**1st June 2017**

**Overview**

The Twitter development team offers an overview and all the documentation necessary (either directly or indirectly) to allow third parties to connect to their platform’s content and services. Most of this documentation can be found at <https://dev.twitter.com/docs>. Their offered documentation for developers include the following:

* **Twitter for websites**: A standard suit which allows websites to display tweets, tweet timelines, follow, or share buttons, essentially allowing a link between a website and its Twitter account in order to engage its audience. <https://dev.twitter.com/web/overview>
* **Twitter Kit**: Encompasses the documentation and SDK necessary to search, display, authorize, and post content on Twitter through mobile devices. Android and iOS supported. <https://dev.twitter.com/twitterkit/overview>
* **Twitter Cards**: Allows developers to, with a few modifications to their websites, automatically attach rich photos and videos to tweets. Whenever someone tweets about their website, a “card” with their rich media of preference will be added to the tweet. <https://dev.twitter.com/cards/overview>
* **Streaming APIs**: Allows for a low latency access to Twitter’s global stream of data. Streaming clients have messages pushed onto them whenever updates occur, thus requiring a constant connection to Twitter. Useful for monitoring and processing tweets in real-time. <https://dev.twitter.com/streaming/overview>
* **Ads API**: Allows for developers to create and manage ad campaigns on the Twitter platform. Among other things, Twitter offers options to schedule campaigns, retrieve analytics, and manage audiences. <https://dev.twitter.com/ads/overview> (Payed Service)
* **MoPub:** Allows for the monetization for apps through mobile devices. <https://www.mopub.com/resources/docs/> (Requires your app to be on the app store)
* **Gnip**: Provides commercial-grade to real-time and historical Twitter data for businesses. While it does provide social data from other big medias’ APIs (e.g. Facebook, YouTube), it will only fetch and provide data that is publicly displayed by said APIs. Full firehose-based sources of social data is only available for the Twitter API and a few key partners, such as Disqus. <https://gnip.com/> (Payed Service)
* **REST APIs:** Provides tools and documentation for general-purpose uses of Twitter through third-party applications. <https://dev.twitter.com/rest/public>

**REST API**

The REST APIs provide programmatic access to Twitter data, such as creating new tweets, reading user profiles, and retrieving lists. These APIs identify applications using OAuth, an open protocol for secure authorization (<https://dev.twitter.com/oauth.html>). Responses are in JSON format.

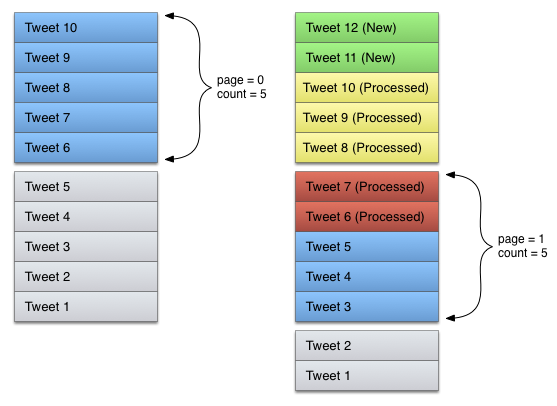
There are two forms of authentication:

* **User authentication**: The most common form of authentication. Once the user has given a specific app this authorization, the app will be permitted to make requests on the user’s behalf.
* **Application-only authentication**: Applications make API requests on their own behalf. This method includes the advantage of, at times, permitting a larger number of requests per window. Its downside is that, because it is devoid of user context, some methods will not be available (e.g. a tweet must be created by a logged user).

All applications must authenticate using OAuth, as it allows Twitter to analyze how applications engage with its data, as well as prevent abusive behavior.

The REST API works on a 15-minutes window. The standard buckets are either 15 calls per window or 180 calls per window, with different method allowing for a greater number of requests. The following link provides an overview of bucket sizes: <https://dev.twitter.com/rest/public/rate-limits>.

It is worth noting that making requests on the behalf of a user and making requests on the behalf of an application draw from two different request pools. (<https://dev.twitter.com/rest/public/rate-limits>).

Because Twitter works with timelines instead of pages, searching through lists of tweets as if they were organized by page can return redundant data. Instead, during the first call for tweets, developers should take note of the id of the very last tweet from their query. Subsequent queries should start their search from the lowest id downwards.

