OGRe Documentation

Release 4.5.1

David Tucker

May 23, 2016

Contents

1 Contents **3**
1.1 About . . . 3
1.2 Tutorial . . . 4
1.3 API Reference . . . 7
1.4 License . . . 10

2 Indices and tables **13**

Python Module Index **15**

OGRe fetches geotagged data from publicly available APIs based on a provided keyword, location, period of time, or media type. Results are returned in GeoJSON format.

CHAPTER 1

Contents

1.1 About

OGRe is a component of a larger project called OpenFusion that aims to gather crowdsourced data and produce useful visualizations from it.

1.1.1 OpenFusion

Development began for a "Geotagged Sensor Fusion" idea in late 2013 as a Senior Design project at the University of California at Santa Cruz with sponsorship from Lawrence Livermore National Laboratory. There are currently 4 components to the project including a mobile data logger, a web application, a retriever, and a visualizer. Some examples of questions this project tries to answer are listed below.

- What images were posted in Times Square when the temperature was 50 degrees?
- What was posted within 1 kilometer of any Tweet mentioning Barack Obama?
- What posts mentioning the Super Bowl were created within 5 minutes of a humidity measurement of 50%?

See also:

https://gsf.soe.ucsc.edu/

Mobile Data Logger

The mobile data collection suite consists of a series of sensors and a mobile application. In addition to utilizing sensors common in mobile devices (i.e. the camera, microphone, and GPS receiver), students developed a pluggable device with onboard sensors. The external sensors connect to via a 3.5mm headset jack and relay measurements using a microcontroller. Gathered data then gets transmitted to the OpenFusion web application.

Note: Currently, only the iPhone 4S and 5S are officially supported.

See also:

https://github.com/mbaptist23/open-fusion-iOS

Web Application

Mobile devices upload geotagged data to the web application where it is stored in a database for later retrieval. OpenFusion coordinates programmatic uploads and downloads through a public REST API. Users may access a deployment of the web application online using their favorite browser. There, they can query the database of measurements and use it to create fusions with data from other sources.

See also:

https://github.com/bkeyoumarsi/open-fusion-webapp

Retriever

The web application uses a retriever called OGRe to collect relevant data from public sources such as Twitter. It merges the results with gathered data and writes the combined set to a GeoJSON file that gets handed off to the visualizer.

See also:

https://github.com/dmtucker/ogre

Visualizer

After a query is run, a client-side visualizer called Vizit fetches data from the web application via AJAX and plots it on a map or timeline. Vizit is capable of rendering any GeoJSON file, and it supports a number of special properties to affect the resulting visualization.

See also:

https://github.com/dmtucker/vizit

1.2 Tutorial

Follow this simple guide to quickly get started using OGRe.

1.2.1 Installation

To get OGRe, use one of the following methods:

1. pip *(recommended)*

```
$ pip install ogre
```

2. easy_install

```
$ easy_install ogre
```

3. git

```
$ git clone http://github.com/dmtucker/ogre.git
$ cd ogre
$ python setup.py install
```

Note: This method requires separate installation(s) for dependencies!

1.2.2 Usage

To use OGRe, you must import it into your module:

```
from ogre import OGRe
```

Next, create an OGRe instance with your API credentials:

```
retriever = OGRe(
    keys={
        'Twitter': {
            'consumer_key': 'xxxxxxxxxxxxxxxxxxxxx',
            'access_token': 'xxxxxxxxxxxxxxxxxxxxxxxxxxx' +
                            'xxxxxxxxxxxxxxxxxxxxxxxxxxx' +
                            'xxxxxxxxxxxxxxxxxxxxxxxxxxx' +
                            'xxxxxxxxxxxxxxxxxxxxxxxxxxx'
        }
    }
)
```

Finally, make queries using the *.get* method.

Note: "The [Twitter] Search API is not complete index of all Tweets, but instead an index of recent Tweets. At the moment that index includes between 6-9 days of Tweets."

