TwitterSearch Documentation

Release 1.0.0

Christian Koepp

June 05, 2015

Contents

TwitterSearch was developed as part of a project about social media at the Carl von Linde-Akademie, an institution of the Technische Universität München. Thus, *TwitterSearch* is a data collecting toolkit and is **not implementing** the whole Twitter API but the Search API and the User Timeline API. The library is fully accessible through the official repository at github and maintained by Christian Koepp.

TwitterSearch is using the REST API in **version 1.1** only. In its recent version, the library is using identifiers of tweets to navigate throughout the available list of tweets. Doing so enables a more flexible and efficient iteration than the traditional method of using pages. 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 of this documentation.

All classes and their methods are tested against the latest Python 2 and Python 3 versions automatically. The current state of all branches is visible through Travis CI. Additionally, you should note that with version 1.0 and upwards PEP-8 compatibility is enforced. Those checks were done by running the *pep8* toolkit. If you're interested in contributing to *TwitterSearch* make sure your patches are PEP-8 compatible. Additionally, patches are required to not break the current test cases and to bring test cases with them to test new features and functionalities.

The history of commits and changes of this library can be either accessed by using the official github repository . Also, a summary of the major changes is provided within the `CHANGELOG.rst` file of the package.

Warning: If you're upgrading from a version < 1.0.0 be aware that the API changed! To support PEP-8 completely, former methods named `someMethod()` were renamed to `some_method()`. Apart from this issue, four other API changes were introduced with version 1.0.0:

- simplified proxy functionality: no usage of dicts but plain strings as only HTTPS proxies can be supported
- simplified geo-code parameter: `TwitterSearchOrder.set_geocode(...,metric=True)` renamed to `set_geocode(...,imperial_metric=True)`
- simplified `TwitterSearch.get_statistics()`: switched from dict to tuple style (`{'queries':<int>, 'tweets':<int>}` to `(<int>,<int>)`)
- additional feature: timelines of users can now be accessed using the new class `TwitterUserOrder`

In total those changes can be done quickly without browsing this documentation. If you are not able to do those changes just keep using the versions < 1.0.0. Those will stay available through pypi and therefore will be installable in the future using the common installation methods.

CHAPTER 1

Table of contents

1.1 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.set_keywords(['#Hashtag1', '#Hashtag2'])

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

    for tweet in ts.search_tweets_iterable(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)
```

If you're into the access of a timeline of a certain user, you can do this by using the same pattern:

```
from TwitterSearch import *

try:
    # create a TwitterUserOrder for user named 'NeinQuarterly'
    tuo = TwitterUserOrder('NeinQuarterly') # is equal to TwitterUserOrder(458966079)

    # it's about time to create TwitterSearch object again
    ts = TwitterSearch(
        consumer_key = 'aaabbb',
        consumer_secret = 'cccddd',
        access_token = '111222',
        access_token_secret = '333444'
    )

    # start asking Twitter about the timeline
    for tweet in ts.search_tweets_iterable(tuo):
```

```
        print('@%s tweeted: %s' % (tweet['user']['screen_name'], tweet['text']))

except TwitterSearchException as e: # catch all those ugly errors
    print(e)
```

Please note those code snippets are already working examples executable in both, Python2 **and** Python3.

1.1.1 Accessible information

Note: The Twitter Search API does not reveal tweets older than a week (or sometimes dating back to 10 days). Be aware that the official Twitter Search does not use the Twitter Search API but a internal Twitter interface. Thus, it reveals tweets older than those you can collect through the public API. If you need access to old tweets you might have pay a commercial provider to give you access to its Twitter archives.

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/p
                        '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/d
'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
         '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/b74
         'profile_background_image_url_https': 'https://si0.twimg.com/profile_background_images/87288
         'profile_background_tile': True,
         'profile_banner_url': 'https://pbs.twimg.com/profile_banners/528140042/1369624796',
         'profile_image_url': 'http://a0.twimg.com/profile_images/378800000047155611/7581e79882f1c9f1
         'profile_image_url_https': 'https://si0.twimg.com/profile_images/378800000047155611/7581e798
         '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}}
```

1.1.2 Architecture

TwitterSearch consists of four classes: TwitterSearch, TwitterSearchOrder, TwitterUserOrder and TwitterSearchException.

