TwitterAPI Documentation

Release 2.3

geduldig

Contents

1	Topics	3
2	Modules	9
Pvi	thon Module Index	13

Minimal Python wrapper for Twitter's REST and Streaming APIs

The principle behind TwitterAPI's design is to provide a single method for accessing the Twitter API. You can call the request method with *any* endpoint found on Twitter's developer site, the complete reference for all endpoints. The benefits of a single-method approach are: less code for me to maintain, and just a single method for you to learn. Here is a quck example:

```
from TwitterAPI import TwitterAPI
api = TwitterAPI(consumer_key, consumer_secret, access_token_key, access_token_secret)
r = api.request('search/tweets', {'q':'pizza'})
print r.status_code
```

If you want Twitter's entire response as one long string, containing tweets in this example, you would use r.text. But, often an iterator is more useful:

```
for item in r.get_iterator():
    print item['user']['screen_name'], item['text']
```

The iterator returns decoded JSON objects. What makes the iterator very powerful is it works with both REST API and Streaming API endpoints. No syntax changes required; just supply any endpoint and parameters that are found on Twitter's dev site.

TwitterAPI is compatible with Python 2 and Python 3. It authenticates using either OAauth 1 or OAuth 2. It also supports web proxy server authentication. All with very little code change for you.

Contents 1

2 Contents

CHAPTER 1

Topics

Authentication

The first thing you need to do is create an application on apps.twitter.com and generate your oAuth keys.

Twitter supports both user and application authentication, known as oAuth 1 and oAuth 2, respectively. User authentication gives you access to all API endpoints, basically read and write persmission. Application authentication gives you access to just the read portion of the API – so, no creating or destroying tweets. Application authentication also has elevated rate limits.

User Authentication

Application Authentication

Proxy Server Authentication

If you are behind a firewall, you may also need to authenticate with a web proxy server in order to reach Twitter's servers. For this situation you include an additional argument in the initializer:

Replace SERVER:PORT with your proxy server, and replace USER and PASSWORD with your proxy server credentials.

Error Handling

Besides tweet statuses, the REST API and Streaming API iterators may return error and other messages. It is up to the application to test what type of object has been returned. Message types are documented here and here.

REST API Messages

REST API endpoints can return many more types of messages than Streaming API endpoints. Depending on the endpoint, you may want to handle a particular type of message, such as exceeding a rate limit or posting a duplicate tweet. Here is a general pattern for simply printing out any message and error code:

```
r = api.request('search/tweets', {'q':'pizza'})
for item in r.get_iterator():
    if 'text' in item:
        print item['text']
    elif 'message' in item:
        print '%s (%d)' % (item['message'], item['code'])
```

Streaming API Messages

Streaming API endpoints return a variety of messages, most are not really errors. For example, a "limit" message contains the number of tweets missing from the stream. This happens when the number of tweets matching your filter exceeds a threshold set by Twitter. Other useful messages are "disconnect" and "delete". The pattern is similar to the one preceding:

```
r = api.request('statuses/filter', {'track':'pizza'})
for item in r.get_iterator():
    if 'text' in item:
        print item['text']
    elif 'limit' in item:
        print '%d tweets missed' % item['limit']['track']
    elif 'disconnect' in item:
        print 'disconnecting because %s' % item['disconnect']['reason']
        break
```

Even if you are not interested in handling errors it is necessary to test that the object returned by an iterator is a valid tweet status before using the object.

4 Chapter 1. Topics

Paging Results

Paging refers to getting successive batches of results. The Streaming API endpoints in a sense do this inherently. REST API endpoints will never return more than a specificied maximum number of results. When you request *search/tweets*, for example, by default you will get at most 20 tweets. You can increase that number to a maximum of 100. This is the page size. Twitter provides a way to get successive pages, so it is possible to get more than 100 tweets with *search/tweets*, just not in a single request.

If you don't want to implement paging yourself, you can use the TwitterRestPager helper class with any REST API endpoint that returns multiples of something. The following, for example, searches for all tweets containing 'pizza' that Twitter has stored – about a week's worth.

```
r = TwitterRestPager(api, 'search/tweets', {'q':'pizza', 'count':100})
for item in r.get_iterator():
    if 'text' in item:
        print item['text']
    elif 'message' in item and item['code'] == 88:
        print 'SUSPEND, RATE LIMIT EXCEEDED: %s\n' % item['message']
        break
```

Tiny Examples

All the examples assume api is an authenticated instance of TwitterAPI. Typically, this is done as follows:

Get your last 50 tweets

```
r = api.request('statuses/home_timeline', {'count':50})
for item in r.get_iterator():
    if 'text' in item:
        print item['text']
```

