TwitterSearch Documentation

Release 0.75

Christian Koepp

Jan 01, 2018

Contents

TwitterSearch was (and still is) developed as part of a project about social media at the Carl von Linde-Akademie which is part of the Technische Universität München. Thus it is a data collecting toolkit and **not implementing** the whole Twitter API but the Search API.

It's using the REST API in **version 1.1** only. In it's recent version it directly uses IDs of tweets to navigate throughout the available tweets instead of pages, which is way more comfortable to use and more convenient for really getting all possible tweets.

Also *TwitterSearch* is build to be highly flexible in its usage making it usable even within exotic use-cases. Details about non-default use-cases can be found in the *Advanced usage* sections within the class articles.

CHAPTER 1

Architecture

TwitterSearch consists of three classes: TwitterSearch, TwitterSearchOrder and TwitterSearchException.

CHAPTER 2

Table of contents

2.1 Basic usage

In most cases you probably just like to iterate through all available tweets as easy as possible. And there it is, a very minimal example to do exactly this:

```
from TwitterSearch import *

try:
    tso = TwitterSearchOrder()
    tso.setKeywords(['#Hashtag1', '#Hashtag2'])

    ts = TwitterSearch(
            consumer_key = 'aaabbb',
            consumer_secret = 'cccddd',
            access_token = '111222',
            access_token_secret = '333444'
        )

    for tweet in ts.searchTweetsIterable(tso):
        print('@%s tweeted: %s' % (tweet['user']['screen_name'], tweet['text']))

    except TwitterSearchException as e: # take care of all those ugly errors if there␣
↪are some
        print(e)
```

Please note that is a working example for both, Python2 **and** Python3.

2.1.1 Accessible information

The creator of this library doesn't like to hide any informations from you. Therefore the data you'll receive is quite a lot. A typical tweet, as mentioned in the section above, consists of a huge `dict`.

You may ask the question "*But what does this field exactly mean?*". Well, that's where the job of *TwitterSearch* ends and the Twitter documentation joins the fun.