To not repeat certain code-fragments the class TwitterOrder is also available. However, this class is rarely used directly and only contains few basic methods.

1.2 TwitterSearch

1.2.1 TwitterSearch package

Submodules

TwitterSearch.TwitterOrder module

class TwitterSearch.TwitterOrder.**TwitterOrder**
Bases: object

Basic interface class to inherit from. Methods raising NotImplementedError exceptions need to be implemented by all children

arguments = {}

create_search_url()
Generates an url-encoded query string from stored key-values tuples. Has to be implemented within child classes

Raises NotImplementedError

set_count(*cnt*)
Sets 'count' parameter used to define the number of tweets to return per page. Maximum and default value is 100

Parameters **cnt** – Integer containing the number of tweets per page within a range of 1 to 100

Raises TwitterSearchException

set_include_entities(*include*)
Sets 'include entities' parameter to either include or exclude the entities node within the results

Parameters **include** – Boolean to trigger the 'include entities' parameter

Raises TwitterSearchException

set_max_id(*twid*)
Sets 'max_id' parameter used to return only results with an ID less than (that is, older than) or equal to the specified ID

Parameters **twid** – A valid tweet ID in either long (Py2k) or integer (Py2k + Py3k) format

Raises TwitterSearchException

set_search_url(*url*)
Reads given query string and stores key-value tuples. Has to be implemented within child classes

Parameters **url** – A string containing the twitter API endpoint URL

Raises NotImplementedError

set_since_id(*twid*)
Sets 'since_id' parameter used to return only results with an ID greater than (that is, more recent than) the specified ID

Parameters **twid** – A valid tweet ID in either long (Py2k) or integer (Py2k + Py3k) format

Raises TwitterSearchException

TwitterSearch.TwitterSearch module

class TwitterSearch.TwitterSearch.**TwitterSearch**(*consumer_key*, *consumer_secret*, *access_token*, *access_token_secret*, ***attr*)
Bases: object

This class contains the actual functionality of this library. It is responsible for correctly transmitting your data to the Twitter API (v1.1 only) and returning the results to your program afterwards. It is configured using an implementation of *TwitterOrder* along with valid Twitter credentials. Currently two different implementations are usable: *TwitterUserOrder* for retrieving the timeline of a certain user and *TwitterSearchOrder* for accessing the Twitter Search API.

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

authenticate(*verify=True*)
Creates an authenticated and internal oauth2 handler needed for queries to Twitter and verifies credentials if needed. If verify is true, it also checks if the user credentials are valid. The **default** value is *True*

Parameters **verify** – boolean variable to directly check. Default value is True

check_http_status(*http_status*)
Checks if given HTTP status code is within the list at TwitterSearch.exceptions and raises a TwitterSearchException if this is the case. Example usage: checkHTTPStatus(200) and checkHTTPStatus(401)

Parameters **http_status** – Integer value of the HTTP status of the last query. Invalid statuses will raise an exception.

Raises TwitterSearchException

exceptions = {420: 'Enhance Your Calm: You are being rate limited', 502: 'Bad Gateway: Twitter is down or being up

get_amount_of_tweets()
Returns current amount of tweets available within this instance

Returns The amount of tweets currently available

Raises TwitterSearchException

get_metadata()
Returns all available meta data collected during last query. See Advanced usage for example

Returns Available meta information about the last query in form of a dict

Raises TwitterSearchException

get_minimal_id()
Returns the minimal tweet ID of the current response

Returns minimal tweet identification number

Raises TwitterSearchException

get_proxy()
Returns the current proxy url or None if no proxy is set

Returns A string containing the current HTTPS proxy (e.g. `my.proxy.com:8080`) or `None` is no proxy is used

get_statistics()
Returns dict with statistical information about amount of queries and received tweets. Returns statistical values about the number of queries and the sum of all tweets received by this very instance of *TwitterSearch*. Example usage: `print("Queries done: %i. Tweets received: %i" % ts.get_statistics())`

Returns A `tuple` with `queries` and `tweets` keys containing integers. E.g. `(1,100)` which stands for one query that contained one hundred tweets.

get_tweets()
Returns all available data from last query. See Advanced usage for example

Returns All tweets found using the last query as a `dict`

Raises TwitterSearchException

next()
Python2 comparability method. Simply returns `self.__next__()`

Returns the `__next__()` method of this class

search_next_results()
Triggers the search for more results using the Twitter API. Raises exception if no further results can be found. See Advanced usage for example

Returns `True` if there are more results available within the Twitter Search API

Raises TwitterSearchException

search_tweets(*order*)
Creates an query string through a given TwitterSearchOrder instance and takes care that it is send to the Twitter API. This method queries the Twitter API **without** iterating or reloading of further results and returns response. See Advanced usage for example

Parameters **order** – A TwitterOrder instance. Can be either TwitterSearchOrder or TwitterUserOrder

Returns Unmodified response as `dict`.

