**API Reference**

This page contains some basic documentation for the Tweepy module.

**tweepy.api — Twitter API wrapper**

***class*API([*auth\_handler=None*][, *host='api.twitter.com'*][, *search\_host='search.twitter.com'*][, *cache=None*][, *api\_root='/1'*][, *search\_root=''*][, *retry\_count=0*][, *retry\_delay=0*][, *retry\_errors=None*][, *timeout=60*][, *parser=ModelParser*][, *compression=False*][, *wait\_on\_rate\_limit=False*][, *wait\_on\_rate\_limit\_notify=False*][, *proxy=None*])**

This class provides a wrapper for the API as provided by Twitter. The functions provided in this class are listed below.

|  |  |
| --- | --- |
| **Parameters:** | * **auth\_handler** – authentication handler to be used * **host** – general API host * **search\_host** – search API host * **cache** – cache backend to use * **api\_root** – general API path root * **search\_root** – search API path root * **retry\_count** – default number of retries to attempt when error occurs * **retry\_delay** – number of seconds to wait between retries * **retry\_errors** – which HTTP status codes to retry * **timeout** – The maximum amount of time to wait for a response from Twitter * **parser** – The object to use for parsing the response from Twitter * **compression** – Whether or not to use GZIP compression for requests * **wait\_on\_rate\_limit** – Whether or not to automatically wait for rate limits to replenish * **wait\_on\_rate\_limit\_notify** – Whether or not to print a notification when Tweepy is waiting for rate limits to replenish * **proxy** – The full url to an HTTPS proxy to use for connecting to Twitter. |

**Timeline methods**

**API.home\_timeline([*since\_id*][, *max\_id*][, *count*][, *page*])**

Returns the 20 most recent statuses, including retweets, posted by the authenticating user and that user’s friends. This is the equivalent of /timeline/home on the Web.

|  |  |
| --- | --- |
| **Parameters:** | * **since\_id** – Returns only statuses with an ID greater than (that is, more recent than) the specified ID. * **max\_id** – Returns only statuses with an ID less than (that is, older than) or equal to the specified ID. * **count** – The number of results to try and retrieve per page. * **page** – Specifies the page of results to retrieve. Note: there are pagination limits. |
| **Return type:** | list of **Status** objects |

**API.statuses\_lookup(*id\_*[, *include\_entities*][, *trim\_user*][, *map\_*][, *include\_ext\_alt\_text*][, *include\_card\_uri*])**

Returns full Tweet objects for up to 100 tweets per request, specified by the **id\_** parameter.

|  |  |
| --- | --- |
| **Parameters:** | * **id\_** – A list of Tweet IDs to lookup, up to 100 * **include\_entities** – The entities node will not be included when set to false. Defaults to true. * **trim\_user** – A boolean indicating if user IDs should be provided, instead of complete user objects. Defaults to False. * **map\_** – A boolean indicating whether or not to include tweets that cannot be shown. Defaults to False. * **include\_ext\_alt\_text** – If alt text has been added to any attached media entities, this parameter will return an ext\_alt\_text value in the top-level key for the media entity. * **include\_card\_uri** – A boolean indicating if the retrieved Tweet should include a card\_uri attribute when there is an ads card attached to the Tweet and when that card was attached using the card\_uri value. |
| **Return type:** | list of **Status** objects |

**API.user\_timeline([*id/user\_id/screen\_name*][, *since\_id*][, *max\_id*][, *count*][, *page*])**

Returns the 20 most recent statuses posted from the authenticating user or the user specified. It’s also possible to request another user’s timeline via the id parameter.

|  |  |
| --- | --- |
| **Parameters:** | * **id** – Specifies the ID or screen name of the user. * **user\_id** – Specifies the ID of the user. Helpful for disambiguating when a valid user ID is also a valid screen name. * **screen\_name** – Specifies the screen name of the user. Helpful for disambiguating when a valid screen name is also a user ID. * **since\_id** – Returns only statuses with an ID greater than (that is, more recent than) the specified ID. * **max\_id** – Returns only statuses with an ID less than (that is, older than) or equal to the specified ID. * **count** – The number of results to try and retrieve per page. * **page** – Specifies the page of results to retrieve. Note: there are pagination limits. |
| **Return type:** | list of **Status** objects |

**API.retweets\_of\_me([*since\_id*][, *max\_id*][, *count*][, *page*])**

Returns the 20 most recent tweets of the authenticated user that have been retweeted by others.

|  |  |
| --- | --- |
| **Parameters:** | * **since\_id** – Returns only statuses with an ID greater than (that is, more recent than) the specified ID. * **max\_id** – Returns only statuses with an ID less than (that is, older than) or equal to the specified ID. * **count** – The number of results to try and retrieve per page. * **page** – Specifies the page of results to retrieve. Note: there are pagination limits. |
| **Return type:** | list of **Status** objects |

**API.mentions\_timeline([*since\_id*][, *max\_id*][, *count*])**

Returns the 20 most recent mentions, including retweets.

|  |  |
| --- | --- |
| **Parameters:** | * **since\_id** – Returns only statuses with an ID greater than (that is, more recent than) the specified ID. * **max\_id** – Returns only statuses with an ID less than (that is, older than) or equal to the specified ID. * **count** – The number of results to try and retrieve per page. |
| **Return type:** | list of **Status** objects |

**Status methods**

**API.get\_status(*id*[, *trim\_user*][, *include\_my\_retweet*][, *include\_entities*][, *include\_ext\_alt\_text*][, *include\_card\_uri*])**

Returns a single status specified by the ID parameter.

|  |  |
| --- | --- |
| **Parameters:** | * **id** – The numerical ID of the status. * **trim\_user** – A boolean indicating if user IDs should be provided, instead of complete user objects. Defaults to False. * **include\_my\_retweet** – A boolean indicating if any Tweets returned that have been retweeted by the authenticating user should include an additional current\_user\_retweet node, containing the ID of the source status for the retweet. * **include\_entities** – The entities node will not be included when set to false. Defaults to true. * **include\_ext\_alt\_text** – If alt text has been added to any attached media entities, this parameter will return an ext\_alt\_text value in the top-level key for the media entity. * **include\_card\_uri** – A boolean indicating if the retrieved Tweet should include a card\_uri attribute when there is an ads card attached to the Tweet and when that card was attached using the card\_uri value. |
| **Return type:** | **Status** object |

**API.update\_status(*status*[, *in\_reply\_to\_status\_id*][, *auto\_populate\_reply\_metadata*][, *exclude\_reply\_user\_ids*][, *attachment\_url*][, *media\_ids*][, *possibly\_sensitive*][, *lat*][, *long*][, *place\_id*][, *display\_coordinates*][, *trim\_user*][, *enable\_dmcommands*][, *fail\_dmcommands*][, *card\_uri*])**

Updates the authenticating user’s current status, also known as Tweeting.