Get your entire timeline

```
pager = TwitterRestPager(api, 'statuses/home_timeline', {'count':200})
for item in pager.get_iterator(wait=60):
    if 'text' in item:
        print item['text']
```

Post a tweet

```
r = api.request('statuses/update', {'status': 'I need pizza!'})
print 'SUCCESS' if r.status_code == 200 else 'FAILURE'
```

1.3. Paging Results 5

Post a tweet with a picture

Delete a tweet

```
r = api.request('statuses/destroy/:%d' % TWEET_ID)
print 'SUCCESS' if r.status_code == 200 else 'FAILURE'
```

Stream tweets

```
r = api.request('statuses/filter', {'track':'pizza'})
for item in r.get_iterator():
    if 'text' in item:
        print item['text']
```

Fault Tolerant Streams and Pages

There are a number of reasons for a stream to stop. Twitter will break your connection if you have more than two streams open with the same credentials, or if your credentials are not valid. Occassionally, the problem will be internal to Twitter and you will be disconnected. Other causes might be network instability or connection timeout.

Endless Stream

In order to keep a Streaming API request going indefinitely, you will need to re-make the request whenever the connection drops. TwitterAPI defines two exception classes for this purpose.

TwitterRequestError is thrown whenever the request fails (i.e. when the response status code is not 200). A status code of 500 or higher indicates a server error which is safe to ignore. Any other status code indicates an error with your request which you should fix before re-trying.

TwitterConnectionError is thrown when the connection times out or is interrupted. You can always immediately try making the request again.

Sometimes Twitter will inform you to close the connection by sending you a "disconnect" message. The message will contain a code which indicates the reason. Messages with a code of 2, 5, 6, or 7 are serious and you will need to fix the problem before making a new request. You can ignore all other messages.

6 Chapter 1. Topics

```
while True:
   try:
        iterator = api.request('statuses/filter', {'track':'pizza'}).get_iterator()
        for item in iterator:
            if 'text' in item:
                print(item['text'])
            elif 'disconnect' in item:
                event = item['disconnect']
                if event['code'] in [2,5,6,7]:
                    # something needs to be fixed before re-connecting
                    raise Exception(event['reason'])
                else:
                    # temporary interruption, re-try request
   except TwitterRequestError as e:
        if e.status_code < 500:</pre>
            # something needs to be fixed before re-connecting
        else:
            # temporary interruption, re-try request
    except TwitterConnectionError:
        # temporary interruption, re-try request
        pass
```

Last Week's Pages

Requests for REST API endpoints can throw TwitterRequestError and TwitterConnectionError. They do not, however, return "disconnect" messages. Twitter returns error messages for these endpoints with "message". Most of these errors require attention before re-trying the request, except those with codes of 130 or 131, which are internal server errors.

For making continuous REST API requests (i.e. paging), TwitterAPI provides TwitterRestPager. If you use this class to request tweets that have been posted back to one week old, for example, the class's iterator will handle both types of exceptions automatically. The iterator also handles "message" objects with 130 or 131 codes for you. Any other "message" object gets passed on for you to handle.

One final consideration is the endpoint's rate limit, determinted by the endpoint and whether you authenticate with oAuth 1 or oAuth 2. By default, the iterator waits 5 seconds between requests. This is sufficient for 180 requests in 15 minutes, the rate limit for "search/tweets" with oAuth 1. You can do better with oAuth 2. It permits 450 requests every 15 minutes, or 1 request per 2 seconds. The example below sets the wait assuming oAuth 2 rate limits.

```
iterator = TwitterRestPager(api, 'search/tweets', {'q':'pizza'}).get_iterator(wait=2)
try:
    for item in iterator:
        if 'text' in item:
            print(item['text'])
        elif 'message' in item:
            # something needs to be fixed before re-connecting
            raise Exception(item['message'])
```

8 Chapter 1. Topics

CHAPTER 2

Modules

TwitterAPI.TwitterAPI

Access REST API or Streaming API resources.

Parameters

- consumer_key Twitter application consumer key
- consumer_secret Twitter application consumer secret
- access_token_key Twitter application access token key
- access_token_secret Twitter application access token secret
- auth_type "oAuth1" (default) or "oAuth2"
- proxy_url HTTPS proxy URL (ex. "https://USER:PASSWORD@SERVER:PORT")

request (resource, params=None, files=None, method_override=None)
Request a Twitter REST API or Streaming API resource.

