granary Documentation

Release 1.6

Ryan Barrett

November 27, 2016

Contents

CHAPTER 1

About

Granary is a library and REST API that converts between a wide variety of formats:

- Facebook, Flickr, Google+, Instagram, and Twitter native APIs
- Instagram and Google+ scraped HTML
- ActivityStreams
- microformats2 HTML
- microformats2 JSON
- Atom
- XML

Here's how to get started:

- Granary is available on PyPi. Install with `pip install granary`.
- *Click here for getting started docs.*
- Click here for reference docs.
- The REST API and demo app are deployed at granary-demo.appspot.com.

CHAPTER 2

Using

All dependencies are handled by pip and enumerated in requirements.txt. We recommend that you install with pip in a virtualenv. (App Engine details.)

The library and REST API are both based on the OpenSocial Activity Streams service.

Let's start with an example. This code using the library:

```
from granary import twitter
...
tw = twitter.Twitter(ACCESS_TOKEN_KEY, ACCESS_TOKEN_SECRET)
tw.get_activities(group_id='@friends')
```

is equivalent to this `HTTP GET` request:

```
https://granary-demo.appspot.com/twitter/@me/@friends/@app/
  ?access_token_key=ACCESS_TOKEN_KEY&access_token_secret=ACCESS_TOKEN_SECRET
```

They return the authenticated user's Twitter stream, ie tweets from the people they follow. Here's the JSON output:

```
{
  "itemsPerPage": 10,
  "startIndex": 0,
  "totalResults": 12,
  "items": [{
      "verb": "post",
      "id": "tag:twitter.com,2013:374272979578150912",
      "url": "http://twitter.com/evanpro/status/374272979578150912",
      "content": "Getting stuff for barbecue tomorrow. No ribs left! Got some nice tenderloin though
      "actor": {
      "username": "evanpro",
        "displayName": "Evan Prodromou",
        "description": "Prospector.",
        "url": "http://twitter.com/evanpro",
      },
      "object": {
        "tags": [{
            "url": "http://4sq.com/1cw5vf6",
            "startIndex": 113,
            "length": 22,
            "objectType": "article"
          }, "..."],
      },
    }, "..."]
```

```
  "..."
}
```

The request parameters are the same for both, all optional: `USER_ID` is a source-specific id or `@me` for the authenticated user. `GROUP_ID` may be `@all`, `@friends` (currently identical to `@all`), `@self`, or `@search`; `APP_ID` is currently ignored; best practice is to use `@app` as a placeholder.

Paging is supported via the `startIndex` and `count` parameters. They're self explanatory, and described in detail in the OpenSearch spec and OpenSocial spec.

When using the `GROUP_ID` `@search` (for platforms that support it — currently Twitter and Instagram), provide a search string via the `q` parameter. The API is loosely based on the OpenSearch spec, the OpenSocial Core Container spec, and the OpenSocial Core Gadget spec.

Output data is JSON Activity Streams 1.0 objects wrapped in the OpenSocial envelope, which puts the activities in the top-level `items` field as a list and adds the `itemsPerPage`, `totalCount`, etc. fields.

Most Facebook requests and all Twitter, Google+, Instagram, and Flickr requests will need OAuth access tokens. If you're using Python on Google App Engine, oauth-dropins is an easy way to add OAuth client flows for these sites. Otherwise, here are the sites' authentication docs: Facebook, Flickr, Google+, Instagram, Twitter.

If you get an access token and pass it along, it will be used to sign and authorize the underlying requests to the sources providers. See the demos on the REST API *endpoints above* for examples.

CHAPTER 3

Using the REST API

The *endpoints above* all serve the OpenSocial Activity Streams REST API. Request paths are of the form:

```
/USER_ID/GROUP_ID/APP_ID/ACTIVITY_ID?startIndex=...&count=...&format=FORMAT&access_token=...
```

All query parameters are optional. `FORMAT` may be `json` (the default), `xml`, or `atom`, both of which return Atom. The rest of the path elements and query params are *described above*.

Errors are returned with the appropriate HTTP response code, e.g. 403 for Unauthorized, with details in the response body.

By default, responses are cached and reused for 5m without re-fetching the source data. (Instagram responses are cached for 60m.) You can prevent this by adding the `cache=false` query parameter to your request.

To use the REST API in an existing ActivityStreams client, you'll need to hard-code exceptions for the domains you want to use e.g. `facebook.com`, and redirect HTTP requests to the corresponding *endpoint above*.

The web UI (granary-demo.appspot.com) currently only fetches Facebook access tokens for users. If you want to use it to access a Facebook page, you'll need to get an access token manually with the Graph API Explorer (click on the *Get To...* drop-down) . Then, log into Facebook on granary-demo.appspot.com and paste the page access token into the `access_token` text box.

(Google+ pages aren't supported in their API.)

CHAPTER 4

Using the library

See the *example above* for a quick start guide.

