ResponseBot Documentation

Release 0.1.0

East Agile

Aug 30, 2017

Contents

Contents:

CHAPTER 1

Getting Started

Introduction

A framework to quickly develop listen-and-answer twitter bots. You can define multiple actions to be executed every time a tweet arrives.

Installation

```
$ pip install responsebot
```

Quick start

Authenticate

Create a `.responsebot` file in your project root with your Twitter API credentials (which can be obtained after you created a Twitter application here).

```
[auth]
consumer_key = <consumer_key>
consumer_secret = <consumer_secret>
token_key = <token_key>
token_secret = <token_secret>
```

Create a handler

```
from responsebot.handlers import BaseTweetHandler, register_handler


@register_handler
class MyTweetHandler(BaseTweetHandler):
    def get_filter(self):
        return TweetFilter(track=['Donald Trump'], follow=['<your personal Twitter id>'])

    def on_tweet(self, tweet):
        print('Received tweet: %s from %s' % (tweet.text, tweet.user.screen_name))
```

Execute

```
$ start_responsebot --handlers-package <python path to your package/module>
```

Test

The bot should now receive tweets containing 'Donald Trump' or tweets posted by you. You should see ResponseBot outputs

```
Received tweet: <your tweet content> from <your sender tweet account>
```

Handler

For a list of methods you can define in your custom handlers, see handler reference.

Reply to tweets

You can reply to received tweets, see more in the tutorial's client section

CHAPTER 2

Tutorial

Authenticate

ResponseBot uses Twitter API, which requires authentication. To do so, you need to register an application with Twitter here. Twitter will provide your app with a consumer key & secret pair, you need to generate an additional app token key & secret pair from your app management panel. After you have the credentials, put it under the `[auth]` section in the `.responsebot` configuration file in your project root (or whichever directory you run ResponseBot from).

```
[auth]
consumer_key = <consumer_key>
consumer_secret = <consumer_secret>
token_key = <token_key>
token_secret = <token_secret>
```

If the credentials you provided are correct, when you run ResponseBot it should show like below

```
$ start_responsebot --handlers-package <path_to_handler>
  [INFO] 2016-05-04 10:54:13 ResponseBot started
  [INFO] 2016-05-04 10:54:16 Successfully authenticated as <twitter_screen_name>
```

Otherwise it should show an error

```
[ERROR] 2016-05-04 10:52:17 Could not authenticate.
```

You can pass the credentials as a `start_responsebot`'s option instead of using a config file:

```
$ start_responsebot --auth <consumer_key> <consumer_secret> <token_key> <token_secret>
```

Listen to public stream or user stream

The bot can listen to every tweets in the world (that match some keywords) or it can listen to its authenticated user's timeline, as if it is that user. By default, the bot listen to the public stream, you can tell it to listen to the user stream as follow:

```
$ start_responsebot --user-stream
```

See more about stream and filters here.

Handler

To receive an incoming tweet, you need to subclass `BaseTweetHandler` and implement the `on_tweet` method. You can specify what kind of tweets the bot should listen to by returning an appropriate TweetFilter in the `get_filter` method.

```
class MyTweetHandler(BaseTweetHandler):
    def on_tweet(self, tweet):
        print('Received tweet: %s from %s' % (tweet.text, tweet.user.screen_name))

    def get_filter(self):
     return TweetFilter(track=['Donald Trump'], follow=['<your personal Twitter id>'])
```

See what `tweet` object contains in reference.

If you're listening on user stream, you can catch non-tweet events (user following, etc.) as in this tutorial.

Client

If your application want to post a reply to the received tweet, you can use ResponseBot's Twitter client to do so:

```
class MyTweetHandler(BaseTweetHandler):
    def on_tweet(self, tweet):
        self.client.tweet('Howdy @%s' % tweet.user.screen_name)
```

The `client` object can also retweet, get or delete a specific tweet by ID. See reference.

CHAPTER 3

Streams and filters

Streams

Twitter has three stream APIs to listen to. ResponseBot currently implement the public stream and user stream listening methods. By default the bot listen to public stream, you can tell the bot to listen to user stream by setting it in the config file:

```
[stream]
user_stream=true
```

or pass in a flag in the start command:

```
$ start_responsebot --user-stream
```

Filters

The public stream utilize two parameters: `track` and `follow`, to filter global streams (you cannot listen to every tweets in the world, you must have either 1 keyword to track or 1 user to follow). You provide these parameters in your handlers' `get_filter` method as follow:

```
def get_filter(self):
    return TweetFilter(track=['keyword'], follow=['user_id'])
```

By default the BaseTweetHandler returns a filter with the `follow` parameter set as the bot's authenticated user and an empty `track` parameter.

The `follow` parameter will not be used if you use `user_stream`. See more about TweetFilter

CHAPTER 4

User event handling

If you're listening to user stream, you can catch non-tweet events (user following, etc.) by subclassing the BaseEventHandler class and set it in your tweet handler, as follow:

```
class MyEventHandler(BaseEventHandler):
    def on_follow(self, event):
        pass

    def handle(self, event):
        super(MyEventHandler, self).handle(event)
        # do sth


class MyTweetHandler(BaseTweetHandler):
    event_handler_class = MyEventHandler
```

Currently we support callbacks `on_<event>` so you can easily implementing them. You can override `handle` for more customization if needed.

For a list of Twitter user events, visit this docs.