Raises TwitterSearchException

search_tweets_iterable(*order*, *callback=None*)
Returns itself and queries the Twitter API. Is called when using an instance of this class as iterable. See Basic usage for examples

Parameters

- **order** – An instance of TwitterOrder class (e.g. TwitterSearchOrder or TwitterUserOrder)
- **callback** – Function to be called after a new page is queried from the Twitter API

Returns Itself using `self` keyword

send_search(*url*)
Queries the Twitter API with a given query string and stores the results internally. Also validates returned HTTP status code and throws an exception in case of invalid HTTP states. Example usage `sendSearch('?q=One+Two&count=100')`

Parameters **url** – A string of the URL to send the query to

Raises TwitterSearchException

set_proxy(*proxy*)
Sets a HTTPS proxy to query the Twitter API

Parameters **proxy** – A string of containing a HTTPS proxy e.g. `set_proxy("my.proxy.com:8080")`.

Raises TwitterSearchException

set_supported_languages(*order*)
Loads currently supported languages from Twitter API and sets them in a given TwitterSearchOrder instance. See Advanced usage for example

Parameters **order** – A TwitterOrder instance. Can be either TwitterSearchOrder or TwitterUserOrder

TwitterSearch.TwitterSearchException module

exception TwitterSearch.TwitterSearchException.**TwitterSearchException**(*code*, *msg=None*)

Bases: Exception

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.

TwitterSearch.TwitterSearchOrder module

class TwitterSearch.TwitterSearchOrder.**TwitterSearchOrder**

Bases: *TwitterSearch.TwitterOrder.TwitterOrder*

This class is for configurating all available arguments of the Twitter Search API (v1.1). It also creates valid query strings which can be used in other environments identical to the syntax of the Twitter Search API.

add_keyword(*word*, *or_operator=False*)
Adds a given string or list to the current keyword list

Parameters

- **word** – String or list of at least 2 character long keyword(s)
- **or_operator** – Boolean. Concatenates all elements of parameter word with `OR`. Is ignored is word is not a list. Thus it is possible to search for `foo OR bar`. Default value is False which corresponds to a search of `foo AND bar`.

Raises TwitterSearchException

create_search_url()
Generates (urlencoded) query string from stored key-values tuples

Returns A string containing all arguments in a url-encoded format

iso_6391 = ('aa', 'ab', 'ae', 'af', 'ak', 'am', 'an', 'ar', 'as', 'av', 'ay', 'az', 'ba', 'be', 'bg', 'bh', 'bi', 'bm', 'bn', 'bo', 'br'

remove_all_filters()
Removes all filters

remove_attitude_filter()
Remove attitude filter

remove_link_filter()
Remove the current link filter

remove_question_filter()
Remove the current question filter

remove_source_filter()
Remove the current source filter

set_callback(*func*)
Sets 'callback' parameter. If supplied, the response will use the JSONP format with a callback of the given name

Parameters **func** – A string containing the name of the callback function

Raises TwitterSearchException

set_geocode(*latitude*, *longitude*, *radius*, *imperial_metric=True*)
Sets geolocation parameters to return only tweets by users located within a given radius of the given latitude/longitude. The location is preferentially taking from the Geotagging API, but will fall back to their Twitter profile.

Parameters

- **latitude** – A integer or long describing the latitude
- **longitude** – A integer or long describing the longitude
- **radius** – A integer or long describing the radius
- **imperial_metric** – Whether the radius is given in metric (kilometers) or imperial (miles) system. Default is `True` which relates to usage of the imperial kilometer metric

Raises TwitterSearchException

set_keywords(*words*, *or_operator=False*)
Sets a given list as the new keyword list

Parameters

- **words** – A list of at least 2 character long new keywords
- **or_operator** – Boolean. Concatenates all elements of parameter word with `OR`. Enables searches for `foo OR bar`. Default value is False which corresponds to a search of `foo AND bar`.