Clone or download this repo into a directory named `granary` (note the underscore instead of dash). Each source works the same way. Import the module for the source you want to use, then instantiate its class by passing the HTTP handler object. The handler should have a `request` attribute for the current HTTP request.

The useful methods are `get_activities()` and `get_actor()`, which returns the current authenticated user (if any). See the individual method docstrings for details. All return values are Python dicts of decoded ActivityStreams JSON.

The `microformats2.*_to_html()` functions are also useful for rendering ActivityStreams objects as nicely formatted HTML.

CHAPTER 5

Troubleshooting/FAQ

Check out the oauth-dropins Troubleshooting/FAQ section. It's pretty comprehensive and applies to this project too. For searchability, here are a handful of error messages that have solutions there:

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

ImportError: cannot import name certs

ImportError: cannot import name tweepy

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

CHAPTER 6

Future work

We'd love to add more sites! Off the top of my head, YouTube, Tumblr, WordPress.com, Sina Weibo, Qzone, and RenRen would be good candidates. If you're looking to get started, implementing a new site is a good place to start. It's pretty self contained and the existing sites are good examples to follow, but it's a decent amount of work, so you'll be familiar with the whole project by the end.

CHAPTER 7

Development

Pull requests are welcome! Feel free to ping me with any questions.

You'll need the App Engine Python SDK version 1.9.15 or later (for vendor support). Add it to your `$PYTHONPATH`, e.g. `export PYTHONPATH=$PYTHONPATH:/usr/local/google_appengine`, and then run:

```
virtualenv local
source local/bin/activate
pip install -r requirements.txt
python setup.py test
```

If you send a pull request, please include (or update) a test for the new functionality if possible! The tests require the App Engine SDK.

If you want to work on oauth-dropins at the same time, install it in "source" mode with `pip install -e <path to oauth-dropins repo>`.

To deploy:

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

To deploy facebook-atom, twitter-atom, instagram-atom, and plusstreamfeed after a granary change:

```
#!/bin/tcsh
foreach s (facebook-atom twitter-atom instagram-atom plusstreamfeed)
  cd ~/src/$s && ~/google_appengine/appcfg.py update .
end
```

To deploy the old `*-activitystreams.appspot.com` apps:

```
cd old_apps
rm -f app.yaml && ln -s app.twitter.yaml app.yaml && \
  ~/google_appengine/appcfg.py update . && \
rm -f app.yaml && ln -s app.facebook.yaml app.yaml && \
  ~/google_appengine/appcfg.py update . && \
rm -f app.yaml && ln -s app.instagram.yaml app.yaml && \
  ~/google_appengine/appcfg.py update . && \
git co -- app.yaml
```

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.

This ActivityStreams validator is useful for manual testing.

CHAPTER 8

Related work

Apache Streams is a similar project that translates between storage systems and database as well as social schemas. It's a Java library, and its design is heavily structured. Here's the list of formats it supports. It's mainly used by People Pattern.

Gnip similarly converts social network data to ActivityStreams and supports many more source networks. Unfortunately, it's commercial, there's no free trial or self-serve signup, and plans start at $500.

DataSift looks like broadly the same thing, except they offer self-serve, pay as you go billing, and they use their own proprietary output format instead of ActivityStreams. They're also aimed more at data mining as opposed to individual user access.

Cliqset's FeedProxy used to do this kind of format translation, but unfortunately it and Cliqset died.

Facebook used to officially support ActivityStreams, but that's also dead.

There are a number of products that download your social network data, normalize it, and let you query and visualize it. SocialSafe and ThinkUp are two of the most mature. There's also the lifelogging/lifestream aggregator vein of projects that pull data from multiple source sites. Storytlr is a good example. It doesn't include Facebook, Google+, or Instagram, but does include a number of smaller source sites. There are lots of others, e.g. the Lifestream WordPress plugin. Unfortunately, these are generally aimed at end users, not developers, and don't usually expose libraries or REST APIs.

On the open source side, there are many related projects. php-mf2-shim adds microformats2 to Facebook and Twitter's raw HTML. sockethub is a similar "polyglot" approach, but more focused on writing than reading.

CHAPTER 9

Changelog

9.1 1.6 - 2016-11-26

- Twitter:
  - Handle new "extended" tweets with hidden reply-to @-mentions and trailing URLs for media, quote tweets, etc. Background: https://dev.twitter.com/overview/api/upcoming-changes-to-tweets
  - Bug fix: ensure like.author.displayName is a plain unicode string so that it can be pickled normally, e.g. by App Engine's memcache.
  - Bug fix: handle names with emoji correctly in favorites_html_to_likes().
  - Bug fix: handle search queries with unicode characters.
- Atom:
  - Render full original quoted tweet in retweets of quote tweets.
- microformats2 HTML:
  - Optionally follow and fetch rel="author" links.
  - Improve mapping between microformats2 and ActivityStreams 'photo' types. (mf2 'photo' type is a note or article *with* a photo, but AS 'photo' type *is* a photo. So, map mf2 photos to underlying type without photo.)
  - Support location properties beyond h-card, e.g. h-adr, h-geo, u-geo, and even when properties like latitude and longitude appear at the top level.
- Error handling: return HTTP 502 for non-JSON API responses, 504 for connection failures.