OGRe may be also be executed directly as shown below.

```
$ python -m ogre
```

or (if OGRe was installed with pip)

```
$ ogre
```

As one would expect, API keys are necessary whenever OGRe is run, and they may be passed one of two ways: the *keys* parameter or via environment variables. If the latter is preferred, the following table shows what variables are needed for each source:

Source	Key Variable(s)
Twitter	TWITTER_CONSUMER_KEY, TWITTER_ACCESS_TOKEN

Examples

To find 50 image-less Tweets posted within 1km of Twitter headquarters, use:

```
retriever.fetch(
    sources=('Twitter',),
    media=('text',),
    keyword='',
    quantity=50,
    location=(37.781157, -122.398720, 1, 'km'),
    interval=None
)
```

Note: Either a keyword or location is required.

To issue the same query directly from the command line, use the following:

```
$ python -m ogre --keys "{\
>   'Twitter': {\
>     'consumer_key': '<Insert Twitter Consumer Key.>',\
>     'access_token': '<Insert Twitter Access Token.>'\
>   }\
> }\
> --sources Twitter \
> --media text \
> --quantity 50 \
> --location 37.781157 -122.398720 1 km
```

Alternatively, environment variables are checked when keys are unspecified.

```
$ export TWITTER_CONSUMER_KEY='<Insert Twitter Consumer Key.>'
$ export TWITTER_ACCESS_TOKEN='<Insert Twitter Access Token.>'
$ python -m ogre --sources Twitter --media text \
> --quantity 50 --location 37.781157 -122.398720 1 km
```

See also:

For a description of all available command line arguments, run:

```
$ python -m ogre --help
```

Results return as a single GeoJSON FeatureCollection. So, the example above could return:


```
{
    "type": "FeatureCollection",
    "features": [
        {
            "geometry": {
                "type": "Point",
                "coordinates": [
                    -122.3970807,
                    37.77541704
                ]
            },
            "type": "Feature",
            "properties": {
                "source": "Twitter",
                "text": "Sending good thoughts to my babe @annecurtissmith...",
                "timestamp": "2014-04-04T02:03:28.431000Z"
            }
        },
        {
            "geometry": {
                "type": "Point",
                "coordinates": [
                    -122.41160509,
                    37.78093192
                ]
           },
             "type": "Feature",
            "properties": {
                "source": "Twitter",
                "text": "I'm at Huckleberry Bicycles...",
                "timestamp": "2014-04-04T02:03:13.190000Z"
            }
        },
        ...
```

```
    ]
}
```

Say we wanted to run the same query and possibly have images returned too. Additional mediums can be specified with subsequent *media* flags like this:

```
$ python -m ogre \
> --sources Twitter \
> --media text --media image \
> --quantity 50 \
> --location 37.781157 -122.398720 1 km
```

1.3 API Reference

OGRe Query Handler

`OGRe` – retriever object template

`OGRe.fetch()` – method for making a retriever fetch data

`OGRe.get()` – alias of `OGRe.fetch()`

class `ogre.api.`**`OGRe`**(*keys*)

Create objects that contain API keys and API access points.

OGRe was made a class to avoid requiring API keys with every API call. Since this is a library meant for developers, it didn't seem appropriate to use a configuration file. Also, importing the keys from the OS environment subjects them to data leak. This way developers are responsible for keeping their keys safe, and they can use the environment if they choose. Twython, the Twitter API wrapper, also uses this scheme.

`fetch()` – method for retrieving data from a public source

`get()` – backwards-compatible alias of `fetch()`

fetch(*sources*, *media=('image', 'sound', 'text', 'video')*, *keyword=''*, *quantity=15*, *location=None*, *interval=None*, ***kwargs*)

Get geotagged data from public APIs.

See also:

`ogre.validation.validate()` describes the format each parameter must have. It is also a good idea to check the module of any sources that will be searched (e.g. `ogre.Twitter.twitter()`) for extra constraints on parameters.