For each update attempt, the update text is compared with the authenticating user’s recent Tweets. Any attempt that would result in duplication will be blocked, resulting in a 403 error. A user cannot submit the same status twice in a row.

While not rate limited by the API, a user is limited in the number of Tweets they can create at a time. If the number of updates posted by the user reaches the current allowed limit this method will return an HTTP 403 error.

|  |  |
| --- | --- |
| **Parameters:** | * **status** – The text of your status update. * **in\_reply\_to\_status\_id** – The ID of an existing status that the update is in reply to. Note: This parameter will be ignored unless the author of the Tweet this parameter references is mentioned within the status text. Therefore, you must include @username, where username is the author of the referenced Tweet, within the update. * **auto\_populate\_reply\_metadata** – If set to true and used with in\_reply\_to\_status\_id, leading @mentions will be looked up from the original Tweet, and added to the new Tweet from there. This wil append @mentions into the metadata of an extended Tweet as a reply chain grows, until the limit on @mentions is reached. In cases where the original Tweet has been deleted, the reply will fail. * **exclude\_reply\_user\_ids** – When used with auto\_populate\_reply\_metadata, a comma-separated list of user ids which will be removed from the server-generated @mentions prefix on an extended Tweet. Note that the leading @mention cannot be removed as it would break the in-reply-to-status-id semantics. Attempting to remove it will be silently ignored. * **attachment\_url** – In order for a URL to not be counted in the status body of an extended Tweet, provide a URL as a Tweet attachment. This URL must be a Tweet permalink, or Direct Message deep link. Arbitrary, non-Twitter URLs must remain in the status text. URLs passed to the attachment\_url parameter not matching either a Tweet permalink or Direct Message deep link will fail at Tweet creation and cause an exception. * **media\_ids** – A list of media\_ids to associate with the Tweet. You may include up to 4 photos or 1 animated GIF or 1 video in a Tweet. * **possibly\_sensitive** – If you upload Tweet media that might be considered sensitive content such as nudity, or medical procedures, you must set this value to true. * **lat** – The latitude of the location this Tweet refers to. This parameter will be ignored unless it is inside the range -90.0 to +90.0 (North is positive) inclusive. It will also be ignored if there is no corresponding long parameter. * **long** – The longitude of the location this Tweet refers to. The valid ranges for longitude are -180.0 to +180.0 (East is positive) inclusive. This parameter will be ignored if outside that range, if it is not a number, if geo\_enabled is disabled, or if there no corresponding lat parameter. * **place\_id** – A place in the world. * **display\_coordinates** – Whether or not to put a pin on the exact coordinates a Tweet has been sent from. * **trim\_user** – A boolean indicating if user IDs should be provided, instead of complete user objects. Defaults to False. * **enable\_dmcommands** – When set to true, enables shortcode commands for sending Direct Messages as part of the status text to send a Direct Message to a user. When set to false, disables this behavior and includes any leading characters in the status text that is posted * **fail\_dmcommands** – When set to true, causes any status text that starts with shortcode commands to return an API error. When set to false, allows shortcode commands to be sent in the status text and acted on by the API. * **card\_uri** – Associate an ads card with the Tweet using the card\_uri value from any ads card response. |
| **Return type:** | **Status** object |

**API.update\_with\_media(*filename*[, *status*][, *in\_reply\_to\_status\_id*][, *auto\_populate\_reply\_metadata*][, *lat*][, *long*][, *source*][, *place\_id*][, *file*])**

*Deprecated*: Use [**API.media\_upload()**](http://docs.tweepy.org/en/latest/api.html#API.media_upload) instead. Update the authenticated user’s status. Statuses that are duplicates or too long will be silently ignored.

|  |  |
| --- | --- |
| **Parameters:** | * **filename** – The filename of the image to upload. This will automatically be opened unless *file* is specified * **status** – The text of your status update. * **in\_reply\_to\_status\_id** – The ID of an existing status that the update is in reply to. * **auto\_populate\_reply\_metadata** – Whether to automatically include the @mentions in the status metadata. * **lat** – The location’s latitude that this tweet refers to. * **long** – The location’s longitude that this tweet refers to. * **source** – Source of the update. Only supported by Identi.ca. Twitter ignores this parameter. * **place\_id** – Twitter ID of location which is listed in the Tweet if geolocation is enabled for the user. * **file** – A file object, which will be used instead of opening *filename*. *filename* is still required, for MIME type detection and to use as a form field in the POST data |
| **Return type:** | **Status** object |

**API.destroy\_status(*id*)**

Destroy the status specified by the id parameter. The authenticated user must be the author of the status to destroy.

|  |  |
| --- | --- |
| **Parameters:** | **id** – The numerical ID of the status. |
| **Return type:** | **Status** object |

**API.retweet(*id*)**

Retweets a tweet. Requires the id of the tweet you are retweeting.

|  |  |
| --- | --- |
| **Parameters:** | **id** – The numerical ID of the status. |
| **Return type:** | **Status** object |

**API.retweeters(*id*[, *cursor*][, *stringify\_ids*])**

Returns up to 100 user IDs belonging to users who have retweeted the Tweet specified by the id parameter.

|  |  |
| --- | --- |
| **Parameters:** | * **id** – The numerical ID of the status. * **cursor** – Breaks the results into pages. Provide a value of -1 to begin paging. Provide values as returned to in the response body’s next\_cursor and previous\_cursor attributes to page back and forth in the list. * **stringify\_ids** – Have ids returned as strings instead. |
| **Return type:** | list of Integers |

**API.retweets(*id*[, *count*])**

Returns up to 100 of the first retweets of the given tweet.

|  |  |
| --- | --- |
| **Parameters:** | * **id** – The numerical ID of the status. * **count** – Specifies the number of retweets to retrieve. |
| **Return type:** | list of **Status** objects |

**API.unretweet(*id*)**

Untweets a retweeted status. Requires the id of the retweet to unretweet.

|  |  |
| --- | --- |
| **Parameters:** | **id** – The numerical ID of the status. |
| **Return type:** | **Status** object |

**User methods**

**API.get\_user(*id/user\_id/screen\_name*)**

Returns information about the specified user.

|  |  |
| --- | --- |
| **Parameters:** | * **id** – Specifies the ID or screen name of the user. * **user\_id** – Specifies the ID of the user. Helpful for disambiguating when a valid user ID is also a valid screen name. * **screen\_name** – Specifies the screen name of the user. Helpful for disambiguating when a valid screen name is also a user ID. |
| **Return type:** | **User** object |