9.2 1.5 - 2016-08-25

- REST API:
  - Support tag URI for user id, app id, and activity id.
- Twitter:
  - Better error message when uploading a photo with an unsupported type.
  - Only include original quote tweets, not retweets of them.
  - Skip fetching retweets for protected accounts since the API call always 403s.

- Flickr:
  - Better username detection. Flickr's API is very inconsistent about username vs real name vs path alias. This specifically detects when a user name is probably actually a real name because it has a space.
  - Uploading: detect and handle App Engine's 10MB HTTP request limit.
  - Bug fix in create: handle unicode characters in photo/video description, hashtags, and comment text.
- Atom:
  - Bug fix: escape &s in attachments' text (e.g. quote tweets).

9.3 1.4.1 - 2016-06-27

- Bump oauth-dropins requirement to 1.4.

9.4 1.4.0 - 2016-06-27

- REST API:
  - Cache silo requests for 5m by default, 60m for Instagram because they aggressively blocking scraping. You can skip the cache with the new cache=false query param.
- Facebook:
  - Upgrade from API v2.2 to v2.6. https://developers.facebook.com/docs/apps/changelog
  - Add reaction support.
  - De-dupe event RSVPs by user.
- Twitter:
  - Switch create() to use brevity for counting characters. https://github.com/kylewm/brevity
  - Fix bug in create() that occasionally incorrectly escaped ., +, and - characters.
  - Fix text rendering bug when there are multipl photos/videos.
  - When replying to yourself, don't add a self @-mention.
- Instagram:
  - Fix bugs in scraping.
- Upgrade to requests 2.10.0 and requests-toolbelt 0.60, which support App Engine.

9.5 1.3.1 - 2016-04-07

- Update oauth-dropins dependency to >=1.3.

9.6 1.3.0 - 2016-04-06

- Support posting videos! Currently in Facebook, Flickr, and Twitter.
- Instagram:
  - Add support for scraping, since they're locking down their API and requiring manual approval.
  - Linkify @-mentions in photo captions.
- Facebook:
  - Fetch Open Graph stories aka `news.publish` actions.
  - Many bug fixes for photo posts: better privacy detection, fix bug that attached comments to wrong posts.
- Twitter:
  - Handle all photos/videos attached to a tweet, not just the first.
  - Stop fetching replies to @-mentions.
- Atom:
  - Render attachments.
  - Add `xml:base`.
- microformats2:
  - Load and convert h-card.
  - Implement full post type discovery algorithm, using mf2util. https://indiewebcamp.com/post-type-discovery
  - Drop support for h-as-* classes, both incoming and outgoing. They're deprecated in favor of post type discovery.
  - Drop old deprecated `u-like` and `u-repost` properties.
- Misc bug fixes.
- Set up Coveralls.

9.7 1.2.0 - 2016-01-11

- Improve original post discovery algorithm. (bridgy #51)
- Flickr tweaks. (bridgy #466)
- Add mf2, activitystreams, atom, and search to interactive UI. (#31, #29)
- Improved post type discovery (using mf2util).
- Extract user web site links from all fields in profile (e.g. description/bio).
- Add fabricated fragments to comment/like permalinks (e.g. #liked-by-user123) so that object urls are always unique (multiple silos).
- Improve formatting/whitespace support in create/preview (multiple silos).
- Google+:
  - Add search.
- Facebook:

  - Fetch more things in get_activities: photos, events, RSVPs.
  - Support person tags in create/preview.
  - Prevent facebook from automatically consolidating photo posts by uploading photos to "Timeline Photos" album.
  - Include title in create/preview.
  - Improve object id parsing/resolving.
  - Improve tag handling.
  - Bug fix for fetching nested comments.
  - Misc improvements, API error/flakiness handling.
- Flickr:
  - Create/preview support for photos, comments, favorites, tags, person tags, location.
- Twitter:
  - Create/preview support for location, multiple photos.
  - Fetch quote tweets.
  - Fetching user mentions improvements, bug fixes.
  - Fix embeds.
  - Misc AS conversion improvements.
- microformats2:
  - Improve like and repost rendering.
- Misc bug fixes.
- Set up CircleCI.

9.8 1.1.0 - 2015-09-06

- Add Flickr.
- Facebook:
  - Fetch multiple id formats, e.g. with and without USERID_ prefix.
  - Support threaded comments.
  - Switch from /posts API endpoint to /feed.
- Google+:
  - Support converting plus.google.com HTML to ActivityStreams.
- Instagram:
  - Support location.
- Improve original post discovery algorithm.
- New logo.

9.9 1.0.1 - 2015-07-11

- Bug fix for atom template rendering.
- Facebook, Instagram: support access_token parameter.

9.10 1.0 - 2015-07-10

- Initial PyPi release.