Raises TwitterSearchException

set_language(*lang*)
Sets 'lang' parameter used to only fetch tweets within a certain language

Parameters **lang** – A 2-letter language code string (ISO 6391 compatible)

Raises TwitterSearchException

set_link_filter()
Only search for tweets including links

set_locale(*lang*)
Sets 'locale' parameter to specify the language of the query you are sending (only ja is currently effective)

Parameters **lang** – A 2-letter language code string (ISO 6391 compatible)

Raises TwitterSearchException

set_negative_attitude_filter()
Only search for tweets with negative attitude

set_positive_attitude_filter()
Only search for tweets with positive attitude

set_question_filter()
Only search for tweets asking a question

set_result_type(*result_type*)
Sets 'result_type' parameter to specify what type of search results you would prefer to receive. The current default is "mixed." Valid values include: - mixed: Include both popular and real time results - recent: return only the most recent results - popular: return only the most popular results :param result_type: A string containing one of the three valid result types

Raises TwitterSearchException

set_search_url(*url*)
Reads given query string and stores key-value tuples

Parameters **url** – A string containing a valid URL to parse arguments from

set_source_filter(*source*)
Only search for tweets entered via given source

Parameters **source** – String. Name of the source to search for. An example would be `source=twitterfeed` for tweets submitted via TwitterFeed

Raises TwitterSearchException

set_until(*date*)
Sets 'until' parameter used to return only tweets generated before the given date

Parameters **date** – A datetime instance

Raises TwitterSearchException

TwitterSearch.TwitterUserOrder module

class TwitterSearch.TwitterUserOrder.**TwitterUserOrder**(*user*)
Bases: *TwitterSearch.TwitterOrder.TwitterOrder*

This class configures all arguments available of the user_timeline endpoint of the Twitter API (version 1.1 only). It also creates a valid query string out of the current configuration.

create_search_url()
Generates (urlencoded) query string from stored key-values tuples

Returns A string containing all arguments in a url-encoded format

set_contributor_details(*contdetails*)
Sets 'contributor_details' parameter used to enhance the contributors element of the status response to include the screen_name of the contributor. By default only the user_id of the contributor is included

Parameters **contdetails** – Boolean triggering the usage of the parameter

Raises TwitterSearchException

set_exclude_replies(*exclude*)
Sets 'exclude_replies' parameter used to prevent replies from appearing in the returned timeline

Parameters **exclude** – Boolean triggering the usage of the parameter

Raises TwitterSearchException

set_include_rts(*rts*)
Sets 'include_rts' parameter. When set to False, the timeline will strip any native retweets from the returned timeline

Parameters **rts** – Boolean triggering the usage of the parameter

Raises TwitterSearchException

set_search_url(*url*)
Reads given query string and stores key-value tuples

Parameters **url** – A string containing a valid URL to parse arguments from

set_trim_user(*trim*)
Sets 'trim_user' parameter. When set to True, each tweet returned in a timeline will include a user object including only the status authors numerical ID

Parameters **trim** – Boolean triggering the usage of the parameter

Raises TwitterSearchException

TwitterSearch.utils module

Module contents

1.3 Advanced usage: The `TwitterSearch` class

This is the main class of this library where all the action takes place. There are many ways to use it and the most common ones are explained in this section.

1.3.1 Constructor of `TwitterSearch`

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, proxy=None)
```

1.3.2 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. Note that this validation query is part of the rate-limiting as done by Twitter. If you are sure your credentials are correct you can disable this feature.

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.

1.3.3 Proxy usage

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

1.3.4 Avoid rate-limitation using a callback method

Sometimes there is the need to build in certain delays in order to avoid being rate-limited by Twitter. One way to add an artificial delay to your queries is to use the build-in module `time` of Python in combination with a callback method. The following example demonstrates how to use the `callback` argument of the `TwitterSearch.search_tweets_iterable()` method properly. In this particular case every 5th call to the Twitter API activates a delay of 60 seconds.