What follows are some notable queries through the API:

* **/statuses/user\_timeline.json**: Returns a given user’s (through their ID or alias) tweets and replies. May be set to omit replies by that user, as well as start from a specific tweet (through a tweet’s id) downwards. Max of 200 tweets per window.
* [**/statuses/show/{id}.json**](https://apigee.com/embed/console/twitter?req=%7B%22resource%22%3A%22statuses_user_timeline%22%2C%22params%22%3A%7B%22query%22%3A%7B%22count%22%3A%22200%22%2C%22exclude_replies%22%3A%22true%22%2C%22screen_name%22%3A%22sargon_of_akkad%22%7D%2C%22template%22%3A%7B%7D%2C%22headers%22%3A%7B%7D%2C%22body%22%3A%7B%22attachmentFormat%22%3A%22mime%22%2C%22attachmentContentDisposition%22%3A%22form-data%22%7D%7D%2C%22verb%22%3A%22get%22%7D#twitter#4): returns a single status, by ID.
* [**/search/tweets.json**](https://apigee.com/embed/console/twitter?req=%7B%22resource%22%3A%22search_tweets%22%2C%22params%22%3A%7B%22query%22%3A%7B%22q%22%3A%22battlerite%22%2C%22result_type%22%3A%22people%22%7D%2C%22template%22%3A%7B%7D%2C%22headers%22%3A%7B%7D%2C%22body%22%3A%7B%22attachmentFormat%22%3A%22mime%22%2C%22attachmentContentDisposition%22%3A%22form-data%22%7D%7D%2C%22verb%22%3A%22get%22%7D#twitter#10)**:** Returns a list of tweets based on a string of search, up to 1000 characters. Up to a maximum of 100 tweets per window. (NOTE: This feature is not exhaustive, and will mostly return tweets Twitter finds relevant in the last 7 days period. For a more comprehensive search of historic tweets, the StreamingAPI should be used).
* **/geo/**: A library of commands regarding locations in the real world. May be used to obtain all known information from a place (/id/{place\_id}.json); returns the id of a place, given a coordinate (reverse\_geosearch.json); returns a possible place that can be attached to a tweet, given some sort of identifying information (such as coordinates or IP address) (/search.json).
* **/friends/id.json**: Returns a list of who the specified user is following.
* **/followers/id.json**: Returns a list of who the specified user is being followed by.
* **/friendships/show.json**: Returns the friendship status between two arbitrary users.
* **/lists/:** A library of commands regarding lists and their metadata (such as participating users, subscribers, and most recent statuses).
* **/users/lookup.json**:Returns metadata of up to 100 users per window.

The complete list of commands can be found at <https://dev.twitter.com/rest/reference>.

**Authentication**

After setting up your app and a Twitter account, go to <https://apps.twitter.com> and create a new application. Make sure that you have granted the app all pertinent permissions (e.g.: if your app should be able to write tweets, but not access your private messages, change your settings accordingly). Note that your Twitter account requires a valid phone number to have apps connect to it.

Once your app is properly set at <https://apps.twitter.com>, click on the Keys and Access Token tab. Take note of your Consumer Key and your Consumer Secret. DO NOT SHARE YOUR SECRET WITH ANYONE ELSE.

If your app will make requests on its own behalf (Application-only), have your application send a POST oauth2/token (<https://dev.twitter.com/oauth/application-only>) along with your Consumer Key and Secret. This method will return a bearer token which will never change for this specific application. Should the token need to be discarded for whatever reason, use a POST oauth2/invalidate\_token call.

If your app will make requests on your behalf (User Authentication), you will need an access token for your app. That can either be done by creating one on the Keys and Access Tokens tab in your <https://apps.twitter.com> or by having your application requesting either a PIN-based authorization (<https://dev.twitter.com/oauth/pin-based>) or a 3-legged authorization (<https://dev.twitter.com/oauth/3-legged> ) , both of which will require the app to provide the Consumer Key, Secret, and the application’s name at <https://apps.twitter.com>. Take not of your Access Token number and Secret. DO NOT SHARE YOUR SECRET WITH ANYONE ELSE. Once created, this Access Token and its secret can be viewed at any time at your <https://apps.twitter.com> Keys and Access Token tab.

**Installing Twython**

There are several libraries on the internet to handle API/Application interface, in many different programming languages. For this example, Twython, a Python based library, will be used. Twython supports both Python 2 and Python 3. It can be retrieved at <https://github.com/ryanmcgrath/twython> , with its manual being located at <https://twython.readthedocs.io/en/latest/index.html> .

Once downloaded, extract the contents into the folder of your choosing. Then, open your command prompt and type:

*python C:\Twithon\setup.py install*

Notice that your operating system might not have Python in its PATH environmental variables, in which case, replace “python” with your python’s executable complete path.

Also notice that should your Twython installation be in a different folder than the one stated above, you must replace your command line accordingly.

Once the installation is complete, extract copy “portal.py” (attached to this manual) to Twython’s root folder (the one which has setup.py in it) and create a folder called “Results”; plural, with capital R. FAILURE IN DOING SO WILL CAUSE SOME OF THE FUNCTIONS TO CRASH

**Using Twython**

Open your IDLE shell (or your Python IDE of choice) and load portal.py. Insert your APP Key and secret (also called Consumer Key and Secret) into its respective quotes. If you plan on having your app doing requests on your behalf (as if it were logged on as a user), replace the OAuth Token and its secret in their respective quotes.

