Facebook SDK for Python Documentation

Release 1.0.0

Martey Dodoo

April 03, 2016

Contents

Installation

The SDK currently supports Python 2.6, 2.7, 3.3, 3.4, and 3.5. The requests package is required.

We recommend using pip and virtualenv to install the SDK. Please note that the SDK's Python package is called **facebook-sdk**:

1.1 Installing from Git

For the newest features, you should install the SDK directly from Git.

```
virtualenv facebookenv
source facebookenv/bin/activate
pip install -e git+https://github.com/mobolic/facebook-sdk.git#egg=facebook-sdk
```

1.2 Installing a Released Version

If your application requires maximum stability, you will want to use a version of the SDK that has been officially released.

```
virtualenv facebookenv
source facebookenv/bin/activate
pip install facebook-sdk
```

CHAPTER 2

Integrating the SDK with Other Frameworks

2.1 Flask

The examples directory contains an example of using the SDK in Flask.

2.2 Google App Engine

Because the SDK uses *requests* (which requires socket support in Google App Engine), you will need to enable billing.

2.3 Tornado

The examples directory contains an example of using the SDK in Tornado.

CHAPTER 3

Support & Development

3.1 Mailing List

Questions about the SDK should be sent to its Google Group.

3.2 Reporting Bugs

Bugs with the SDK should be reported on the issue tracker at Github. Bugs with Facebook's Graph API should be reported on Facebook's bugtracker.

3.2.1 Security Issues

Security issues with the SDK that would adversely affect users if reported publicly should be sent through private email to the project maintainer at martey @ marteydodoo.com (GPG key ID is 0x2cd700988f74c455).

3.3 Contributing

3.3.1 Use Github Pull Requests

All potential code changes should be submitted as pull requests on Github. A pull request should only include commits directly applicable to its change (e.g. a pull request that adds a new feature should not include PEP8 changes in an unrelated area of the code).

3.3.2 Code Style

Code *must* be compliant with PEP 8. Use the latest version of pep8 or flake8 to catch issues.

Git commit messages should include a summary and proper line wrapping.

3.3.3 Update Tests and Documentation

All non-trivial changes should include full test coverage. Please review the package's documentation to ensure that it is up to date with any changes.

CHAPTER 4

API Reference

This page contains specific information on the SDK's classes, methods and functions.

4.1 class facebook.GraphAPI

A client for the Facebook Graph API. The Graph API is made up of the objects or nodes in Facebook (e.g., people, pages, events, photos) and the connections or edges between them (e.g., friends, photo tags, and event RSVPs). This client provides access to those primitive types in a generic way.

You can read more about Facebook's Graph API here.

Parameters

- `access_token` – A `string` that identifies a user, app, or page and can be used by the app to make graph API calls. Read more about access tokens here.
- `timeout` - A `float` describing (in seconds) how long the client will be waiting for a response from Facebook's servers. See more here.
- `version` - A `string` describing the version of Facebook's Graph API to use. Valid API versions are `2.0`, `2.1` and `2.2`. The default version is `2.0` and is used if the version keyword argument is not provided.
- `proxies` - A `dict` with proxy-settings that Requests should use. See Requests documentation.

Example

```
import facebook

graph = facebook.GraphAPI(access_token='your_token', version='2.2')
```

4.1.1 Methods

get_object

Returns the given object from the graph as a `dict`. A list of supported objects can be found here.

Parameters

- `id` – A `string` that is a unique ID for that particular resource.

Example

```
post = graph.get_object(id='post_id')
print(post['message'])
```

get_objects

Returns all of the given objects from the graph as a `dict`. Each given ID maps to an object.

Parameters

- `ids` – A `list` containing IDs for multiple resources.

Example

```
post_ids = ['post_id_1', 'post_id_2']
posts = graph.get_objects(ids=post_ids)

# Each given id maps to an object.
for post_id in post_ids:
    print(posts[post_id]['created_time'])
```

get_connections

Returns all connections for a given object as a `dict`.

Parameters

- `id` – A `string` that is a unique ID for that particular resource.
- `connection_name` - A `string` that specifies the connection or edge between objects, e.g., feed, friends, groups, likes, posts. If left empty, `get_connections` will simply return the authenticated user's basic information.

