oauth-dropins Documentation

Release 1.9

Ryan Barrett

Mar 24, 2018

Contents

CHAPTER 1

About

This is a collection of drop-in Google App Engine request handlers for the initial OAuth client flows for many popular sites, including Blogger, Disqus, Dropbox, Facebook, Flickr, Google+, IndieAuth, Instagram, Medium, Tumblr, Twitter, and WordPress.com.

- Available on PyPi. Install with `pip install oauth-dropins`.
- *Click here for getting started docs.*
- Click here for reference docs.
- A demo app is deployed at oauth-dropins.appspot.com.

Requires either the App Engine Python SDK or the Google Cloud SDK (aka `gcloud`) with the `gcloud-appengine-python` and `gcloud-appengine-python-extras` components. All other dependencies are handled by pip and enumerated in requirements.txt. We recommend that you install with pip in a virtualenv. App Engine details here.

If you clone the repo directly or want to contribute, see *Development* for setup instructions.

This software is released into the public domain. See LICENSE for details.

CHAPTER 2

Quick start

Here's a full example of using the Facebook drop-in.

1. Make sure you have either the App Engine Python SDK version 1.9.15 or later (for vendor support) or the Google Cloud SDK (aka `gcloud`) installed and on your `$PYTHONPATH`, e.g. `export PYTHONPATH=$PYTHONPATH:/usr/local/google_appengine`. oauth-dropins's `setup.py` file needs it during installation.
2. Install oauth-dropins into a virtualenv somewhere your App Engine project's directory, e.g. `local/`:

```
source local/bin/activate
pip install oauth-dropins
```

3. Add this to the `appengine_config.py` file in your project's root directory (background):

```
from google.appengine.ext import vendor
vendor.add('local')
from oauth_dropins.appengine_config import *
```

4. Put your Facebook application's ID and secret in two plain text files in your app's root directory, `facebook_app_id` and `facebook_app_secret`. (If you use git, you'll probably also want to add them to your `.gitignore`.)
5. Create a `facebook_oauth.py` file with these contents:

```
from oauth_dropins import facebook
import webapp2

application = webapp2.WSGIApplication([
  ('/facebook/start_oauth', facebook.StartHandler.to('/facebook/oauth_callback')),
  ('/facebook/oauth_callback', facebook.CallbackHandler.to('/next'))]
```

6. Add these lines to `app.yaml`:

```
- url: /facebook/(start_oauth|oauth_callback)
  script: facebook_oauth.application
  secure: always
```

Voila! Send your users to `/facebook/start_oauth` when you want them to connect their Facebook account to your app, and when they're done, they'll be redirected to `/next?access_token=...` in your app.

All of the sites provide the same API. To use a different one, just import the site module you want and follow the same steps. The filenames for app keys and secrets also differ by site; appengine_config.py has the full list.

CHAPTER 3

Usage details

There are three main parts to an OAuth drop-in: the initial redirect to the site itself, the redirect back to your app after the user approves or declines the request, and the datastore entity that stores the user's OAuth credentials and helps you use them. These are implemented by *StartHandler*, *CallbackHandler*, and *auth entities*, respectively.

The request handlers are full WSGI applications and may be used in any Python web framework that supports WSGI (PEP 333). Internally, they're implemented with webapp2.

3.1 `StartHandler`

This HTTP request handler class redirects you to an OAuth-enabled site so it can ask the user to grant your app permission. It has two useful methods:

- `to(callback_path, scopes=None)` is a factory method that returns a request handler class you can use in a WSGI application. The argument should be the path mapped to *CallbackHandler* in your application. This also usually needs to match the callback URL in your app's configuration on the destination site.

If you want to add OAuth scopes beyond the default one(s) needed for login, you can pass them to the `scopes` kwarg as a string or sequence of strings, or include them in the `scopes` query parameter in the POST request body. This is currently supported with Facebook, Google+, Blogger, and Instagram.

Some of the sites that use OAuth 1 support alternatives. For Twitter, `StartHandler.to` takes an additional `access_type` kwarg that may be `read` or `write`. It's passed through to Twitter x_auth_access_type. For Flickr, the start handler accepts a `perms` POST query parameter that may be `read`, `write` or `delete`; it's passed through to Flickr unchanged. (Flickr claims it's optional, but sometimes breaks if it's not provided.)

- `redirect_url(state=None)` returns the URL to redirect to at the destination site to initiate the OAuth flow. `StartHandler` will redirect here automatically if it's used in a WSGI application, but you can also instantiate it and call this manually if you want to control that redirect yourself:

```
class MyHandler(webapp2.RequestHandler):
  def get(self):
    ...
    handler_cls = facebook.StartHandler.to('/facebook/oauth_callback')
```

(continues on next page)

(continued from previous page)

```
    handler = handler_cls(self.request, self.response)
    self.redirect(handler.redirect_url())
```

However, this is *not* currently supported for Google+ and Blogger. Hopefully that will be fixed in the future.