```
from TwitterSearch import *
import time

try:
    tso = TwitterSearchOrder()
    tso.set_keywords(['foo', 'bar'])

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

    def my_callback_closure(current_ts_instance): # accepts ONE argument: an instance of TwitterSearc
        queries, tweets_seen = current_ts_instance.get_statistics()
        if queries > 0 and (queries % 5) == 0: # trigger delay every 5th query
            time.sleep(60) # sleep for 60 seconds

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

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

Remember that the callback is called every time a query to the Twitter API is performed. It's in your responsibility to make sure that your code doesn't have any unwanted side-effects or throws unintended exceptions. Also, every closure submitted via the `callback` argument is called with a the current instance of *TwitterSearch*. Performing a delay is just one way to use this callback pattern.

1.3.5 Avoid rate-limitation manually

As you might know there is a certain amount of *meta-data* available when using *TwitterSearch*. Some users might want to rely only on the `get_statistics()` method of the *TwitterSearch* to trigger, for example, an artificial

delay. This function returns a tuple of two integers. The first integer represents the amount of queries sent to Twitter so far, while the second one is an automatically increasing counter of the so far received tweets during those queries. Thus, an example taking those two meta-information into account could look like:

```
from TwitterSearch import *
import time

try:
    tso = TwitterSearchOrder()
    tso.set_keywords(['foo', 'bar'])

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

    sleep_for = 60 # sleep for 60 seconds
    last_amount_of_queries = 0 # used to detect when new queries are done

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

        current_amount_of_queries = ts.get_statistics()[0]
        if not last_amount_of_queries == current_amount_of_queries:
            last_amount_of_queries = current_amount_of_queries
            time.sleep(sleep_for)

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

1.3.6 Returned tweets

This library is trying to not hide anything from your eyes except the complexity of its 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_entit
                     '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/35183917488
                                    '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.jp
                                '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 Andro
          'text': '@joshdallas @ginnygoodwin home during wintertime"@ThatsEarth: Hohenzollern Castle
          '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/4764
                                                   '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/31
                    'profile_background_image_url_https': 'https://si0.twimg.com/profile_background_i
                    'profile_background_tile': True,
                    'profile_banner_url': 'https://pbs.twimg.com/profile_banners/65599486/1372576102
                    'profile_image_url': 'http://a0.twimg.com/profile_images/3763288269/57c274f19592f
                    'profile_image_url_https': 'https://si0.twimg.com/profile_images/3763288269/57c27
                    '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.

1.3.7 Access meta data

An output of the available meta data from the query to the Twitter API is stored in a `dict` structure. You can access it by calling `get_metadata()` 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
'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&oa
'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 `get_metadata()['content-location']`. Do **NOT** save or output those information to insecure environments!

If you are interested in the amount of queries that this library did automatically on your behalf you can access those information easily by calling `get_statistcs()`. A trivial example use-case could be to print out those informations as part of a debugging or logging facility: `print("Queries done: %i. Tweets received: %i" % ts.get_statistics())`

1.3.8 TwitterSearch without automatic iteration

It is also perfectly possible to use *TwitterSearch* without any automatic 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 its own. Just assume that we would like to implement this feature independently again. A possible solution of this could look like:

```
from TwitterSearch import *

try:
    tso = TwitterSearchOrder()
    tso.set_keywords(['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.search_tweets(tso)

        # print rate limiting status
        print( "Current rate-limiting status: %i" % ts.get_metadata()['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.set_max_id(next_max_id)

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

1.3.9 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.set_supported_languages(tso)

    # try to set German (see ISO 639-1) as language
    ts.set_language('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)
```

1.4 Advanced usage: The `TwitterSearchException` class

It is possible to print an *TwitterSearch* exception. Doing so will result in strings of the type `Error <TwitterSearchException.code>: <TwitterSearchException.message>`. For those new to Python, the standard way to print an exceptions is:

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

1.4.1 List of exceptions

There are *two* different kinds of exceptions in *TwitterSearch*. The first kind is based on the HTTP status of the query to the Twitter API while the second type of exceptions are based on misconfiguration of the library. Misconfiguration can be performed, 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*
1000	Neither a list nor a string
1001	Not a list object
1002	No ISO 6391-1 language code
1003	No valid result type
1004	Invalid number
1005	Invalid unit
1006	Invalid callback string
1007	Not a date object
1008	Invalid boolean
1009	Invalid string
1010	Not a valid TwitterSearchOrder object
1011	No more results available
1012	No meta data available
1013	No tweets available
1014	No results available
1015	No keywords given
1016	Invalid dict
1017	Invalid user id or screen-name
1018	Not a callable function

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* from the Twitter API. 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

1.4.2 Advanced exception usage

The HTTP exceptions are somehow configurable. Imagine there is a reason why you don't like TwitterSearch to raise an exception when a 404 HTTP status is returned by the Twitter API. Instead 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. Anyway, 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 too much of trouble.