An example of how such a tweet looks like is the following dict:

```
       {'contributors': None,
'coordinates': None,
'created_at': 'Tue Jul 02 11:43:18 +0000 2013',
'entities': {'hashtags': [],
             'media': [{'display_url': 'pic.twitter.com/dJLxVZaSW9',
                        'expanded_url': 'http://twitter.com/EarthBeauties/status/
↪351897277473882113/photo/1',
                        'id': 351897277478076417,
                        'id_str': '351897277478076417',
                        'indices': [78, 100],
                        'media_url': 'http://pbs.twimg.com/media/BOIwtZ2CAAEFRXU.jpg',
                        'media_url_https': 'https://pbs.twimg.com/media/
↪BOIwtZ2CAAEFRXU.jpg',
                        'sizes': {'large': {'h': 375,
                                            'resize': 'fit',
                                            'w': 600},
                                  'medium': {'h': 375,
                                             'resize': 'fit',
                                             'w': 600},
                                  'small': {'h': 213,
                                            'resize': 'fit',
                                            'w': 340},
                                  'thumb': {'h': 150,
                                            'resize': 'crop',
                                            'w': 150}},
                        'source_status_id': 351897277473882113,
                        'source_status_id_str': '351897277473882113',
                        'type': 'photo',
                        'url': 'http://t.co/dJLxVZaSW9'}],
             'symbols': [],
             'urls': [],
             'user_mentions': [{'id': 786796010,
                                'id_str': '786796010',
                                'indices': [33, 47],
                                'name': u'Earth Pictures\u2122',
                                'screen_name': 'EarthBeauties'}]},
'favorite_count': 0,
'favorited': False,
'geo': None,
'id': 352029711347617792,
'id_str': '352029711347617792',
'in_reply_to_screen_name': 'EarthBeauties',
'in_reply_to_status_id': 351897277473882113,
'in_reply_to_status_id_str': '351897277473882113',
'in_reply_to_user_id': 786796010,
'in_reply_to_user_id_str': '786796010',
'lang': 'in',
'metadata': {'iso_language_code': 'in', 'result_type': 'recent'},
'place': None,
'possibly_sensitive': False,
'retweet_count': 0,
'retweeted': False,
'source': 'web',
'text': 'mau dong dibangunin rmh kekgini "@EarthBeauties: Hohenzollern Castle,␣
↪Germany http://t.co/dJLxVZaSW9',
```

```
'truncated': False,
'user': {'contributors_enabled': False,
         'created_at': 'Sun Mar 18 04:22:51 +0000 2012',
         'default_profile': False,
         'default_profile_image': False,
         'description': u"girl non-smoking alcohol-free \u2022 @PLAYMAKERKIDSHC␣
↪\u2022 DSFF \u2022 15\u221e \u2022 NotWild'",
         'entities': {'description': {'urls': []},
                      'url': {'urls': [{'display_url': 'instagram.com/giwaang',
                                        'expanded_url': 'http://instagram.com/giwaang
↪',
                                        'indices': [0, 22],
                                        'url': 'http://t.co/vCyfkrdTwa'}]}},
         'favourites_count': 1,
         'follow_request_sent': False,
         'followers_count': 661,
         'following': False,
         'friends_count': 176,
         'geo_enabled': False,
         'id': 528140042,
         'id_str': '528140042',
         'is_translator': False,
         'lang': 'id',
         'listed_count': 1,
         'location': u"SwiekeCity\u2022PinkBabyRoom's",
         'name': 'EarStud',
         'notifications': False,
         'profile_background_color': 'BADFCD',
         'profile_background_image_url': 'http://a0.twimg.com/profile_background_
↪images/872889954/b7439a65d39bdff360c934bd6f33c3b7.jpeg',
         'profile_background_image_url_https': 'https://si0.twimg.com/profile_
↪background_images/872889954/b7439a65d39bdff360c934bd6f33c3b7.jpeg',
         'profile_background_tile': True,
         'profile_banner_url': 'https://pbs.twimg.com/profile_banners/528140042/
↪1369624796',
         'profile_image_url': 'http://a0.twimg.com/profile_images/378800000047155611/
↪7581e79882f1c9f1bbe4b706a023e2c9_normal.jpeg',
         'profile_image_url_https': 'https://si0.twimg.com/profile_images/
↪378800000047155611/7581e79882f1c9f1bbe4b706a023e2c9_normal.jpeg',
         'profile_link_color': 'FF0000',
         'profile_sidebar_border_color': '000000',
         'profile_sidebar_fill_color': '252429',
         'profile_text_color': '666666',
         'profile_use_background_image': True,
         'protected': False,
         'screen_name': 'giwaang',
         'statuses_count': 10199,
         'time_zone': None,
         'url': 'http://t.co/vCyfkrdTwa',
         'utc_offset': None,
         'verified': False}}
```

2.2 The TwitterSearch class

This class contains the actual functionality of this library. It is responsible for correctly transmitting your data to the Twitter API and returning the results to your program afterwards.

If you're looking for the exceptions thrown by this class based on the HTTP status, please read the article about TwitterSearchException.

2.2.1 Available methods

Method	Description	Example
`setProxy(<dict>)`	Sets a proxy. Because only HTTPS requests are done, it's sufficient to only input a dict containing a `https` entry	`setProxy({'https':'10.0.0.1:123'})`
`authenticate(<bool>)`	Creates an authenticated oauth2 client and if `verify` is true, it also checkes if the user credentials are valid. The **default** value is *True*	`authenticate()`, `authenticate(True)`
`searchTweetsIterable(<dict>)`	Queries the Twitter API, iterates through tweets and reloads available older tweets automatically	see Basic usage
`searchTweets(<dict>)`	Queries the Twitter API **without** iterating or reloading of further results and returns response	see *Advanced usage*
`sentSearch(<string>)`	Queries the Twitter Search API with a given query string, determines if there are more results available in API and returns response.	`sentSearch('?q=One+Two&count=100')`
`searchNextResults()`	Queries the API for more tweets and returns response	see *Advanced usage*
`getMetadata()`	Returns a dict of meta information about the last query	see *Advanced usage*
`getTweets()`	Returns a dict of all tweets returned by last query	see *Advanced usage*
`getStatistics()`	Returns a dict of the type `{ 'queries' : <int>, 'tweets' : <int> }` with statistical values about the number of queries and the sum of all tweets recieved by this very instance of *TwitterSearch*	`print "Queries done %i / Tweets received %i" % (ts.getStatistics()['queries'], ts.getStatistics()['tweets'])`
`checkHTTPStatus(<int>)`	Checks if given HTTP status code is in `TwitterSearch.exceptions` and raises `TwitterSearchException` if this is the case	`checkHTTPStatus(200)`, `checkHTTPStatus(401)`
`setSupportedLanguages(<dict>)`	Loads currently supported languages from Twitter and stores them in a `TwitterSearchOrder` instance	see *Advanced usage*