3.2 CallbackHandler

This class handles the HTTP redirect back to your app after the user has granted or declined permission. It also has two useful methods:

- `to(callback_path)` is a factory method that returns a request handler class you can use in a WSGI application, similar to *StartHandler*. The callback path is the path in your app that users should be redirected to after the OAuth flow is complete. It will include a `state` query parameter with the value provided by the `StartHandler`. It will also include an OAuth token in its query parameters, either `access_token` for OAuth 2.0 or `access_token_key` and `access_token_secret` for OAuth 1.1. It will also include an `auth_entity` query parameter with the string key of an *auth entity* that has more data (and functionality) for the authenticated user. If the user declined the OAuth authorization request, the only query parameter besides `state` will be `declined=true`.
- `finish(auth_entity, state=None)` is run in the initial callback request after the OAuth response has been processed. `auth_entity` is the newly created auth entity for this connection, or `None` if the user declined the OAuth authorization request.

By default, `finish` redirects to the path you specified in `to()`, but you can subclass `CallbackHandler` and override it to run your own code inside the OAuth callback instead of redirecting:

```
class MyCallbackHandler(facebook.CallbackHandler):
  def finish(self, auth_entity, state=None):
    self.response.write('Hi %s, thanks for connecting your %s account.' %
        (auth_entity.user_display_name(), auth_entity.site_name()))
```

However, this is *not* currently supported for Google+ and Blogger. Hopefully that will be fixed in the future.

3.3 Auth entities

Each site defines an App Engine datastore ndb.Model class that stores each user's OAuth credentials and other useful information, like their name and profile URL. The class name is of the form SiteAuth, e.g. FacebookAuth. Here are the useful methods:

- `site_name()` returns the human-readable string name of the site, e.g. "Facebook".
- `user_display_name()` returns a human-readable string name for the user, e.g. "Ryan Barrett". This is usually their first name, full name, or username.
- `access_token()` returns the OAuth access token. For OAuth 2 sites, this is a single string. For OAuth 1.1 sites (currently just Twitter, Tumblr, and Flickr), this is a `(string key, string secret)` tuple.

The following methods are optional. Auth entity classes usually implement at least one of them, but not all.

- `api()` returns a site-specific API object. This is usually a third party library dedicated to the site, e.g. Tweepy or python-instagram. See the site class's docstring for details.
- `urlopen(data=None, timeout=None)` wraps `urllib2.urlopen()` and adds the OAuth credentials to the request. Use this for making direct HTTP request to a site's REST API. Some sites may provide `get()` instead, which wraps `requests.get()`.

- `http()` returns an `httplib2.Http` instance that adds the OAuth credentials to requests.

CHAPTER 4

Troubleshooting/FAQ

1. If you get this error:

```
bash: ./bin/easy_install: ...bad interpreter: No such file or directory
```

You've probably hit this open virtualenv bug (fixed but not merged): virtualenv doesn't support paths with spaces.

The easy fix is to recreate the virtualenv in a path without spaces. If you can't do that, then after creating the virtualenv, but before activating it, edit the activate, easy_install and pip files in `local/bin/` to escape any spaces in the path.

For example, in `activate`, `VIRTUAL_ENV=".../has space/local"` becomes `VIRTUAL_ENV=".../has\ space/local"`, and in `pip` and `easy_install` the first line changes from `#!".../has space/local/bin/python"` to `#!".../has\ space/local/bin/python"`.

This should get virtualenv to install in the right place. If you do this wrong at first, you'll have installs in `/usr/local/lib/python2.7/site-packages` that you need to delete, since they'll prevent virtualenv from installing into the local `site-packages`.

1. If you're using Twitter, and `import requests` or something similar fails with:

```
ImportError: cannot import name certs
```

or you see an exception like:

```
File ".../site-packages/tweepy/auth.py", line 68, in _get_request_token
  raise TweepError(e)
TweepError: must be _socket.socket, not socket
```

...you need to configure App Engine's SSL. Add this to your `app.yaml`:

```
libraries:
- name: ssl
  version: latest
```

If you use dev_appserver, you'll also need to apply this workaround (more background). Annoying, I know.