```
from TwitterSearch import *

tso = TwitterSearchOrder()
tso.set_keywords(['strange', 'use-case'])
tso.set_include_entities(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.search_tweets_iterable(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!`.

1.5 Advanced usage: The `TwitterSearchOrder` class

This class mainly acts as a plain container for configuration all parameters currently available by the Twitter Search API. 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 this very Twitter API endpoint. 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. Please note that this endpoint has a different maximum size than the one used in `TwitterUserOrder`.

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***	`add_keyword(<string>)`, `add_keyword(<list>, or_operator=True)`, `set_keywords(<list>)`	`add_keyword('#Hashtag')`, `set_keywords(['foo','bar'])`
q	*optional*	`set_link_filter()`, `remove_link_filter()`	`set_link_filter()`, `remove_link_filter()`
q	*optional*	`set_question_filter()`, `remove_question_filter)`	`set_question_filter()`, `remove_question_filter)`
q	*optional*	`set_source_filter(<string>)`, `remove_source_filter)`	`set_source_filter('twitterfeed')`
q	*optional*	`set_positive_attitude_filter()`, `set_negative_attitude_filter`, `remove_attitude_filter)`	`set_positive_attitude_filter()`
geocode	*optional*	`set_geocode(latitude<float>, longitude<float>, radius<int,long>, imperial_metric=<True,False>`	`set_geocode(52.5233,13.4127,10,imper`
lang	*optional*	`set_language(<ISO-6391-string>)`	`set_language('en')`
locale	*optional*	`set_locale(<ISO-6391-string>)`	`set_locale('ja')`
result_type	*optional*	`set_result_type(<mixed,recent,popular>)`	`set_result_type('recent')`
count	*optional*	`set_count(<int>[Range: 1-100])`	`set_count(42)`
until	*optional*	`set_until(<datetime.date>)`	`set_until(datetime.date(2012, 12, 24))`
since_id	*optional*	`set_since_id(<int,long>[Range: >0])`	`set_since_id(250075927172759552)`
max_id	*optional*	`set_max_id(<int,long>[Range: >0])`	`set_max_id(249292149810667520)`
include_entities	*optional*	`set_include_etities(<bool,int>)`	`set_include_etities(True)`, `set_include_etities(1)`
callback	*optional*	`set_callback(<string>)`	`set_callback('myMethod')`

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

1.5.1 Advanced filtering

There are certain types of filters the Twitter Search API can handle. All of them are listed in the Twitter documentation. Unfortunately such advanced filters can be quite messy. Read this chapter if you'd like to use advanced queries of the Twitter API. All filters **not** based on keyword can be removed by using `TwitterSearchOrder.remove_all_filters()`.

Keywords with spaces

Twitter does search for keywords separately by default. This means looking for `James Bond` will return tweets containing the words `James` and `Bond` like in `There was a bond of friendship between James and`

`Amy`. However, sometimes you might like to look for reviews of the newest `James Bond` movie and therefore such tweets won't be much of a help for you. In this case make sure to add a keyword surrounded by `"`. Twitter will take such keywords as one phrase resulting in tweets actually containing `James Bond` like in `My name is Bond ... James Bond`.

TwitterSearch will handle those issues for you by default as calling `add_keyword("James Bond")` will look for `"James Bond"` as one full phrase while `add_keyword(["James","Bond"])` will result in a search for *James AND Bond*.

Excepting keywords

Sometimes you might like to look for tweets containing specific words but not containing a different one. It's easy to except a certain keyword by applying a dash prefix (`-`) to it. Thus, a line like `set_keywords(['Porsche', '-Fiat'])` will give you all tweets containing the word `Porsche` but only if there is no `Fiat` in it.