Parameters

- **sources** (*tuple*) – Specify public APIs to get content from (required). "Twitter" is currently the only supported source.
- **media** (*tuple*) – Specify content mediums to fetch. "image", "sound", "text", and "video" are supported.
- **keyword** (*str*) – Specify search criteria.
- **quantity** (*int*) – Specify a quota of results to fetch.
- **location** (*tuple*) – Specify a place (latitude, longitude, radius, unit) to search.
- **interval** (*tuple*) – Specify a period of time (earliest, latest) to search.

Raises ValueError

Return type dict

Returns GeoJSON FeatureCollection

Note: Additional runtime modifiers may be specified to change the way results are retrieved. Runtime modifiers are relayed to each source module, and that is where they are documented.

---

get(*sources*, *keyword=''*, *what=('image', 'sound', 'text', 'video')*, *when=None*, *where=None*, *how_many=15*, ***kwargs*)

Provide a backwards-compatible alias of `fetch()`.

Parameters

- **sources** (*tuple*) – This parameter corresponds directly in `fetch()`.
- **keyword** (*str*) – This parameter corresponds directly in `fetch()`.
- **what** (*tuple*) – This parameter corresponds to *media* in `fetch()`.
- **when** (*tuple*) – This parameter corresponds to *interval* in `fetch()`.
- **where** (*tuple*) – This parameter corresponds to *location* in `fetch()`.
- **how_many** (*int*) – This parameter corresponds to *quantity* in `fetch()`.

Return type dict

Returns GeoJSON FeatureCollection

Note: `get()` is deprecated. `fetch()` should be used instead.

OGRe Twitter Interface

`twitter()` : method for fetching data from Twitter

ogre.Twitter.**sanitize_twitter**(*keys*, *media=('image', 'text')*, *keyword=''*, *quantity=15*, *location=None*, *interval=None*)

Validate and prepare parameters for use in Twitter data retrieval.

See also:

`ogre.validation.validate()` describes the format each parameter must have.

Parameters

- **keys** (*dict*) – Specify Twitter API keys. Twitter **requires** a "consumer_key" and "access_token".
- **media** (*tuple*) – Specify content mediums to make lowercase and deduplicate. "image" and "text" are supported mediums.
- **keyword** (*str*) – Specify search criteria to incorporate the requested media in.
- **quantity** (*int*) – Specify a quota of results.
- **location** (*tuple*) – Specify a location to format as a Twitter geocode ("<latitude>,<longitude>,<radius><unit>").
- **interval** (*tuple*) – Specify earliest and latest moments to convert to Twitter Snowflake IDs.

Raises ValueError

Return type tuple

Returns Each passed parameter is returned (in order) in the proper format.

`ogre.Twitter.twitter`(*keys*, *media=('image', 'text')*, *keyword=''*, *quantity=15*, *location=None*, *interval=None*, ***kwargs*)

Fetch Tweets from the Twitter API.

See also:

`sanitize_twitter()` describes more about the format each parameter must have.