Example

```
# Get all of the authenticated user's friends
friends = graph.get_connections(id='me', connection_name='friends')

# Get all the comments from a post
comments = graph.get_connections(id='post_id', connection_name='comments')
```

put_object

Writes the given object to the graph, connected to the given parent.

Parameters

- `parent_object` – A `string` that is a unique ID for that particular resource. The `parent_object` is the parent of a connection or edge. E.g., profile is the parent of a feed, and a post is the parent of a comment.
- `connection_name` - A `string` that specifies the connection or edge between objects, e.g., feed, friends, groups, likes, posts.

Example

```
# Writes 'Hello, world' to the active user's wall.
graph.put_object(parent_object='me', connection_name='feed',
                 message='Hello, world')
```

```
# Writes a comment on a post
graph.put_object(parent_object='post_id', connection_name='comments',
                 message='First!')
```

put_wall_post

Writes a wall post to the given profile's wall. It defaults to writing to the authenticated user's wall if no `profile_id` is specified.

Parameters

- `message` - A `string` that will be posted to the user's wall.
- `attachment` - A `dict` that adds a structured attachment to the message being posted to the Wall. If you are sharing a URL, you will want to use the `attachment` parameter so that a thumbnail preview appears in the post. It should be a `dict` of the form:

```
 attachment =  {
     'name': '',
     'link': '',
     'caption': '',
     'description': '',
     'picture': ''
}
```

- `profile_id` - A `string` that is a unique ID for that particular user. Defaults to the authenticated user's wall.

Example

```
attachment =  {
    'name': 'Link name'
    'link': 'http://www.example.com/',
    'caption': 'Check out this example',
    'description': 'This is a longer description of the attachment',
    'picture': 'http://www.example.com/thumbnail.jpg'
}

graph.put_wall_post(message='Check this out...', attachment=attachment)
```

put_comment

Writes the given message as a comment on an object.

Parameters

- `object_id` - A `string` that is a unique id for a particular resource.
- `message` - A `string` that will be posted as the comment.

Example

```
graph.put_comment(object_id='post_id', message='Great post...')
```

put_like

Writes a like to the given object.

Parameters

- `object_id` - A `string` that is a unique id for a particular resource.

Example

```
graph.put_like(object_id='comment_id')
```

put_photo

https://developers.facebook.com/docs/graph-api/reference/user/photos#publish

Upload an image using multipart/form-data. Returns JSON with the IDs of the photo and its post.

Parameters

- `image` - A file object representing the image to be uploaded.
- `album_path` - A path representing where the image should be uploaded. Defaults to */me/photos* which creates/uses a custom album for each Facebook application.

Example

```
# Upload an image with a caption.
graph.put_photo(image=open('img.jpg', 'rb'), message='Look at this cool photo!')
# Upload a photo to an album.
graph.put_photo(image=open("img.jpg", 'rb'), album_path=album_id + "/photos")
# Upload a profile photo for a Page.
graph.put_photo(image=open("img.jpg", 'rb'), album_path=page_id + "/picture")
```

delete_object

Deletes the object with the given ID from the graph.

Parameters

- `id` - A `string` that is a unique ID for a particular resource.

Example

```
graph.delete_object(id='post_id')
```

CHAPTER 5

Changelog

5.1 Version 1.0.0

- Python 3 support.
- More comprehensive test coverage.
- Full Unicode support.
- Better exception handling.
- Vastly improved documentation.

5.2 Version 0.4.0 (2012-10-15)

- Add support for deleting application requests.
- Fix minor documentation error in README.
- Verify signed request parsing succeeded when creating OAuth token.
- Convert README to ReStructuredText.

5.3 Version 0.3.2 (2012-07-28)

- Add support for state parameters in auth dialog URLs.
- Fixes bug with Unicode app secrets.
- Add optional timeout support for faster API requests.
- Random PEP8 compliance fixes.

5.4 Version 0.3.1 (2012-05-16)

- Minor documentation updates.
- Removes the develop branch in favor of named feature branches.

This client library is designed to support the Facebook Graph API and the official Facebook JavaScript SDK, which is the canonical way to implement Facebook authentication. You can read more about the Graph API by accessing its official documentation.