OR concatenating of keywords

It is possible to search for `foo OR bar` with *TwitterSearch*. Instead of simply adding those keywords like we did in the basic usage chapter, you can use the `or_operator=True` parameter. It's also possible to concatenate different keyword:

```
from TwitterSearch import *

try:
    tso = TwitterSearchOrder()
    tso.set_keywords(['Goofy', 'Nyancat'], or_operator = True)
    tso.add_keyword('BMW')

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

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

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

Concatenating several keywords can be tricky as the syntax of the Twitter Search API is pretty undocumented and only roughly defined. In my tests it turned out at a query like `Goofy OR Nycancat BMW` seemed to be the very same as `(Goofy OR Nycancat) AND BMW` although there is nothing mentioned in the documentation about concatenations of keywords. If you'd like to make sure your combination works, better use the official Twitter Search to perform some tests and see whether Twitter handles your query correctly.

Tweets of/from/mentioning a certain user

In this example we'll use the twitter account of Eric Jarosinski and his twitter user Nein Quarterly.

It's also possible to search for tweets of a certain user. You'd better use `TwitterUserOrder` for this as this actually queries the timeline of the user instead of using the Twitter Search API. Nonetheless, it's also possible to do through `TwitterSearchOrder`. Just add the prefix of `from:` to the username. Using standard *TwitterSearch* methods this would look like `add_keyword("from:neinquarterly")`.

Tweets directly to a user can be collected using the `to:` prefix in front of the username. Due to this tweets to `neinquarterly` can be collected using `add_keyword("to:neinquarterly")`.

If you'd like to receive tweets referencing a certain user you are able to gather them by using a `@` prefix in front of the username. Thus, the corresponding code snipped is `add_keyword("@neinquarterly")`.

Tweets with hyperlinks

In a different scenario you might be only interested in tweets containing a hyperlink. You can look for those tweets using the filter method of *TwitterSearch*:

```
from TwitterSearch import *

try:
    tso = TwitterSearchOrder()
    tso.set_keywords(['Mickey', '#Mouse'], or_operator = True)
    tso.set_link_filter()

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

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

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

This will return all tweets with a hyperlink in them and containing the keyword `Mickey` or the hashtag `#Mouse`. To remove a already set link filter, the method `remove_link_filter()` was added.

Tweets containing a question

It's also possible to receive only tweets that are asking a question. You can do so by setting the filter via `TwitterSearchOrder.set_question_filter()`. A removal of this filter can be done with `TwitterSearchOrder.remove_question_filter()`. Be aware that this filtering is done by Twitter and it doesn't necessary work well as it might miss questions in certain languages.

Attitude filtering

Twitter also offers an attitude-based filtering mechanism. You can search for positive tweets by using `TwitterSearchOrder.set_positive_attitude_filter()` and for negative ones by using `TwitterSearchOrder.set_negative_attitude_filter()`. The attitude filtering can be removed using `TwitterSearchOrder.remove_attitude_filter()`. Note that this filter mechanism is performed by Twitter directly and you may miss tweets not detected by those. This especially holds true for tweets not authored in English.

Source filtering

If you're interested in tweets only submitted using a specific software you can do so using the method `TwitterSearchOrder.set_source_filter(<string>)`. Calling

`set_source_filter("twitterfeed")` gives you only tweets submitted using TwitterFeed. The removal of this filter can be performed through `TwitterSearchOrder.remove_source_filter()`.

Time-based filtering

TwitterSearch tries to concentrate on simple query and does prefer to submit arguments as parameters instead of merging them into the query string. Thus *TwitterSearch* will generate raw query strings like `?q=foobar&until=2010-12-27` instead of `?q=foobar+since:2010-12-27`. Both versions will return the very same tweets but while the first one separates the values in different parameters, the second one just merges everything together. Doing so is likely to lead to long and possibly wrong query strings. Remember that you're perfectly able to submit stuff like `?q=foobar+since:2010-12-27+until:2010-12-26` which is obviously non-sense. If you would still like to dump everything into the `q` parameter you can do so manually by using `set_keywords(['since:2010-12-27','until:2010-12-26'])` for example.

If you have no specific reason to actually include those time-based filters into the search query parameter directly, you should use the default methods of `set_since_id()` and/or `set_until()`.