**API.me()**

Returns the authenticated user’s information.

|  |  |
| --- | --- |
| **Return type:** | **User** object |

**API.friends([*id/user\_id/screen\_name*][, *cursor*][, *skip\_status*][, *include\_user\_entities*])**

Returns an user’s friends ordered in which they were added 100 at a time. If no user is specified it defaults to the authenticated user.

|  |  |
| --- | --- |
| **Parameters:** | * **id** – Specifies the ID or screen name of the user. * **user\_id** – Specifies the ID of the user. Helpful for disambiguating when a valid user ID is also a valid screen name. * **screen\_name** – Specifies the screen name of the user. Helpful for disambiguating when a valid screen name is also a user ID. * **cursor** – Breaks the results into pages. Provide a value of -1 to begin paging. Provide values as returned to in the response body’s next\_cursor and previous\_cursor attributes to page back and forth in the list. * **count** – The number of results to try and retrieve per page. * **skip\_status** – A boolean indicating whether statuses will not be included in the returned user objects. Defaults to false. * **include\_user\_entities** – The user object entities node will not be included when set to false. Defaults to true. |
| **Return type:** | list of **User** objects |

**API.followers([*id/screen\_name/user\_id*][, *cursor*])**

Returns a user’s followers ordered in which they were added. If no user is specified by id/screen name, it defaults to the authenticated user.

|  |  |
| --- | --- |
| **Parameters:** | * **id** – Specifies the ID or screen name of the user. * **user\_id** – Specifies the ID of the user. Helpful for disambiguating when a valid user ID is also a valid screen name. * **screen\_name** – Specifies the screen name of the user. Helpful for disambiguating when a valid screen name is also a user ID. * **cursor** – Breaks the results into pages. Provide a value of -1 to begin paging. Provide values as returned to in the response body’s next\_cursor and previous\_cursor attributes to page back and forth in the list. * **count** – The number of results to try and retrieve per page. * **skip\_status** – A boolean indicating whether statuses will not be included in the returned user objects. Defaults to false. * **include\_user\_entities** – The user object entities node will not be included when set to false. Defaults to true. |
| **Return type:** | list of **User** objects |

**API.lookup\_users([*user\_ids*][, *screen\_names*][, *include\_entities*][, *tweet\_mode*])**

Returns fully-hydrated user objects for up to 100 users per request.

There are a few things to note when using this method.

* You must be following a protected user to be able to see their most recent status update. If you don’t follow a protected user their status will be removed.
* The order of user IDs or screen names may not match the order of users in the returned array.
* If a requested user is unknown, suspended, or deleted, then that user will not be returned in the results list.
* If none of your lookup criteria can be satisfied by returning a user object, a HTTP 404 will be thrown.

|  |  |
| --- | --- |
| **Parameters:** | * **user\_ids** – A list of user IDs, up to 100 are allowed in a single request. * **screen\_names** – A list of screen names, up to 100 are allowed in a single request. * **include\_entities** – The entities node will not be included when set to false. Defaults to true. * **tweet\_mode** – Valid request values are compat and extended, which give compatibility mode and extended mode, respectively for Tweets that contain over 140 characters. |
| **Return type:** | list of **User** objects |

**API.search\_users(*q*[, *count*][, *page*])**

Run a search for users similar to Find People button on Twitter.com; the same results returned by people search on Twitter.com will be returned by using this API (about being listed in the People Search). It is only possible to retrieve the first 1000 matches from this API.

|  |  |
| --- | --- |
| **Parameters:** | * **q** – The query to run against people search. * **count** – Specifies the number of statuses to retrieve. May not be greater than 20. * **page** – Specifies the page of results to retrieve. Note: there are pagination limits. |
| **Return type:** | list of **User** objects |

**Direct Message Methods**

**API.get\_direct\_message([*id*][, *full\_text*])**

Returns a specific direct message.

|  |  |
| --- | --- |
| **Parameters:** | * **id** – The id of the Direct Message event that should be returned. * **full\_text** – A boolean indicating whether or not the full text of a message should be returned. If False the message text returned will be truncated to 140 chars. Defaults to False. |
| **Return type:** | **DirectMessage** object |

**API.list\_direct\_messages([*count*][, *cursor*])**

Returns all Direct Message events (both sent and received) within the last 30 days. Sorted in reverse-chronological order.

|  |  |
| --- | --- |
| **Parameters:** | * **count** – The number of results to try and retrieve per page. * **cursor** – Breaks the results into pages. Provide a value of -1 to begin paging. Provide values as returned to in the response body’s next\_cursor and previous\_cursor attributes to page back and forth in the list. |
| **Return type:** | list of **DirectMessage** objects |

**API.send\_direct\_message(*recipient\_id*, *text*[, *quick\_reply\_type*][, *attachment\_type*][, *attachment\_media\_id*])**

Sends a new direct message to the specified user from the authenticating user.

|  |  |
| --- | --- |
| **Parameters:** | * **recipient\_id** – The ID of the user who should receive the direct message. * **text** – The text of your Direct Message. Max length of 10,000 characters. * **quick\_reply\_type** –   The Quick Reply type to present to the user:   * + options - Array of Options objects (20 max).   + text\_input - Text Input object.   + location - Location object. * **attachment\_type** – The attachment type. Can be media or location. * **attachment\_media\_id** – A media id to associate with the message. A Direct Message may only reference a single media\_id. |
| **Return type:** | **DirectMessage** object |

**API.destroy\_direct\_message(*id*)**

Deletes the direct message specified in the required ID parameter. The authenticating user must be the recipient of the specified direct message. Direct Messages are only removed from the interface of the user context provided. Other members of the conversation can still access the Direct Messages.

|  |  |
| --- | --- |
| **Parameters:** | **id** – The id of the Direct Message that should be deleted. |
| **Return type:** | None |

**Friendship Methods**

**API.create\_friendship(*id/screen\_name/user\_id*[, *follow*])**

Create a new friendship with the specified user (aka follow).

|  |  |
| --- | --- |
| **Parameters:** | * **id** – Specifies the ID or screen name of the user. * **screen\_name** – Specifies the screen name of the user. Helpful for disambiguating when a valid screen name is also a user ID. * **user\_id** – Specifies the ID of the user. Helpful for disambiguating when a valid user ID is also a valid screen name. * **follow** – Enable notifications for the target user in addition to becoming friends. |
| **Return type:** | **User** object |