The methods `next()`, `__next__()` and `__iter__()` are used during the iteration process. For more information about those methods please consult the official Python documentation.

Constructor

The constructor needed to set your credentials for the Twitter API. The parameters are `__init__( consumer_key, consumer_secret, access_token, access_token_secret, verify=True)`.

If you're new to Python take a look at the following example:

```
ts1 = TwitterSearch(
    consumer_key = 'aaabbb',
    consumer_secret = 'cccddd',
    access_token = '111222',
    access_token_secret = '333444'
)

# equals

ts2 = TwitterSearch('aaabbb', 'cccddd', '111222', '333444', verify=True)
```

Authentication and verification

Please be aware that there is **no further check** whether or not your credentials are valid if you set `verify=False` in the constructor. If you're skipping the verification process of *TwitterSearch* you can avoid some traffic and one query (which is also rate-limited by Twitter).

But be aware that you're only saving **one** request at all by avoiding the automatic verification process. Due to the fact that json doesn't consume much traffic at all, this may only be a way for very conservative developers or some exotic scenarios.

Proxy usage

To use a HTTPS proxy at initialization of the *TwitterSearch* class, an addition argument named `proxy={ 'https' : 'some.proxy:888' }` can be used. Otherwise the authentication will fail if the client has no direct access to the Twitter API.

Returned tweets

TwitterSearch is trying to not hide anything from your eyes except the complexity of it's functions. Due to this you're able to get all the information available (which can be quite a lot).