1.5.2 Advanced usage examples

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.set_language('nl')
    tso.set_locale('ja')
    tso.set_keywords(['One','Two'])
    tso.add_keyword('myKeyword')

    print(tso.create_search_url())

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 different URL.

```
from TwitterSearch import TwitterSearchOrder, TwitterSearchException

try:
    tso = TwitterSearchOrder()
    tso.set_language('nl')
    tso.set_locale('ja')
    tso.set_keywords(['One','Two'])
    tso.add_keyword('myKeyword')

    querystr = tso.create_search_url()

    # create a new TwitterSearchOrder based on the old query string and work with it
    tso2 = TwitterSearchOrder()
```

```
    tso2.set_search_url(querystr + '&result_type=mixed&include_entities=true')
    tso2.set_locale('en')
    print(tso2.create_search_url())

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 `set_search_url()` is not checked. Due to this it is perfectly valid to to stuff like `set_search_url('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 `set_search_url()` all previous configured parameters are lost.

1.6 Advanced usage: The `TwitterUserOrder` class

This class mainly acts as a plain container for configuration all parameters currently available by the Twitter Search API. 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 *200*. This is because it is the maximum of tweets returned by this very Twitter API endpoint. 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. Please note that this endpoint has a different maximum size than the one used in `TwitterSearchOrder`.

API Parameter	Type	Modifying methods	Example
user_id	*optional*	constructor (either screen-name or ID of user required)	`TwitterUserOrder("some_username")`
screen_name	*optional*	constructor (either screen-name or ID of user required)	`TwitterUserOrder(123457890)`
count	*optional*	`set_count(<int>[Range: 1-200])`	`set_count(42)`
until	*optional*	`set_until(<datetime.date>)`	`set_until(datetime.date(2012, 12, 24))`
since_id	*optional*	`set_since_id(<int,long>[Range: >0])`	`set_since_id(250075927172759552)`
max_id	*optional*	`set_max_id(<int,long>[Range: >0])`	`set_max_id(249292149810667520)`
trim_user	*optional*	`set_trim_user(<bool>)`	`set_trim_user(True)`
exclude_replies	*optional*	`set_exclude_replies(<bool>)`	`set_exclude_replies(False)`
contributor_details	*optional*	`set_contributor_details(<bool>)`	`set_contributor_details(True)`
include_rts	*optional*	`set_include_rts(<bool>)`	`set_include_rts(True)`

If you're not familiar with the meaning of the parameters, please have a look at the Twitter User Timeline API documentation. Most parameter are self-describing anyway. Only special use-cases may require those detailed configura-

tion values to be set, so don't worry if you don't touch one of those advanced methods in your code.

1.6.1 Advanced usage examples

You may want to use `TwitterUserOrder` 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 `TwitterUserOrder.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 TwitterUserOrder, TwitterSearchException

try:
    tuo = TwitterUserOrder("some_user")
    tuo.set_trim_user(True)
    tuo.set_exclude_replies(False)
    tuo.set_include_rts(True)

    print(tuo.create_search_url())

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

You'll receive `?trim_user=true&exclude_replies=false&include_rts=true` 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 a new instance of `TwitterUserOrder` with the same configuration but for a different user. This is one way to do exactly this:

```
from TwitterSearch import TwitterSearchOrder, TwitterSearchException

try:
    tuo = TwitterUserOrder("some_user")
    tuo.set_trim_user(True)
    tuo.set_exclude_replies(False)
    tuo.set_include_rts(True)

    querystr = tuo.createSearchURL()

    # create a new TwitterSearchOrder based on the old query string and work with it
    tuo2 = TwitterUserOrder("some_other_user")
    tuo2.set_search_url(querystr)

    print(tso2.create_search_url())

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

This piece of code will also result in an output of `?trim_user=true&exclude_replies=false&include_rts=true`.

Please be aware that the sense of arguments given by `set_search_url()` is not checked. Due to this it is perfectly valid to to stuff like `set_search_url('?trim_user=true&exclude_replies=false&include_rts=true&count=1337&foo=bar')`. When manually setting the string, the leading `?` sign is optional. Due to this you can force TwitterSearch to request custom queries. But be aware that those non-compatible queries are likely to fail. Use such techniques with caution as it 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 `TwitterUserOrder` methods.

Also note that if you're using `set_search_url()` all previous configured parameters are lost and overridden.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

CHAPTER 3

Contribution

Feel free to open issues, submit code or fork.

t

A

C

E

G

I

N

R

S

T