**API.destroy\_friendship(*id/screen\_name/user\_id*)**

Destroy a friendship with the specified user (aka unfollow).

|  |  |
| --- | --- |
| **Parameters:** | * **id** – Specifies the ID or screen name of the user. * **screen\_name** – Specifies the screen name of the user. Helpful for disambiguating when a valid screen name is also a user ID. * **user\_id** – Specifies the ID of the user. Helpful for disambiguating when a valid user ID is also a valid screen name. |
| **Return type:** | **User** object |

**API.show\_friendship(*source\_id/source\_screen\_name*, *target\_id/target\_screen\_name*)**

Returns detailed information about the relationship between two users.

|  |  |
| --- | --- |
| **Parameters:** | * **source\_id** – The user\_id of the subject user. * **source\_screen\_name** – The screen\_name of the subject user. * **target\_id** – The user\_id of the target user. * **target\_screen\_name** – The screen\_name of the target user. |
| **Return type:** | **Friendship** object |

**API.friends\_ids(*id/screen\_name/user\_id*[, *cursor*])**

Returns an array containing the IDs of users being followed by the specified user.

|  |  |
| --- | --- |
| **Parameters:** | * **id** – Specifies the ID or screen name of the user. * **screen\_name** – Specifies the screen name of the user. Helpful for disambiguating when a valid screen name is also a user ID. * **user\_id** – Specifies the ID of the user. Helpful for disambiguating when a valid user ID is also a valid screen name. * **cursor** – Breaks the results into pages. Provide a value of -1 to begin paging. Provide values as returned to in the response body’s next\_cursor and previous\_cursor attributes to page back and forth in the list. |
| **Return type:** | list of Integers |

**API.followers\_ids(*id/screen\_name/user\_id*)**

Returns an array containing the IDs of users following the specified user.

|  |  |
| --- | --- |
| **Parameters:** | * **id** – Specifies the ID or screen name of the user. * **screen\_name** – Specifies the screen name of the user. Helpful for disambiguating when a valid screen name is also a user ID. * **user\_id** – Specifies the ID of the user. Helpful for disambiguating when a valid user ID is also a valid screen name. * **cursor** – Breaks the results into pages. Provide a value of -1 to begin paging. Provide values as returned to in the response body’s next\_cursor and previous\_cursor attributes to page back and forth in the list. |
| **Return type:** | list of Integers |

**Account Methods**

**API.verify\_credentials([*include\_entities*][, *skip\_status*][, *include\_email*])**

Verify the supplied user credentials are valid.

|  |  |
| --- | --- |
| **Parameters:** | * **include\_entities** – The entities node will not be included when set to false. Defaults to true. * **skip\_status** – A boolean indicating whether statuses will not be included in the returned user objects. Defaults to false. * **include\_email** – When set to true email will be returned in the user objects as a string. |
| **Return type:** | **User** object if credentials are valid, otherwise False |

**API.rate\_limit\_status()**

Returns the current rate limits for methods belonging to the specified resource families. When using application-only auth, this method’s response indicates the application-only auth rate limiting context.

|  |  |
| --- | --- |
| **Parameters:** | **resources** – A comma-separated list of resource families you want to know the current rate limit disposition for. |
| **Return type:** | **JSON** object |

**API.update\_profile\_image(*filename*)**

Update the authenticating user’s profile image. Valid formats: GIF, JPG, or PNG

|  |  |
| --- | --- |
| **Parameters:** | **filename** – local path to image file to upload. Not a remote URL! |
| **Return type:** | **User** object |

**API.update\_profile\_background\_image(*filename*)**

Update authenticating user’s background image. Valid formats: GIF, JPG, or PNG

|  |  |
| --- | --- |
| **Parameters:** | **filename** – local path to image file to upload. Not a remote URL! |
| **Return type:** | **User** object |

**API.update\_profile([*name*][, *url*][, *location*][, *description*])**

Sets values that users are able to set under the “Account” tab of their settings page.

|  |  |
| --- | --- |
| **Parameters:** | * **name** – Maximum of 20 characters * **url** – Maximum of 100 characters. Will be prepended with “[http://](NULL)” if not present * **location** – Maximum of 30 characters * **description** – Maximum of 160 characters |
| **Return type:** | **User** object |

**Favorite Methods**

**API.favorites([*id*][, *page*])**

Returns the favorite statuses for the authenticating user or user specified by the ID parameter.

|  |  |
| --- | --- |
| **Parameters:** | * **id** – The ID or screen name of the user to request favorites * **page** – Specifies the page of results to retrieve. Note: there are pagination limits. |
| **Return type:** | list of **Status** objects |

**API.create\_favorite(*id*)**

Favorites the status specified in the ID parameter as the authenticating user.

|  |  |
| --- | --- |
| **Parameters:** | **id** – The numerical ID of the status. |
| **Return type:** | **Status** object |

**API.destroy\_favorite(*id*)**

Un-favorites the status specified in the ID parameter as the authenticating user.

|  |  |
| --- | --- |
| **Parameters:** | **id** – The numerical ID of the status. |
| **Return type:** | **Status** object |

**Block Methods**

**API.create\_block(*id/screen\_name/user\_id*)**

Blocks the user specified in the ID parameter as the authenticating user. Destroys a friendship to the blocked user if it exists.

|  |  |
| --- | --- |
| **Parameters:** | * **id** – Specifies the ID or screen name of the user. * **screen\_name** – Specifies the screen name of the user. Helpful for disambiguating when a valid screen name is also a user ID. * **user\_id** – Specifies the ID of the user. Helpful for disambiguating when a valid user ID is also a valid screen name. |
| **Return type:** | **User** object |

**API.destroy\_block(*id/screen\_name/user\_id*)**

Un-blocks the user specified in the ID parameter for the authenticating user.

|  |  |
| --- | --- |
| **Parameters:** | * **id** – Specifies the ID or screen name of the user. * **screen\_name** – Specifies the screen name of the user. Helpful for disambiguating when a valid screen name is also a user ID. * **user\_id** – Specifies the ID of the user. Helpful for disambiguating when a valid user ID is also a valid screen name. |
| **Return type:** | **User** object |

**API.blocks([*page*])**

Returns an array of user objects that the authenticating user is blocking.

|  |  |
| --- | --- |
| **Parameters:** | **page** – Specifies the page of results to retrieve. Note: there are pagination limits. |
| **Return type:** | list of **User** objects |

**API.blocks\_ids([*cursor*])**

Returns an array of numeric user ids the authenticating user is blocking.