Example output with only one tweet included:

```
{'search_metadata': {'completed_in': 0.08,
                     'count': 1,
                     'max_id': 352072665667878913,
                     'max_id_str': '352072665667878913',
                     'next_results': '?max_id=352072665667878912&q=Germany%20castle&
↪count=1&include_entities=1',
                     'query': 'Germany+castle',
                     'refresh_url': '?since_id=352072665667878913&q=Germany%20castle&
↪include_entities=1',
                     'since_id': 0,
                     'since_id_str': '0'},
                     'statuses': [
                     {'contributors': None,
            'coordinates': None,
            'created_at': 'Tue Jul 02 14:33:59 +0000 2013',
            'entities': {'hashtags': [],
                         'media': [{'display_url': 'pic.twitter.com/Oz77FLEong',
                                    'expanded_url': 'http://twitter.com/ThatsEarth/
↪status/351839174887870464/photo/1',
                                    'id': 351839174896259072,
```

```
                                    'id_str': '351839174896259072',
                                    'indices': [117, 139],
                                    'media_url': 'http://pbs.twimg.com/media/
↪BOH73Y3CEAAldKU.jpg',
                                    'media_url_https': 'https://pbs.twimg.com/media/
↪BOH73Y3CEAAldKU.jpg',
                                    'sizes': {'large': {'h': 639,
                                                        'resize': 'fit',
                                                        'w': 960},
                                              'medium': {'h': 399,
                                                         'resize': 'fit',
                                                         'w': 600},
                                              'small': {'h': 226,
                                                        'resize': 'fit',
                                                        'w': 340},
                                              'thumb': {'h': 150,
                                                        'resize': 'crop',
                                                        'w': 150}},
                                    'source_status_id': 351839174887870464,
                                    'source_status_id_str': '351839174887870464',
                                    'type': 'photo',
                                    'url': 'http://t.co/Oz77FLEong'}],
                       'symbols': [],
                       'urls': [],
                       'user_mentions': [{'id': 118504288,
                                          'id_str': '118504288',
                                          'indices': [0, 11],
                                          'name': 'Josh Dallas',
                                          'screen_name': 'joshdallas'},
                                         {'id': 298250825,
                                          'id_str': '298250825',
                                          'indices': [12, 25],
                                          'name': 'Ginnifer Goodwin',
                                          'screen_name': 'ginnygoodwin'},
                                         {'id': 1201661238,
                                          'id_str': '1201661238',
                                          'indices': [49, 60],
                                          'name': 'Earth Pics',
                                          'screen_name': 'ThatsEarth'}]},
           'favorite_count': 0,
           'favorited': False,
           'geo': None,
           'id': 352072665667878913,
           'id_str': '352072665667878913',
           'in_reply_to_screen_name': 'joshdallas',
           'in_reply_to_status_id': None,
           'in_reply_to_status_id_str': None,
           'in_reply_to_user_id': 118504288,
           'in_reply_to_user_id_str': '118504288',
           'lang': 'en',
           'metadata': {'iso_language_code': 'en',
                        'result_type': 'recent'},
           'place': None,
           'possibly_sensitive': False,
           'retweet_count': 0,
           'retweeted': False,
           'source': '<a href="http://twitter.com/download/android" rel="nofollow">
↪Twitter for Android</a>',
```

```
          'text': '@joshdallas @ginnygoodwin home during wintertime"@ThatsEarth:␣
↪Hohenzollern Castle floating above the Clouds,Germany. http://t.co/Oz77FLEong"',
          'truncated': False,
          'user': {'contributors_enabled': False,
                   'created_at': 'Fri Aug 14 09:15:27 +0000 2009',
                   'default_profile': False,
                   'default_profile_image': False,
                   'description': 'Scorpio. 23. MBA Graduate.',
                   'entities': {'description': {'urls': []},
                                'url': {'urls': [{'display_url': 'fanfiction.net/u/
↪4764512/',
                                                  'expanded_url': 'http://www.
↪fanfiction.net/u/4764512/',
                                                  'indices': [0,
                                                              22],
                                                  'url': 'http://t.co/sEKQ1M85H2'}]}}
↪,
                   'favourites_count': 114,
                   'follow_request_sent': False,
                   'followers_count': 300,
                   'following': False,
                   'friends_count': 229,
                   'geo_enabled': False,
                   'id': 65599486,
                   'id_str': '65599486',
                   'is_translator': False,
                   'lang': 'en',
                   'listed_count': 0,
                   'location': 'Kuwait',
                   'name': 'Amal Behbehani',
                   'notifications': False,
                   'profile_background_color': 'DBE9ED',
                   'profile_background_image_url': 'http://a0.twimg.com/profile_
↪background_images/317569734/tumblr_lqc4ttwuJm1qclkveo1_500.jpg',
                   'profile_background_image_url_https': 'https://si0.twimg.com/
↪profile_background_images/317569734/tumblr_lqc4ttwuJm1qclkveo1_500.jpg',
                   'profile_background_tile': True,
                   'profile_banner_url': 'https://pbs.twimg.com/profile_banners/
↪65599486/1372576102',
                   'profile_image_url': 'http://a0.twimg.com/profile_images/
↪3763288269/57c274f19592f6d190957d8eb86c64f1_normal.png',
                   'profile_image_url_https': 'https://si0.twimg.com/profile_images/
↪3763288269/57c274f19592f6d190957d8eb86c64f1_normal.png',
                   'profile_link_color': 'CC3366',
                   'profile_sidebar_border_color': 'DBE9ED',
                   'profile_sidebar_fill_color': 'E6F6F9',
                   'profile_text_color': '333333',
                   'profile_use_background_image': True,
                   'protected': False,
                   'screen_name': 'TigeyGirl',
                   'statuses_count': 18891,
                   'time_zone': 'Santiago',
                   'url': 'http://t.co/sEKQ1M85H2',
                   'utc_offset': -14400,
                   'verified': False}}]}
```

Have a look at the entities documented by Twitter to figure out what a specific key-value tuple does exactly mean.

2.2.2 Advanced usage

Sometime the default use-case is not sufficient and you may like to use *TwitterSearch* in unusual scenarios.

Access meta data

An output of the available meta data from the query to the Twitter API is stored in a `dict`. You can access it by calling `getMetadata()` which will return all meta information about the last query.