1. If you see errors importing or using `tweepy`, it may be because `six.py` isn't installed. Try `pip install six` manually. `tweepy` does include `six` in its dependencies, so this shouldn't be necessary. Please let us know if it happens to you so we can debug!
2. If you get an error like this:

```
  File "oauth_dropins/webutil/test/__init__.py", line 5, in <module>
    import dev_appserver
ImportError: No module named dev_appserver
...
InstallationError: Command python setup.py egg_info failed with error code 1 in /
↪home/singpolyma/src/bridgy/src/oauth-dropins-master
```

. . . you either don't have `/usr/local/google_appengine` in your `PYTHONPATH`, or you have it as a relative directory. pip requires fully qualified directories.

1. If you get an error like this:

```
Running setup.py develop for gdata
...
error: option --home not recognized
...
InstallationError: Command /usr/bin/python -c "import setuptools, tokenize; __
↪file__='/home/singpolyma/src/bridgy/src/gdata/setup.py';␣
↪exec(compile(getattr(tokenize, 'open', open)(__file__).read().replace('\r\n',
↪'\n'), __file__, 'exec'))" develop --no-deps --home=/tmp/tmprBISz_ failed with␣
↪error code 1 in .../src/gdata
```

. . . you may be hitting Pip bug 1833. Are you passing `-t` to `pip install`? Use the virtualenv instead, it's your friend. If you really want `-t`, try removing the `-e` from the lines in `requirements.freeze.txt` that have it.

CHAPTER 5

Changelog

5.1 1.8 - 2017-08-29

- Facebook:
  - Upgrade Graph API from v2.6 to v2.10.
- Flickr:
  - Fix broken `FlickrAuth.urlopen()` method.
- Medium:
  - Bug fix for Medium OAuth callback error handling.
- IndieAuth:
  - Store authorization endpoint in state instead of rediscovering it from `me` parameter, which is going away.

5.2 1.7 - 2017-02-27

- Updates to bundled webutil library, notably WideUnicode class.

5.3 1.6 - 2016-11-21

- Add auto-generated docs with Sphinx. Published at oauth-dropins.readthedocs.io.
- Fix Dropbox bug with fetching access token.

5.4 1.5 - 2016-08-25

- Add Medium.

5.5 1.4 - 2016-06-27

- Upgrade Facebook API from v2.2 to v2.6.

5.6 1.3 - 2016-04-07

- Add IndieAuth.
- More consistent logging of HTTP requests.
- Set up Coveralls.

5.7 1.2 - 2016-01-11

- Flickr:
  - Add upload method.
  - Improve error handling and logging.
- Bug fixes and cleanup for constructing scope strings.
- Add developer setup and troubleshooting docs.
- Set up CircleCI.

5.8 1.1 - 2015-09-06

- Flickr: split out flickr_auth.py file.
- Add a number of utility functions to webutil.

5.9 1.0 - 2015-06-27

- Initial PyPi release.

CHAPTER 6

Development

You'll need the App Engine Python SDK version 1.9.15 or later (for vendor support) or the Google Cloud SDK (aka `gcloud`) with the `gcloud-appengine-python` and `gcloud-appengine-python-extras` components. Add them to your `$PYTHONPATH`, e.g. `export PYTHONPATH=$PYTHONPATH:/usr/local/google_appengine`, and then run:

```
git submodule init
git submodule update
virtualenv local
source local/bin/activate
pip install -r requirements.txt

# We install gdata in source mode, and App Engine doesn't follow .egg-link
# files, so add a symlink to it.
ln -s ../../../src/gdata/src/gdata local/lib/python2.7/site-packages/gdata
ln -s ../../../src/gdata/src/atom local/lib/python2.7/site-packages/atom

python setup.py test
```

Most dependencies are clean, but we've made patches to gdata-python-client below that we haven't (yet) tried to push upstream. If we ever switch its submodule repo for, make sure the patches are included!

- snarfed/gdata-python-client@fabb622
- snarfed/gdata-python-client@8453e33

To deploy:

`python -m unittest discover && git push && ~/google_appengine/appcfg.py update .`

The docs are built with Sphinx, including apidoc, autodoc, and napoleon. Configuration is in docs/conf.py To build them, first install Sphinx with `pip install sphinx`. (You may want to do this outside your virtualenv; if so, you'll need to reconfigure it to see system packages with `virtualenv --system-site-packages local`.) Then, run docs/build.sh.

To convert README.md to README.rst for PyPI or index.rst for Sphinx:

CHAPTER 7

Related work

- Python Social Auth

CHAPTER 8

TODO

- Google+ and Blogger need some love:
  - handle declines
  - allow overriding `CallbackHandler.finish()`
  - support `StartHandler.redirect_url()`
  - allow more than one `CallbackHandler` per app
- clean up app key/secret file handling. (standardize file names? put them in a subdir?)
- implement CSRF protection for all sites
- implement Blogger's v3 API