|  |  |
| --- | --- |
| **Parameters:** | **cursor** – Breaks the results into pages. Provide a value of -1 to begin paging. Provide values as returned to in the response body’s next\_cursor and previous\_cursor attributes to page back and forth in the list. |
| **Return type:** | list of Integers |

**Mute Methods**

**API.create\_mute(*id/screen\_name/user\_id*)**

Mutes the user specified in the ID parameter for the authenticating user.

|  |  |
| --- | --- |
| **Parameters:** | * **id** – Specifies the ID or screen name of the user. * **screen\_name** – Specifies the screen name of the user. Helpful for disambiguating when a valid screen name is also a user ID. * **user\_id** – Specifies the ID of the user. Helpful for disambiguating when a valid user ID is also a valid screen name. |
| **Return type:** | **User** object |

**API.destroy\_mute(*id/screen\_name/user\_id*)**

Un-mutes the user specified in the ID parameter for the authenticating user.

|  |  |
| --- | --- |
| **Parameters:** | * **id** – Specifies the ID or screen name of the user. * **screen\_name** – Specifies the screen name of the user. Helpful for disambiguating when a valid screen name is also a user ID. * **user\_id** – Specifies the ID of the user. Helpful for disambiguating when a valid user ID is also a valid screen name. |
| **Return type:** | **User** object |

**API.mutes([*cursor*][, *include\_entities*][, *skip\_status*])**

Returns an array of user objects the authenticating user has muted.

|  |  |
| --- | --- |
| **Parameters:** | * **cursor** – Breaks the results into pages. Provide a value of -1 to begin paging. Provide values as returned to in the response body’s next\_cursor and previous\_cursor attributes to page back and forth in the list. * **include\_entities** – The entities node will not be included when set to false. Defaults to true. * **skip\_status** – A boolean indicating whether statuses will not be included in the returned user objects. Defaults to false. |
| **Return type:** | list of **User** objects |

**API.mutes\_ids([*cursor*])**

Returns an array of numeric user ids the authenticating user has muted.

|  |  |
| --- | --- |
| **Parameters:** | **cursor** – Breaks the results into pages. Provide a value of -1 to begin paging. Provide values as returned to in the response body’s next\_cursor and previous\_cursor attributes to page back and forth in the list. |
| **Return type:** | list of Integers |

**Spam Reporting Methods**

**API.report\_spam(*id/screen\_name/user\_id*[, *perform\_block*])**

The user specified in the id is blocked by the authenticated user and reported as a spammer.

|  |  |
| --- | --- |
| **Parameters:** | * **id** – Specifies the ID or screen name of the user. * **screen\_name** – Specifies the screen name of the user. Helpful for disambiguating when a valid screen name is also a user ID. * **user\_id** – Specifies the ID of the user. Helpful for disambiguating when a valid user ID is also a valid screen name. * **perform\_block** – A boolean indicating if the reported account should be blocked. Defaults to True. |
| **Return type:** | **User** object |

**Saved Searches Methods**

**API.saved\_searches()**

Returns the authenticated user’s saved search queries.

|  |  |
| --- | --- |
| **Return type:** | list of **SavedSearch** objects |

**API.get\_saved\_search(*id*)**

Retrieve the data for a saved search owned by the authenticating user specified by the given id.

|  |  |
| --- | --- |
| **Parameters:** | **id** – The id of the saved search to be retrieved. |
| **Return type:** | **SavedSearch** object |

**API.create\_saved\_search(*query*)**

Creates a saved search for the authenticated user.

|  |  |
| --- | --- |
| **Parameters:** | **query** – The query of the search the user would like to save. |
| **Return type:** | **SavedSearch** object |

**API.destroy\_saved\_search(*id*)**

Destroys a saved search for the authenticated user. The search specified by id must be owned by the authenticating user.

|  |  |
| --- | --- |
| **Parameters:** | **id** – The id of the saved search to be deleted. |
| **Return type:** | **SavedSearch** object |

**Help Methods**

**API.search(*q*[, *geocode*][, *lang*][, *locale*][, *result\_type*][, *count*][, *until*][, *since\_id*][, *max\_id*][, *include\_entities*])**

Returns a collection of relevant Tweets matching a specified query.

Please note that Twitter’s search service and, by extension, the Search API is not meant to be an exhaustive source of Tweets. Not all Tweets will be indexed or made available via the search interface.

In API v1.1, the response format of the Search API has been improved to return Tweet objects more similar to the objects you’ll find across the REST API and platform. However, perspectival attributes (fields that pertain to the perspective of the authenticating user) are not currently supported on this endpoint.[[1]](http://docs.tweepy.org/en/latest/api.html#id4)[[2]](http://docs.tweepy.org/en/latest/api.html#id5)

|  |  |
| --- | --- |
| **Parameters:** | * **q** – the search query string of 500 characters maximum, including operators. Queries may additionally be limited by complexity. * **geocode** – Returns 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. The parameter value is specified by “latitide,longitude,radius”, where radius units must be specified as either “mi” (miles) or “km” (kilometers). Note that you cannot use the near operator via the API to geocode arbitrary locations; however you can use this geocode parameter to search near geocodes directly. A maximum of 1,000 distinct “sub-regions” will be considered when using the radius modifier. * **lang** – Restricts tweets to the given language, given by an ISO 639-1 code. Language detection is best-effort. * **locale** – Specify the language of the query you are sending (only ja is currently effective). This is intended for language-specific consumers and the default should work in the majority of cases. * **result\_type** –   Specifies 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 in the response   + recent : return only the most recent results in the response   + popular : return only the most popular results in the response * **count** – The number of results to try and retrieve per page. * **until** – Returns tweets created before the given date. Date should be formatted as YYYY-MM-DD. Keep in mind that the search index has a 7-day limit. In other words, no tweets will be found for a date older than one week. * **since\_id** – Returns only statuses with an ID greater than (that is, more recent than) the specified ID. There are limits to the number of Tweets which can be accessed through the API. If the limit of Tweets has occurred since the since\_id, the since\_id will be forced to the oldest ID available. * **max\_id** – Returns only statuses with an ID less than (that is, older than) or equal to the specified ID. * **include\_entities** – The entities node will not be included when set to false. Defaults to true. |
| **Return type:** | **SearchResults** object |

**List Methods**

**API.create\_list(*name*[, *mode*][, *description*])**

Creates a new list for the authenticated user. Note that you can create up to 1000 lists per account.

|  |  |
| --- | --- |
| **Parameters:** | * **name** – The name of the new list. * **mode** – Whether your list is public or private. Values can be public or private. Lists are public by default if no mode is specified. * **description** – The description of the list you are creating. |
| **Return type:** | **List** object |