Parameters

- resource A valid Twitter endpoint (ex. "search/tweets")
- params Dictionary with endpoint parameters or None (default)
- files Dictionary with multipart-encoded file or None (default)
- method_override Request method to override or None (default)

Returns TwitterResponse

Raises TwitterConnectionError

```
class TwitterAPI.TwitterAPI.TwitterResponse (response, stream)
     Response from either a REST API or Streaming API resource call.
          Parameters
                 • response – The requests.Response object returned by the API call
                • stream – Boolean connection type (True if a streaming connection)
     close()
          Disconnect stream (blocks with Python 3).
     get_iterator()
          Get API dependent iterator.
               Returns Iterator for tweets or other message objects in response.
              Raises TwitterConnectionError, TwitterRequestError
     get_rest_quota()
          Quota information in the REST-only response header.
              Returns Dictionary of 'remaining' (count), 'limit' (count), 'reset' (time)
     headers
              Returns Dictionary of API response header contents.
      json(**kwargs)
          Get the response as a JSON object.
              Parameters **kwargs - Optional arguments that json.loads takes.
               Returns response as JSON object.
              Raises ValueError
     status_code
              Returns HTTP response status code.
     text
              Returns Raw API response text.
```

TwitterAPI.TwitterError

```
exception TwitterAPI.TwitterError.TwitterConnectionError(value)
Raised when the connection needs to be re-established

exception TwitterAPI.TwitterError.TwitterError
Base class for Twitter exceptions

exception TwitterAPI.TwitterError.TwitterRequestError(status_code)
Raised when request fails

Optional:
```

TwitterAPI.TwitterRestPager

10 Chapter 2. Modules

Parameters

- api An authenticated TwitterAPI object
- resource String with the resource path (ex. search/tweets)
- params Dictionary of resource parameters

```
get iterator(wait=5, new tweets=False)
```

Iterate response from Twitter REST API resource. Resource is called in a loop to retrieve consecutive pages of results.

Parameters

- wait Floating point number (default=5) of seconds wait between requests. Depending on the resource, appropriate values are 5 or 60 seconds.
- **new_tweets** Boolean determining the search direction. False (default) retrieves old results. True retrieves current results.

Returns JSON objects containing statuses, errors or other return info.

Raises TwitterRequestError

TwitterAPI.TwitterOAuth

Optional class for retrieving Twitter credentials stored in a text file.

Parameters

- **consumer_key** Twitter application consumer key
- consumer_secret Twitter application consumer secret
- access_token_key Twitter application access token key
- access_token_secret Twitter application access token secret

classmethod read_file (file_name=None)

Read OAuth credentials from a text file. File format:

```
consumer_key=YOUR_CONSUMER_KEY
consumer_secret=YOUR_CONSUMER_SECRET
access_token_key=YOUR_ACCESS_TOKEN
access_token_secret=YOUR_ACCESS_TOKEN_SECRET
```

Parameters file_name – File containing credentials or None (default) reads credentials from TwitterAPI/credentials.txt

12 Chapter 2. Modules

Python Module Index

t

```
TwitterAPI.TwitterAPI,9
TwitterAPI.TwitterError,10
TwitterAPI.TwitterOAuth,11
TwitterAPI.TwitterRestPager,10
```

14 Python Module Index

Index

C	TwitterError, 10					
close() (TwitterAPI.TwitterAPI.TwitterResponse	TwitterOAuth (class in TwitterAPI.TwitterOAuth), 11					
method), 10	TwitterRequestError, 10					
C	TwitterResponse (class in TwitterAPI.TwitterAPI), 9					
G	TwitterRestPager (class in TwitterAPI.TwitterRestPager), 10					
get_iterator() (TwitterAPI.TwitterAPI.TwitterResponse method), 10	10					
get_iterator() (TwitterAPI.TwitterRestPager.TwitterRestPager method), 11						
get_rest_quota() (Twitter-						
API.TwitterAPI.TwitterResponse method), 10						
Н						
$\begin{array}{ccc} \text{headers} & \text{(TwitterAPI.TwitterAPI.TwitterResponse} & \text{attribute), } 10 \\ \end{array}$						
J						
json() (TwitterAPI.TwitterAPI.TwitterResponse method),						
10						
R						
read_file() (TwitterAPI.TwitterOAuth.TwitterOAuth class						
method), 11						
request() (TwitterAPI.TwitterAPI method), 9						
S						
status_code (TwitterAPI.TwitterAPI.TwitterResponse at-						
tribute), 10						
T						
$ text \;\; (Twitter API. Twitter Response \;\; attribute), \\ 10$						
TwitterAPI (class in TwitterAPI.TwitterAPI), 9						
TwitterAPI.TwitterAPI (module), 9						
TwitterAPI.TwitterError (module), 10						
TwitterAPI.TwitterOAuth (module), 11						
TwitterAPI.TwitterRestPager (module), 10						
TwitterConnectionError, 10						