Example:

```
{
'content-length': '467129',
'x-rate-limit-reset': '1372773784',
'x-rate-limit-remaining': '170',
'x-xss-protection': '1; mode=block',
'cache-control': 'no-cache, no-store, must-revalidate, pre-check=0, post-check=0',
'status': '200',
'transfer-encoding': 'chunked',
'set-cookie': 'lang=de, guest_id=v1%!xxx; Domain=.twitter.com; Path=/; Expires=Thu,␣
↪01-Jul-2013 14:02:32 UTC',
'expires': 'Tue, 31 Mar 1981 05:00:00 GMT',
'x-access-level': 'read',
'last-modified': 'Tue, 01 Jul 2013 14:02:32 GMT',
'-content-encoding': 'gzip',
'pragma': 'no-cache',
'date': 'Tue, 01 Jul 2013 14:02:32 GMT',
'x-rate-limit-limit': '180',
'content-location': u'https://api.twitter.com/1.1/search/tweets.json?count=100&oauth_
↪body_hash=xxx&oauth_nonce=xxx&oauth_timestamp=xxx&oauth_consumer_key=xxx&oauth_
↪signature_method=HMAC-SHA1&q=Germany+castle&oauth_version=1.0&oauth_token=xxx&oauth_
↪signature=xxx',
'x-transaction': 'xxx',
'strict-transport-security': 'max-age=631138519',
'server': 'tfe',
'x-frame-options': 'SAMEORIGIN',
'content-type': 'application/json;charset=utf-8'
}
```

Be **careful** about those data as it contains sensible data as you can see in `getMetadata()['content-location']`. Do **NOT** save or output those information to insecure environments!

TwitterSearch without iteration

It is also perfectly possible to use *TwitterSearch* without the iteration and to query the Twitter API all by yourself. For example you may like to implement the suggest max_id procedure of Twitter to access the API directly and don't trust the library to do this automatically on it's own.

A possible solution could look like this:

```
from TwitterSearch import *

try:
    tso = TwitterSearchOrder()
    tso.setKeywords(['Germany', 'castle'])
```

```
    ts = TwitterSearch('aaabbb', 'cccddd', '111222', '333444')

    # init variables needed in loop
    todo = True
    next_max_id = 0

    # let's start the action
    while(todo):

        # first query the Twitter API
        response = ts.searchTweets(tso)

        # print rate limiting status
        print "Current rate-limiting status: %i" % ts.getMetadata()['x-rate-limit-
↪reset']

        # check if there are statuses returned and whether we still have work to do
        todo = not len(response['content']['statuses']) == 0

        # check all tweets according to their ID
        for tweet in response['content']['statuses']:
            tweet_id = tweet['id']
            print("Seen tweet with ID %i" % tweet_id)

            # current ID is lower than current next_max_id?
            if (tweet_id < next_max_id) or (next_max_id == 0):
                next_max_id = tweet_id
                next_max_id -= 1 # decrement to avoid seeing this tweet again

        # set lowest ID as MaxID
        tso.setMaxID(next_max_id)

except TwitterSearchException as e:
    print(e)
```

On-the-fly loading of supported languages

As you may have figured out some languages are not supported by Twitter and those that are may change over time. This is why Twitter does provide an endpoint to load all currently supported languages. You may query it to gather current information about the languages in Twitter.

```
from TwitterSearch import *

try:
    tso = TwitterSearchOrder()
    ts = TwitterSearch('aaabbb', 'cccddd', '111222', '333444')

    # load  currently supported languages by Twitter and store them in a␣
↪TwitterSearchOrder object
    ts.setSupportedLanguages(tso)

    # try to set German (see ISO 639-1) as language
    ts.setLanguage('de')
    print('German seems to be officially supported by Twitter. Yay!')
```

```
except TwitterSearchException as e:

    # if we get an 1002 code it means that 'de' is not supported (see␣
↪TwitterSearchException)
    if e.code == 1002:
        print('Oh no - German is not supported :(')
    print(e)
```

2.3 The TwitterSearchOrder class

This class mainly acts as a plain container for configuration all parameters currently available by the Twitter Search API.

If you're looking for the exceptions this class raises, please have a look at the article about TwitterSearchException.

2.3.1 Available methods

There are several parameters which can easily be set and modified by methods in *TwitterSearchOrder*.

