Facebook SDK for Python Documentation

Release 3.1.0

Martey Dodoo

Nov 06, 2018

Contents

CHAPTER 1

Installation

The SDK currently supports Python 2.7 and Python 3.4-3.7. 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).

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. The default version is the oldest current version. It is used if the version keyword argument is not provided.
- `proxies` - A `dict` with proxy-settings that Requests should use. See Requests documentation.
- `session` - A Requests Session object.

Example

```
import facebook

graph = facebook.GraphAPI(access_token="your_token", version="2.12")
```

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.
- `**args` (optional) - keyword args to be passed as query params

Examples

```
# Get the message from a post.
post = graph.get_object(id='post_id', fields='message')
print(post['message'])
```

```
# Retrieve the number of people who say that they are attending or
# declining  to attend a specific event.
event = graph.get_object(id='event_id',
                         fields='attending_count,declined_count')
print(event['attending_count'])
print(event['declined_count'])
```

```
# Retrieve information about a website or page:
# https://developers.facebook.com/docs/graph-api/reference/url/
# Note that URLs need to be properly encoded with the "quote" function
# of urllib (Python 2) or urllib.parse (Python 3).
site_info = graph.get_object(id="https%3A//mobolic.com",
                             fields="og_object")
print(site_info["og_object"]["description"])
```

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.
- `**args` (optional) - keyword args to be passed as query params

Examples

```
# Get the time two different posts were created.
post_ids = ['post_id_1', 'post_id_2']
posts = graph.get_objects(ids=post_ids, fields="created_time")

for post in posts:
    print(post['created_time'])
```

```
# Get the number of people attending or who have declined to attend
# two different events.
event_ids = ['event_id_1', 'event_id_2']
events = graph.get_objects(ids=event_ids, fields='attending_count,declined_count')
```

(continues on next page)

(continued from previous page)

```
for event in events:
    print(event['declined_count'])
```

search

https://developers.facebook.com/docs/places/search

Valid types are: place, placetopic

Parameters

- `type` – A `string` containing a valid type.
- `**args` (optional) - keyword args to be passed as query params

Example

```
# Search for places near 1 Hacker Way in Menlo Park, California.
places = graph.search(type='place',
                      center='37.4845306,-122.1498183',
                      fields='name,location')

# Each given id maps to an object the contains the requested fields.
for place in places['data']:
    print('%s %s' % (place['name'].encode(),place['location'].get('zip')))
```

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.

Examples

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

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

get_all_connections

Iterates over all pages returned by a get_connections call and yields the individual items.

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.

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.

Examples

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

# Add a link and write a message about it.
graph.put_object(
   parent_object="me",
   connection_name="feed",
   message="This is a great website. Everyone should visit it.",
   link="https://www.facebook.com")

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

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.

Examples

```
# 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')
```

get_permissions

https://developers.facebook.com/docs/graph-api/reference/user/permissions/

Returns the permissions granted to the app by the user with the given ID as a `set`.

Parameters

- `user_id` - A `string` containing a user's unique ID.

Example

```
# Figure out whether the specified user has granted us the
# "public_profile" permission.
permissions = graph.get_permissions(user_id=12345)
print('public_profile' in permissions)
```

get_auth_url

https://developers.facebook.com/docs/facebook-login/manually-build-a-login-flow

Returns a Facebook login URL used to request an access token and permissions.

Parameters

- `app_id` - A `string` containing a Facebook appplication ID.
- `canvas_url` - A `string` containing the URL where Facebook should redirect after successful authentication.
- `perms` - An optional `list` of requested Facebook permissions.

Example

```
app_id = "1231241241"
canvas_url = "https://domain.com/that-handles-auth-response/"
perms = ["manage_pages","publish_pages"]
fb_login_url = graph.get_auth_url(app_id, canvas_url, perms)
print(fb_login_url)
```

CHAPTER 5

Changelog

5.1 Version 3.1.0 (2018-11-06)

- Add support for Graph API version 3.1.
- Remove support for Graph API version 2.7.
- Change default Graph API version to 2.8.

5.2 Version 3.0.0 (2018-08-08)

- Add support for Python 3.6 and 3.7.
- Remove support for Python 2.6 and 3.3.
- Add support for Graph API versions 2.8, 2.9, 2.10, 2.11, 2.12, and 3.0.
- Remove support for Graph API versions 2.1, 2.2, 2.3, 2.4, 2.5, and 2.6.
- Change default Graph API version to 2.7.
- Add support for requests' sessions (#201).
- Add versioning to access token endpoints (#322).
- Add new *get_all_connections* method to make pagination easier (#337).
- Add new *get_permissions* method to retrieve permissions that a user has granted an application (#264, #342).
- Remove *put_wall_post* method. Use *put_object* instead.
- Add search method (#362).
- Rename *auth_url* method to *get_auth_url* and move it into the Graph API object (#377, #378, #422).

5.3 Version 2.0.0 (2016-08-08)

- Add support for Graph API versions 2.6 and 2.7.
- Remove support for Graph API version 2.0 and FQL.
- Change default Graph API version to 2.1.
- Fix bug with debug_access_token method not working when the GraphAPI object's access token was set (#276).
- Allow offline generation of application access tokens.

5.4 Version 1.0.0 (2016-04-01)

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

5.5 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.

5.6 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.

5.7 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.