Parameters

- **keys** (*dict*) – Specify an API key and access token.
- **media** (*tuple*) – Specify content mediums to fetch. "text" or "image" are supported mediums.
- **keyword** (*str*) – Specify search criteria. "Queries can be limited due to complexity." If this happens, no results will be returned. To avoid this, follow Twitter Best Practices including the following: "Limit your searches to 10 keywords and operators."
- **quantity** (*int*) – Specify a quota of results to fetch. Twitter will return 15 results by default, and up to 100 can be requested in a single query. If a number larger than 100 is specified, the retriever will make multiple queries in an attempt to satisfy the requested *quantity*, but this is done on a best effort basis. Whether the specified number is returned or not depends on Twitter.
- **location** (*tuple*) – Specify a place (latitude, longitude, radius, unit) to search. Since OGRe only returns geotagged results, the larger the specified radius, the fewer results will be returned. This is because of the way Twitter satisfies geocoded queries. It uses so-called "fuzzy matching logic" to deduce the location of Tweets posted publicly without location data. OGRe filters these out.
- **interval** (*tuple*) – Specify a period of time (earliest, latest) to search. "The Search API is not complete index of all Tweets, but instead an index of recent Tweets." Twitter's definition of "recent" is rather vague, but when an interval is not specified, "that index includes between 6-9 days of Tweets."
- **strict_media** (*bool*) – Specify whether to only return the requested media (defaults to False). Setting this to False helps build caches faster at no additional cost. For instance, since Twitter automatically sends the text of a Tweet back, if *("image",)* is passed for *media*, the text on hand will only be filtered if *strict_media* is True.
- **secure** (*bool*) – Specify whether to prefer HTTPS or not (defaults to True).
- **test** (*bool*) – Specify whether a the current request is a trial run. This affects what gets logged and should be accompanied by the next 3 parameters (*test_message*, *api*, and *network*).
- **test_message** (*str*) – Specify a description of the test to log. This is ignored if the *test* parameter is False.
- **api** (*callable*) – Specify API access point (for dependency injection).
- **network** (*callable*) – Specify a network access point (for dependency injection).

Raises OGReError, OGReLimitError, TwythonError

Return type list

Returns GeoJSON Feature(s)

See also:

Visit https://dev.twitter.com/docs/using-search for tips on how to build queries for Twitter using the *keyword* parameter. More information may also be found at https://dev.twitter.com/docs/api/1.1/get/search/tweets.

OGRe Parameter Validator

`validate()` – check OGRe parameters for errors

`sanitize()` – validate and cleanse OGRe parameters

`ogre.validation.`**`sanitize`**(*media=('image', 'sound', 'text', 'video'), keyword='', quantity=15, location=None, interval=None*)

Validate and transform input to expected types.

See also:

`validate()` describes the format each parameter must have.

Parameters

- **`media`** (`tuple`) – Specify content mediums to make lowercase and deduplicate.
- **`keyword`** (`str`) – Specify search criteria.
- **`quantity`** (`int`) – Specify a quota of results.
- **`location`** (`tuple`) – Specify a location to make numeric (latitude, longitude, radius) and lowercase (unit).
- **`interval`** (`tuple`) – Specify earliest and latest moments to make numeric and sort in ascending order.

Return type tuple

Returns sanitized parameters (media, keyword, quantity, location, interval)

`ogre.validation.`**`validate`**(*media=('image', 'sound', 'text', 'video'), keyword='', quantity=15, location=None, interval=None*)

Check common interface parameters for errors and validity.

Parameters

- **`media`** (`tuple`) – "image", "sound", "text", and "video" are valid mediums.
- **`keyword`** (`str`) – Valid criteria varies by source.
- **`quantity`** (`int`) – Specify a positive quota of desired results.
- **`location`** (`tuple`) – Specify a location (latitude, longitude, radius, unit) composed of 3 numbers and a string, respectively. "km" and "mi" are supported units.
- **`interval`** (`tuple`) – Specify a period of time (earliest, latest) composed of 2 POSIX timestamps (positive numbers). The order of earliest/latest moments does not matter as the lower number will be considered earliest.

Raises ValueError

1.4 License

OGRe is released under the GNU Lesser General Public License.

See also:

GNU LGPLv2+ in Plain English

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

O

ogre.api, 7
ogre.Twitter, 8
ogre.validation, 10

F

fetch() (ogre.api.OGRe method), 7

G

get() (ogre.api.OGRe method), 8

O

OGRe (class in ogre.api), 7
ogre.api (module), 7
ogre.Twitter (module), 8
ogre.validation (module), 10

S

sanitize() (in module ogre.validation), 10
sanitize_twitter() (in module ogre.Twitter), 8

T

twitter() (in module ogre.Twitter), 9

V

validate() (in module ogre.validation), 10