The only parameter with a default value is `count` with *100*. This is because it is the maximum of tweets returned by the Twitter API and in most cases you'd like to reduce traffic and the amount of queries, so it makes sense to set the biggest possible value by default.

Be aware that some parameters *can be* ignored by Twitter. For example currently not every language is detectable by the Search API. TwitterSearch is only responsible for transmitting values according to the Twitter documentation.

API Parameter	Type	Modifying methods	Example
q	***required***	`addKeyword(<string>)`, `setKeywords(<list>)`	`addKeyword('#Hashtag')`, `setKeywords(['foo', 'bar'])`
geocode	*optional*	`setGeocode(latitude<float>, longitude<float>, radius<int,long>, km=<True,False>`	`setGeocode(52.5233,13.4127,10,km=True)`
lang	*optional*	`setLanguage(<ISO-6391-string>)`	`setLanguage('en')`
locale	*optional*	`setLocale(<ISO-6391-string>)`	`setLocale('ja')`
result_type	*optional*	`setResultType(<mixed,recent,popular>)`	`setResultType('recent')`
count	*optional*	`setCount(<int>[Range: 1-100])`	`setCount(42)`
until	*optional*	`setUntil(<datetime.date>)`	`setUntil(datetime.date(2012, 12, 24))`
since_id	*optional*	`setSinceID(<int,long>[Range: >0])`	`setSinceID(250075927172759552)`
max_id	*optional*	`setMaxID(<int,long>[Range: >0])`	`setMaxID(249292149810667520)`
include_entities	*optional*	`setIncludeEntities(<bool,int>)`	`setIncludeEntities(True)`, `setIncludeEntities(1)`
callback	*optional*	`setCallback(<string>)`	`setCallback('myMethod')`

If you're not familar with the meaning of the parameters, please have a look at the Twitter Search API documentation. Most parameter are self-descriping anyway.

2.3.2 Advanced usage

You may want to use *TwitterSearchOrder* for just generating a valid Twitter Search API query string containing all your arguments without knowing too much details about the Twitter API? No problem at all as there is the method `TwitterSearchOrder.createSearchURL()`. It creates and returns an valid Twitter Search API query string. Afterwards the last created string is also available through `TwitterSearchOrder.url`.

```
from TwitterSearch import TwitterSearchOrder, TwitterSearchException

try:
    tso = TwitterSearchOrder()
    tso.setLanguage('nl')
    tso.setLocale('ja')
    tso.setKeywords(['One','Two'])
    tso.addKeyword('myKeyword')

    print(tso.createSearchURL())

except TwitterSearchException as e:
        print(e)
```

You'll receive `?q=One+Two+myKeyword&count=100&lang=nl&locale=ja` as result. Now you are free to

use this string for manually querying Twitter (or any other API using the same parameter as Twitter does).

Maybe you would like to create another *TwitterSearchOrder* instance with a slightly other URL.

```
from TwitterSearch import TwitterSearchOrder, TwitterSearchException

try:
    tso = TwitterSearchOrder()
    tso.setLanguage('nl')
    tso.setLocale('ja')
    tso.setKeywords(['One','Two'])
    tso.addKeyword('myKeyword')

    querystr = tso.createSearchURL()

    # create a new TwitterSearchOrder based on the old query string and work with it
    tso2 = TwitterSearchOrder()
    tso2.setSearchURL(querystr + '&result_type=mixed&include_entities=true')
    tso2.setLocale('en')
    print(tso2.createSearchURL())

except TwitterSearchException as e:
 print(e)
```

This piece of code will finally result in an output of `?q=One+Two+myKeyword&count=100&lang=nl&locale=en&result_t`

Please be aware that the sense of arguments given by `setSearchURL()` is not checked. Due to this it is perfectly valid to to stuff like `setSearchURL('q=Not+my+department&count=1731&locale=Canada&foo=bar')`. When manually setting the string, the leading `?` sign is optional.

Such stuff doesn't make much sense when querying Twitter. However, there may be cases when you're using TwitterSearch is some exotic context where this behavior is needed to avoid the regular checks of the *TwitterSearchOrder* methods.

Be aware that if you're using `setSearchURL()` all previous configured parameters are lost.