**API.destroy\_list([*owner\_screen\_name/owner\_id*, ]*list\_id/slug*)**

Deletes the specified list. The authenticated user must own the list to be able to destroy it.

|  |  |
| --- | --- |
| **Parameters:** | * **owner\_screen\_name** – The screen name of the user who owns the list being requested by a slug. * **owner\_id** – The user ID of the user who owns the list being requested by a slug. * **list\_id** – The numerical id of the list. * **slug** – You can identify a list by its slug instead of its numerical id. If you decide to do so, note that you’ll also have to specify the list owner using the owner\_id or owner\_screen\_name parameters. |
| **Return type:** | **List** object |

**API.update\_list(*list\_id/slug*[, *name*][, *mode*][, *description*][, *owner\_screen\_name/owner\_id*])**

Updates the specified list. The authenticated user must own the list to be able to update it.

|  |  |
| --- | --- |
| **Parameters:** | * **list\_id** – The numerical id of the list. * **slug** – You can identify a list by its slug instead of its numerical id. If you decide to do so, note that you’ll also have to specify the list owner using the owner\_id or owner\_screen\_name parameters. * **name** – The name for the list. * **mode** – Whether your list is public or private. Values can be public or private. Lists are public by default if no mode is specified. * **description** – The description to give the list. * **owner\_screen\_name** – The screen name of the user who owns the list being requested by a slug. * **owner\_id** – The user ID of the user who owns the list being requested by a slug. |
| **Return type:** | **List** object |

**API.lists\_all([*screen\_name*][, *user\_id*][, *reverse*])**

Returns all lists the authenticating or specified user subscribes to, including their own. The user is specified using the **user\_id** or **screen\_name** parameters. If no user is given, the authenticating user is used.

A maximum of 100 results will be returned by this call. Subscribed lists are returned first, followed by owned lists. This means that if a user subscribes to 90 lists and owns 20 lists, this method returns 90 subscriptions and 10 owned lists. The **reverse** method returns owned lists first, so with **reverse=true**, 20 owned lists and 80 subscriptions would be returned.

|  |  |
| --- | --- |
| **Parameters:** | * **screen\_name** – Specifies the screen name of the user. Helpful for disambiguating when a valid screen name is also a user ID. * **user\_id** – Specifies the ID of the user. Helpful for disambiguating when a valid user ID is also a valid screen name. * **reverse** – A boolean indicating if you would like owned lists to be returned first. See description above for information on how this parameter works. |
| **Return type:** | list of **List** objects |

**API.lists\_memberships([*screen\_name*][, *user\_id*][, *filter\_to\_owned\_lists*][, *cursor*][, *count*])**

Returns the lists the specified user has been added to. If **user\_id** or **screen\_name** are not provided, the memberships for the authenticating user are returned.

|  |  |
| --- | --- |
| **Parameters:** | * **screen\_name** – Specifies the screen name of the user. Helpful for disambiguating when a valid screen name is also a user ID. * **user\_id** – Specifies the ID of the user. Helpful for disambiguating when a valid user ID is also a valid screen name. * **filter\_to\_owned\_lists** – A boolean indicating whether to return just lists the authenticating user owns, and the user represented by **user\_id** or **screen\_name** is a member of. * **cursor** – Breaks the results into pages. Provide a value of -1 to begin paging. Provide values as returned to in the response body’s next\_cursor and previous\_cursor attributes to page back and forth in the list. * **count** – The number of results to try and retrieve per page. |
| **Return type:** | list of **List** objects |

**API.lists\_subscriptions([*screen\_name*][, *user\_id*][, *cursor*][, *count*])**

Obtain a collection of the lists the specified user is subscribed to, 20 lists per page by default. Does not include the user’s own lists.

|  |  |
| --- | --- |
| **Parameters:** | * **screen\_name** – Specifies the screen name of the user. Helpful for disambiguating when a valid screen name is also a user ID. * **user\_id** – Specifies the ID of the user. Helpful for disambiguating when a valid user ID is also a valid screen name. * **cursor** – Breaks the results into pages. Provide a value of -1 to begin paging. Provide values as returned to in the response body’s next\_cursor and previous\_cursor attributes to page back and forth in the list. * **count** – The number of results to try and retrieve per page. |
| **Return type:** | list of **List** objects |

**API.list\_timeline(*list\_id/slug*[, *owner\_id/owner\_screen\_name*][, *since\_id*][, *max\_id*][, *count*][, *include\_entities*][, *include\_rts*])**

Returns a timeline of tweets authored by members of the specified list. Retweets are included by default. Use the **include\_rts=false** parameter to omit retweets.

|  |  |
| --- | --- |
| **Parameters:** | * **list\_id** – The numerical id of the list. * **slug** – You can identify a list by its slug instead of its numerical id. If you decide to do so, note that you’ll also have to specify the list owner using the owner\_id or owner\_screen\_name parameters. * **owner\_id** – The user ID of the user who owns the list being requested by a slug. * **owner\_screen\_name** – The screen name of the user who owns the list being requested by a slug. * **since\_id** – Returns only statuses with an ID greater than (that is, more recent than) the specified ID. * **max\_id** – Returns only statuses with an ID less than (that is, older than) or equal to the specified ID. * **count** – The number of results to try and retrieve per page. * **include\_entities** – The entities node will not be included when set to false. Defaults to true. * **include\_rts** – A boolean indicating whether the list timeline will contain native retweets (if they exist) in addition to the standard stream of tweets. The output format of retweeted tweets is identical to the representation you see in home\_timeline. |
| **Return type:** | list of **Status** objects |

**API.get\_list(*list\_id/slug*[, *owner\_id/owner\_screen\_name*])**

Returns the specified list. Private lists will only be shown if the authenticated user owns the specified list.

|  |  |
| --- | --- |
| **Parameters:** | * **list\_id** – The numerical id of the list. * **slug** – You can identify a list by its slug instead of its numerical id. If you decide to do so, note that you’ll also have to specify the list owner using the owner\_id or owner\_screen\_name parameters. * **owner\_id** – The user ID of the user who owns the list being requested by a slug. * **owner\_screen\_name** – The screen name of the user who owns the list being requested by a slug. |
| **Return type:** | **List** object |

**API.add\_list\_member(*list\_id/slug*, *screen\_name/user\_id*[, *owner\_id/owner\_screen\_name*])**

Add a member to a list. The authenticated user must own the list to be able to add members to it. Lists are limited to 5,000 members.