Once you have replaced your keys, run the code. Your app will still need its Access Token. Retrieve it by typing:

*getToken()*

This command will print your access token on the screen. Write it down somewhere, as well as replace the Access Token field in your portal.py code. Once all three (or five) variables have been inserted, you will be ready to begin contacting Twitter.

At this point, your Python IDE might complain you are missing modules, such as “requests” or “requests\_oauthlib”. If so, open your command prompt and type:

*C:\python27\Scripts\pip install requests*

Replace the “requests” in the example above with whatever library you might be missing. If your Python has a different installation directory, replace the one in the example with yours.

Portal.py four methods and one supporting method. The supporting method, currentTime(), simply formats the current date and time into a string for use by the other methods. The main methods are as follow:

**getTimeline(screenName, tweetCount, excludeReplies)**

This method returns the timeline of a user. It takes up to three arguments:

* screenName: obligatory. A string which must be enclosed in quotes. It’s the twitter handle of the user whose timeline will be retrieved (e.g. @jack would be typed as “jack”). There is no difference between capital and non-capital letters.
* tweetCount: An integer indicating how many tweets in reverse-chronological order will be retrieved. Up to 200 tweets may be retrieved per request. If no value is inserted, the default is 200.
* excludeReplies: Either True or False, with the default being False. If set to true, the timeline retrieved will not contain any tweets a person made in reply to someone else. Only “pure” tweets.

If the number of tweets to be retrieved is 5 or less, the tweets will be printed on the screen. If more than 5 tweets are retrieved, they will instead be stored in a text file with the time of the request in the “Results” folder. For more advanced uses of this method, consult the **Advanced Uses of the Twitter API** bellow.

**getStatus(tweetID)**

This method returns a single tweet. It takes a single argument:

* tweetID: The ID of the tweet, which is obligatory. This can be acquired in many ways but, most commonly, by opening the tweet in a different tab on your browser and copying the string of numbers at the end of the tweet’s address.

For more advanced uses of this method, consult the **Advanced Uses of the Twitter API** bellow.

**getFollowers(screenName, userCount, skipStatus)**

**getFollowing(screenName, userCount, skipStatus)**

These two functions will, respectfully, return a list of the followers of a user or a list of who that user is following. It takes up to three arguments:

* screenName: obligatory. A string which must be enclosed in quotes. It’s the twitter handle of the user whose following/follower lists will be retrieved (e.g. @jack would be typed as “jack”). There is no difference between capital and non-capital letters.
* userCount: An integer denoting the number of users that will be retrieved. Up to 200 users per request. Notice that users which have been banned or suspended will still be counted, but will be omitted from the final result. If no value is inserted, the default is 200.
* skipStatus: Can be either True or False, defaulting to True if nothing is inserted. While no visible change can be observed on screen or at the output files, this option means Twitter won’t return the user’s current status (latest tweet) in its reply.

If the number of tweets to be retrieved is 5 or less, the tweets will be printed on the screen. If more than 5 tweets are retrieved, they will instead be stored in a text file with the time of the request in the “Results” folder. For more advanced uses of this method, consult the **Advanced Uses of the Twitter API** bellow.

**Advanced Uses of the Twitter API through Twython**

As stated above, all of the supported commands to the Twitter API can be found at <https://dev.twitter.com/rest/reference> . Twython can handle most of them. To see which ones Twython can handle, go inside your Twython folder, in the twython subfolder. While “api.py” handles the connection with the API, “endpoints.py” actually handles the requests. Open endpoints.py to browse which requests Twython can handle.

A quick survey over the methods show that Twython can actually handle more parameters than the ones that portal.py would suggest. To understand which parameters can be passed and their possible values, follow the link in the appropriate comment.

For example, let’s suppose we wish to further explore what can be retrieved from a user’s timeline. Following the link provided, <https://dev.twitter.com/docs/api/1.1/get/statuses/user_timeline>, will bring us to this call’s documentation page. From the documentation, it can be seen that tweets from a specific timeframe can be retrieved, so long as we have the ID of the first tweet and of the last tweet. The call would look like this:

*GET https://api.twitter.com/1.1/statuses/user\_timeline.json?screen\_name=jack&count=2&since\_id=24012619984051000&max\_id=250126199840518145*

The code itself in Python, however, would look like this:

*twitter = Twython(APP\_KEY, access\_token=ACCESS\_TOKEN)*

*results = twitter.get\_user\_timeline(screen\_name=”jack”, count=2,since\_id=2401261998405100, max\_id=250126199840518145)*

*for result in results:*

*print 'Tweet ID: ', result['id']*

*print 'Text: ', result['text']*

*print 'Created: ', result['created\_at']*

*print '===================//===================\n'*

As it can also be gleaned from the bottom of the page, a slew of information regarding each tweet can be retrieved, including where it was tweeted from, at what time, who tweeted it, if the user is verified, etc. Consult the relevant documentation pages of the methods you intend to use.