2.4 The TwitterSearchException class

This class is all about exceptions (surprise, surprise!). All exception based directly on TwitterSearch will consist of a **code** and a **message** describing the reason of the exception shortly.

You can also print an exception like a string which will result in `Error <TwitterSearchException.code>: <TwitterSearchException.message>`. For those new to Python, this can be easily done like this:

```
except TwitterSearchException as e:
    print(e)
```

2.4.1 List of exceptions

There are *two* kinds of exceptions. Those based on the HTTP status of the query to the Twitter API and those based on misconfiguration of TwitterSearch for example by setting odd parameters or trying to access tweets without querying the API before.

Library based exceptions

All exceptions based on issues within TwitterSearch do have `TwitterSearchException.code >= 1000`

Code	Message	Raised by class
1000	Neither a list nor a string	TwitterSearchOrder
1001	Not a list object	TwitterSearchOrder
1002	No ISO 6391-1 language code	TwitterSearchOrder
1003	No valid result type	TwitterSearchOrder
1004	Invalid number	TwitterSearchOrder
1005	Invalid unit	TwitterSearchOrder
1006	Invalid callback string	TwitterSearchOrder
1007	Not a date object	TwitterSearchOrder
1008	Invalid boolean	TwitterSearchOrder
1009	Invalid string	TwitterSearch
1010	Not a valid TwitterSearchOrder object	TwitterSearch
1011	No more results available	TwitterSearch
1012	No meta data available	TwitterSearch
1013	No tweets available	TwitterSearch
1014	No results available	TwitterSearch
1015	No keywords given	TwitterSearchOrder
1016	Invalid dict	TwitterSearch

HTTP based exceptions

Exceptions based on the HTTP status response of the Twitter API are `TwitterSearchException.code < 1000`. Note that the `code` attribute is exactly the HTTP status value returned to TwitterSearch. All those exceptions are raised in TwitterSearch only.

Code	Message
400	Bad Request: The request was invalid
401	Unauthorized: Authentication credentials were missing or incorrect
403	Forbidden: The request is understood, but it has been refused or access is not allowed
404	Not Found: The URI requested is invalid or the resource requested does not exists
406	Not Acceptable: Invalid format is specified in the request
410	Gone: This resource is gone
420	Enhance Your Calm: You are being rate limited
422	Unprocessable Entity: Image unable to be processed
429	Too Many Requests: Request cannot be served due to the application's rate limit having been exhausted for the resource
500	Internal Server Error: Something is broken
502	Bad Gateway: Twitter is down or being upgraded
503	Service Unavailable: The Twitter servers are up, but overloaded with requests
504	Gateway timeout: The request couldn't be serviced due to some failure within our stack

2.4.2 Advanced usage

Maybe there is an odd reason why you don't want TwitterSearch to raise an exception when a 404 HTTP status is returned by Twitter. Additional you'd like to raise an exception when a 200 HTTP status is returned. Maybe you

would like to test your firewall by doing complex HTTP queries. Okay, don't ask me about use-cases, let's just assume there is some strange reason to do so.

Since TwitterSearch is designed to be used in academic and highly individual scenarios it is perfectly possible to do such crazy stuff without much trouble.

```
from TwitterSearch import *

tso = TwitterSearchOrder()
tso.setKeywords(['strange', 'use-case'])
tso.setIncludeEntities(False)

ts = TwitterSearch(
    consumer_key = 'onetwothree',
    consumer_secret = 'fourfivesix',
    access_token = 'foo',
    access_token_secret = 'bar'
)

# add a HTTP status based exception based on status 200
ts.exceptions.update({200 : 'It worked - damn it!' })

# delete exception based on HTTP status 400
del ts.exceptions[400]

try:
    ts.authenticate()
    for tweet in ts.searchTweetsIterable(tso):
        print("Seen tweed with ID %i" % tweet['id'])

except TwitterSearchException as e:
    if e.code < 1000:
        print("HTTP status based exception: %i - %s" % (e.code, e.message))
    else:
print("Regular exception: %i - %s" % (e.code, e.message))
```

If your credentials are correct you will receive the output `HTTP status based exception: 200 - It worked - damn it!`

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

CHAPTER 4

Contribution

Feel free to open issues, submit code or fork.