|  |  |
| --- | --- |
| **Parameters:** | * **list\_id** – The numerical id of the list. * **slug** – You can identify a list by its slug instead of its numerical id. If you decide to do so, note that you’ll also have to specify the list owner using the owner\_id or owner\_screen\_name parameters. * **screen\_name** – Specifies the screen name of the user. Helpful for disambiguating when a valid screen name is also a user ID. * **user\_id** – Specifies the ID of the user. Helpful for disambiguating when a valid user ID is also a valid screen name. * **owner\_id** – The user ID of the user who owns the list being requested by a slug. * **owner\_screen\_name** – The screen name of the user who owns the list being requested by a slug. |
| **Return type:** | **List** object |

**API.add\_list\_members(*list\_id/slug*, *screen\_name/user\_id*[, *owner\_id/owner\_screen\_name*])**

Add up to 100 members to a list. The authenticated user must own the list to be able to add members to it. Lists are limited to 5,000 members.

|  |  |
| --- | --- |
| **Parameters:** | * **list\_id** – The numerical id of the list. * **slug** – You can identify a list by its slug instead of its numerical id. If you decide to do so, note that you’ll also have to specify the list owner using the owner\_id or owner\_screen\_name parameters. * **screen\_name** – A comma separated list of screen names, up to 100 are allowed in a single request * **user\_id** – A comma separated list of user IDs, up to 100 are allowed in a single request * **owner\_id** – The user ID of the user who owns the list being requested by a slug. * **owner\_screen\_name** – The screen name of the user who owns the list being requested by a slug. |
| **Return type:** | **List** object |

**API.remove\_list\_member(*list\_id/slug*, *screen\_name/user\_id*[, *owner\_id/owner\_screen\_name*])**

Removes the specified member from the list. The authenticated user must be the list’s owner to remove members from the list.

|  |  |
| --- | --- |
| **Parameters:** | * **list\_id** – The numerical id of the list. * **slug** – You can identify a list by its slug instead of its numerical id. If you decide to do so, note that you’ll also have to specify the list owner using the owner\_id or owner\_screen\_name parameters. * **screen\_name** – Specifies the screen name of the user. Helpful for disambiguating when a valid screen name is also a user ID. * **user\_id** – Specifies the ID of the user. Helpful for disambiguating when a valid user ID is also a valid screen name. * **owner\_id** – The user ID of the user who owns the list being requested by a slug. * **owner\_screen\_name** – The screen name of the user who owns the list being requested by a slug. |
| **Return type:** | **List** object |

**API.remove\_list\_members(*list\_id/slug*, *screen\_name/user\_id*[, *owner\_id/owner\_screen\_name*])**

Remove up to 100 members from a list. The authenticated user must own the list to be able to remove members from it. Lists are limited to 5,000 members.

|  |  |
| --- | --- |
| **Parameters:** | * **list\_id** – The numerical id of the list. * **slug** – You can identify a list by its slug instead of its numerical id. If you decide to do so, note that you’ll also have to specify the list owner using the owner\_id or owner\_screen\_name parameters. * **screen\_name** – A comma separated list of screen names, up to 100 are allowed in a single request * **user\_id** – A comma separated list of user IDs, up to 100 are allowed in a single request * **owner\_id** – The user ID of the user who owns the list being requested by a slug. * **owner\_screen\_name** – The screen name of the user who owns the list being requested by a slug. |
| **Return type:** | **List** object |

**API.list\_members(*list\_id/slug*[, *owner\_id/owner\_screen\_name*][, *cursor*])**

Returns the members of the specified list.

|  |  |
| --- | --- |
| **Parameters:** | * **list\_id** – The numerical id of the list. * **slug** – You can identify a list by its slug instead of its numerical id. If you decide to do so, note that you’ll also have to specify the list owner using the owner\_id or owner\_screen\_name parameters. * **owner\_id** – The user ID of the user who owns the list being requested by a slug. * **owner\_screen\_name** – The screen name of the user who owns the list being requested by a slug. * **cursor** – Breaks the results into pages. Provide a value of -1 to begin paging. Provide values as returned to in the response body’s next\_cursor and previous\_cursor attributes to page back and forth in the list. |
| **Return type:** | list of **User** objects |

**API.show\_list\_member(*list\_id/slug*, *screen\_name/user\_id*[, *owner\_id/owner\_screen\_name*])**

Check if the specified user is a member of the specified list.

|  |  |
| --- | --- |
| **Parameters:** | * **list\_id** – The numerical id of the list. * **slug** – You can identify a list by its slug instead of its numerical id. If you decide to do so, note that you’ll also have to specify the list owner using the owner\_id or owner\_screen\_name parameters. * **screen\_name** – Specifies the screen name of the user. Helpful for disambiguating when a valid screen name is also a user ID. * **user\_id** – Specifies the ID of the user. Helpful for disambiguating when a valid user ID is also a valid screen name. * **owner\_id** – The user ID of the user who owns the list being requested by a slug. * **owner\_screen\_name** – The screen name of the user who owns the list being requested by a slug. |
| **Return type:** | **User** object if user is a member of list |

**API.subscribe\_list(*list\_id/slug*[, *owner\_id/owner\_screen\_name*])**

Subscribes the authenticated user to the specified list.

|  |  |
| --- | --- |
| **Parameters:** | * **list\_id** – The numerical id of the list. * **slug** – You can identify a list by its slug instead of its numerical id. If you decide to do so, note that you’ll also have to specify the list owner using the owner\_id or owner\_screen\_name parameters. * **owner\_id** – The user ID of the user who owns the list being requested by a slug. * **owner\_screen\_name** – The screen name of the user who owns the list being requested by a slug. |
| **Return type:** | **List** object |

**API.unsubscribe\_list(*list\_id/slug*[, *owner\_id/owner\_screen\_name*])**

Unsubscribes the authenticated user from the specified list.

|  |  |
| --- | --- |
| **Parameters:** | * **list\_id** – The numerical id of the list. * **slug** – You can identify a list by its slug instead of its numerical id. If you decide to do so, note that you’ll also have to specify the list owner using the owner\_id or owner\_screen\_name parameters. * **owner\_id** – The user ID of the user who owns the list being requested by a slug. * **owner\_screen\_name** – The screen name of the user who owns the list being requested by a slug. |
| **Return type:** | **List** object |

**API.list\_subscribers(*list\_id/slug*[, *owner\_id/owner\_screen\_name*][, *cursor*][, *count*][, *include\_entities*][, *skip\_status*])**

Returns the subscribers of the specified list. Private list subscribers will only be shown if the authenticated user owns the specified list.

