list* of `WordPressOption` instances representing the updated options.

2.3.8 methods.demo

class `wordpress_xmlrpc.methods.demo.`**`SayHello`**

class `wordpress_xmlrpc.methods.demo.`**`AddTwoNumbers`**(*number1*, *number2*)

CHAPTER 3

Internals/Development

3.1 History/CHANGELOG

3.1.1 2.1

(May 12, 2012)

- Added missing import that was causing failures during exception handling.
- Updated fields to match changes in WordPress 3.4 between Beta 1 and RC1.
- Fixed some documentation bugs.

3.1.2 2.0

(April 16, 2012)

- **Major rewrite to support new XML-RPC features in WordPress 3.4.**
  - Rewrote `WordPressPost` and `methods.posts` module.
  - Removed CRUD methods for pages.
  - Added `WordPressTaxonomy` and `WordPressTerm` classes.
  - Added `methods.taxonomies` module.
  - Removed `WordPressCategory` and `WordPressTag` classes.
  - Removed `methods.categories` module.
  - Added `id` field to `WordPressMedia`.
- **Removed support for legacy-style (e.g., Blogger) methods.**
  - Removed `args_start_position` and `default_args_position` parameters on `XmlrpcMethod`.
  - Removed `requires_blog` parameter on `AuthenticatedMethod`.
- Set default values on all fields that are used in `str/unicode` to avoid `AttributeError` exceptions.
- Fixed bug with `FieldMap` that caused `False` boolean values to be ignored.
- Added ability to override `results_class` via a method class constructor kwarg.
- Added support for optional method arguments.

3.1.3 1.5

(August 27, 2011)

- Refactored `FieldMap` to be more flexible.
- Added new `Exception` subclasses for more specific error handling.

3.1.4 1.4

(July 31, 2011)

- Added support for post formats.
- Added support for media methods.

3.1.5 1.3

(July 31, 2011)

- Created test suite.
- Added support for page methods.
- Added support for post/page passwords.

3.1.6 1.2

(June 25, 2011)

- Added support for comments methods.

3.1.7 1.1

(October 11, 2010)

- Implemented automatic conversion of WordPress objects in method invocations.

3.1.8 1.0

(October 10, 2010)

- Initial release.

3.2 Testing

3.2.1 Requirements

`nose` is used as the test runner. Use `easy_install` or `pip` to install:

```
pip nose
```

3.2.2 Configuring against your server

To test this library, we must perform XML-RPC requests against an actual WordPress server. To configure against your own server:

- Copy the included `wp-config-sample.cfg` file to `wp-config.cfg`.
- Edit `wp-config.cfg` and fill in the necessary values.

3.2.3 Running Tests

Note: Be sure to have installed `nose` and created your `wp-config.cfg`.

To run the entire test suite, run the following from the root of the repository:

```
nosetests
```

To run a sub-set of the tests, you can specify a specific feature area:

```
nosetests -a posts
```

You can run against multiple areas:

```
nosetests -a posts -a comments
```

Or you can run everything except a specific area:

```
nosetests -a '!comments'
```

You can use all the normal `nose` command line options. For example, to increase output level:

```
nosetests -a demo --verbosity=3
```

Full usage details:

- nose

3.2.4 Contributing Tests

If you are submitting a patch for this library, please be sure to include one or more tests that cover the changes.

if you are adding new test methods, be sure to tag them with the appropriate feature areas using the `@attr()` decorator.

CHAPTER 4

Indices and tables

- *genindex*
- *modindex*
- *search*

Python Module Index