|  |  |
| --- | --- |
| **Parameters:** | * **list\_id** – The numerical id of the list. * **slug** – You can identify a list by its slug instead of its numerical id. If you decide to do so, note that you’ll also have to specify the list owner using the owner\_id or owner\_screen\_name parameters. * **owner\_id** – The user ID of the user who owns the list being requested by a slug. * **owner\_screen\_name** – The screen name of the user who owns the list being requested by a slug. * **cursor** – Breaks the results into pages. Provide a value of -1 to begin paging. Provide values as returned to in the response body’s next\_cursor and previous\_cursor attributes to page back and forth in the list. * **count** – The number of results to try and retrieve per page. * **include\_entities** – The entities node will not be included when set to false. Defaults to true. * **skip\_status** – A boolean indicating whether statuses will not be included in the returned user objects. Defaults to false. |
| **Return type:** | list of **User** objects |

**API.show\_list\_subscriber(*list\_id/slug*, *screen\_name/user\_id*[, *owner\_id/owner\_screen\_name*])**

Check if the specified user is a subscriber of the specified list.

|  |  |
| --- | --- |
| **Parameters:** | * **list\_id** – The numerical id of the list. * **slug** – You can identify a list by its slug instead of its numerical id. If you decide to do so, note that you’ll also have to specify the list owner using the owner\_id or owner\_screen\_name parameters. * **screen\_name** – Specifies the screen name of the user. Helpful for disambiguating when a valid screen name is also a user ID. * **user\_id** – Specifies the ID of the user. Helpful for disambiguating when a valid user ID is also a valid screen name. * **owner\_id** – The user ID of the user who owns the list being requested by a slug. * **owner\_screen\_name** – The screen name of the user who owns the list being requested by a slug. |
| **Return type:** | **User** object if user is subscribed to list |

**Trends Methods**

**API.trends\_available()**

Returns the locations that Twitter has trending topic information for. The response is an array of “locations” that encode the location’s WOEID (a Yahoo! Where On Earth ID) and some other human-readable information such as a canonical name and country the location belongs in.

|  |  |
| --- | --- |
| **Return type:** | **JSON** object |

**API.trends\_place(*id*[, *exclude*])**

Returns the top 50 trending topics for a specific WOEID, if trending information is available for it.

The response is an array of “trend” objects that encode the name of the trending topic, the query parameter that can be used to search for the topic on Twitter Search, and the Twitter Search URL.

This information is cached for 5 minutes. Requesting more frequently than that will not return any more data, and will count against your rate limit usage.

The tweet\_volume for the last 24 hours is also returned for many trends if this is available.

|  |  |
| --- | --- |
| **Parameters:** | * **id** – The Yahoo! Where On Earth ID of the location to return trending information for. Global information is available by using 1 as the WOEID. * **exclude** – Setting this equal to hashtags will remove all hashtags from the trends list. |
| **Return type:** | **JSON** object |

**API.trends\_closest(*lat*, *long*)**

Returns the locations that Twitter has trending topic information for, closest to a specified location.

The response is an array of “locations” that encode the location’s WOEID and some other human-readable information such as a canonical name and country the location belongs in.

A WOEID is a Yahoo! Where On Earth ID.

|  |  |
| --- | --- |
| **Parameters:** | * **lat** – If provided with a long parameter the available trend locations will be sorted by distance, nearest to furthest, to the co-ordinate pair. The valid ranges for longitude is -180.0 to +180.0 (West is negative, East is positive) inclusive. * **long** – If provided with a lat parameter the available trend locations will be sorted by distance, nearest to furthest, to the co-ordinate pair. The valid ranges for longitude is -180.0 to +180.0 (West is negative, East is positive) inclusive. |
| **Return type:** | **JSON** object |

**Geo Methods**

**API.reverse\_geocode([*lat*][, *long*][, *accuracy*][, *granularity*][, *max\_results*])**

Given a latitude and longitude, looks for places (cities and neighbourhoods) whose IDs can be specified in a call to [**update\_status()**](http://docs.tweepy.org/en/latest/api.html#API.update_status) to appear as the name of the location. This call provides a detailed response about the location in question; the **nearby\_places()** function should be preferred for getting a list of places nearby without great detail.

|  |  |
| --- | --- |
| **Parameters:** | * **lat** – The location’s latitude. * **long** – The location’s longitude. * **accuracy** – Specify the “region” in which to search, such as a number (then this is a radius in meters, but it can also take a string that is suffixed with ft to specify feet). If this is not passed in, then it is assumed to be 0m * **granularity** – Assumed to be **neighborhood** by default; can also be **city**. * **max\_results** – A hint as to the maximum number of results to return. This is only a guideline, which may not be adhered to. |

**API.geo\_id(*id*)**

Given *id* of a place, provide more details about that place.

|  |  |
| --- | --- |
| **Parameters:** | **id** – Valid Twitter ID of a location. |

**Utility methods**

**API.configuration()**

Returns the current configuration used by Twitter including twitter.com slugs which are not usernames, maximum photo resolutions, and t.co shortened URL length. It is recommended applications request this endpoint when they are loaded, but no more than once a day.

**Media methods**

**API.media\_upload(*filename*[, *file*])**

Use this endpoint to upload images to Twitter.

|  |  |
| --- | --- |
| **Parameters:** | * **filename** – The filename of the image to upload. This will automatically be opened unless **file** is specified. * **file** – A file object, which will be used instead of opening **filename**. **filename** is still required, for MIME type detection and to use as a form field in the POST data. |
| **Return type:** | **Media** object |

**API.create\_media\_metadata(*media\_id*, *alt\_text*)**

This endpoint can be used to provide additional information about the uploaded media\_id. This feature is currently only supported for images and GIFs. Call this endpoint to attach additional metadata such as image alt text.

|  |  |
| --- | --- |
| **Parameters:** | * **media\_id** – The ID of the media to add alt text to. * **alt\_text** – The alt text to add to the image. |

**tweepy.error — Exceptions**

The exceptions are available in the tweepy module directly, which means tweepy.error itself does not need to be imported. For example, tweepy.error.TweepError is available as tweepy.TweepError.

***exception*TweepError**

The main exception Tweepy uses. Is raised for a number of things.

When a **TweepError** is raised due to an error Twitter responded with, the error code ([as described in the API documentation](https://developer.twitter.com/en/docs/basics/response-codes)) can be accessed at **TweepError.response.text**. Note, however, that **TweepError**s also may be raised with other things as message (for example plain error reason strings).

***exception*RateLimitError**

Is raised when an API method fails due to hitting Twitter’s rate limit. Makes for easy handling of the rate limit specifically.

Inherits from [**TweepError**](http://docs.tweepy.org/en/latest/api.html#TweepError), so **except TweepError** will catch a **RateLimitError** too.