CHAPTER 5

Reference

responsebot.handlers.base	
responsebot.handlers.event	
responsebot.responsebot_client	
responsebot.models	Entity classes for various entity types for ResponseBot
responsebot.common.exceptions	Exceptions and errors used by Tweet Bot

responsebot.handlers.base

Classes

BaseTweetHandler([client])	An abstract base tweet handler class for the user to subclass.

class responsebot.handlers.base.**BaseTweetHandler**(*client=None*, **args*, ***kwargs*)

An abstract base tweet handler class for the user to subclass.

__init__(*client=None*, **args*, ***kwargs*)

Init a handler, try to create event handler if appropriate.

Parameters **client** – Some Twitter API client for authentication. E.g. `TweetClient`

get_filter()

Override this method for custom filter. By default returns a filter with the bot's authenticated user ID in the follow list.

Example:

```
return TweetFilter(track=['hello'], follow=['<some_user_id>'])`
```

on_event(*event*)

Callback for when a non-tweet event is sent. By default, this will call an event handler passed by *event_handler_class*

Parameters **event** – The received event

on_tweet(*tweet*)

Callback for when a tweet appears in user timeline

Parameters **tweet** (*Tweet*) – The incoming tweet

responsebot.handlers.event

Classes

BaseEventHandler(client)	Abstract event handler.

class responsebot.handlers.event.**BaseEventHandler**(*client*)

Abstract event handler. Read more about event here and here.

__init__(*client*)

Init event handler.

Parameters **client** – Some Twitter API client for authentication. E.g. TweetClient

handle(*event*)

Entry point to handle user events.

Parameters **event** – Received event. See a full list here.

on_access_revoked(*event*)

Event-specific callback for handling access_revoked events. This will trigger when you deauthorize a stream. See more of this here

Parameters **event** (*Event*) – Received event.

on_block(*event*)

Event-specific callback for handling block events. This will trigger when you block someone.

Parameters **event** (*Event*) – Received event.

on_favorite(*event*)

Event-specific callback for handling favorite events. This will trigger when someone like your tweet or you like someone's tweet.

Parameters **event** (*Event*) – Received event.

on_follow(*event*)

Event-specific callback for handling follow events. This will trigger when someone follow the current user or when current user follow someone.

Parameters **event** (*Event*) – Received event.

on_list_created(*event*)

Event-specific callback for handling list_created events. This will trigger when you create a list.

Parameters **event** (*Event*) – Received event.

on_list_destroyed(*event*)

Event-specific callback for handling list_destroyed events. This will trigger when you delete your list.

Parameters **event** (*Event*) – Received event.

on_list_member_added(*event*)

Event-specific callback for handling list_member_added events. This will trigger when you are added to a list or you add someone to your list.

Parameters **event** (*Event*) – Received event.

on_list_member_removed(*event*)
Event-specific callback for handling `list_member_removed` events. This will trigger when you are removed from a list or you remove someone from your list.

Parameters **event** (*Event*) – Received event.

on_list_updated(*event*)
Event-specific callback for handling `list_updated` events. This will trigger when you update your list.

Parameters **event** (*Event*) – Received event.

on_list_user_subscribed(*event*)
Event-specific callback for handling `list_user_subscribed` events. This will trigger when your list is subscribed to or you subscribe to a list.

Parameters **event** (*Event*) – Received event.

on_list_user_unsubscribed(*event*)
Event-specific callback for handling `list_user_unsubscribed` events. This will trigger when your list is unsubscribed from or you unsubscribe from a list.

Parameters **event** (*Event*) – Received event.

on_quoted_tweet(*event*)
Event-specific callback for handling `quoted_tweet` events. This will trigger when someone quote your tweet.

Parameters **event** (*Event*) – Received event.

on_unblock(*event*)
Event-specific callback for handling `unblock` events. This will trigger when you unblock someone.

Parameters **event** (*Event*) – Received event.

on_unfavorite(*event*)
Event-specific callback for handling `unfavorite` events. This will trigger when someone unlike your tweet or you unlike someone's tweet.

Parameters **event** (*Event*) – Received event.

on_unfollow(*event*)
Event-specific callback for handling `unfollow` events. This will trigger when you unfollow someone.

Parameters **event** (*Event*) – Received event.

on_user_update(*event*)
Event-specific callback for handling `user_update` events. This will trigger when you update your profile or private tweets.

Parameters **event** (*Event*) – Received event.

responsebot.responsebot_client

Classes

ResponseBotClient(client, config)	Wrapper for all Twitter API clients.

class `responsebot.responsebot_client.`**ResponseBotClient**(*client*, *config*)
Wrapper for all Twitter API clients.

add_list_member(*list_id*, *user_id*)
Add a user to list

Parameters

- **list_id** – list ID number
- **user_id** – user ID number

Returns *List* object

create_list(*name*, *mode='public'*, *description=None*)
Create a list

Parameters

- **name** – Name of the new list
- **mode** – `'public'` (default) or `'private'`
- **description** – Description of the new list

Returns The new list object

Return type *List*

destroy_list(*list_id*)
Destroy a list

Parameters **list_id** – list ID number

Returns The destroyed list object

Return type *List*

follow(*user_id*, *notify=False*)
Follow a user.

Parameters

- **user_id** – ID of the user in question
- **notify** – whether to notify the user about the following

Returns user that are followed

get_list(*list_id*)
Get info of specified list

Parameters **list_id** – list ID number

Returns *List* object

get_tweet(*id*)
Get an existing tweet.

Parameters **id** – ID of the tweet in question

Returns Tweet object. None if not found

get_user(*id*)
Get a user's info.

Parameters **id** – ID of the user in question

Returns User object. None if not found

is_list_member(*list_id*, *user_id*)
Check if a user is member of a list

Parameters

- **list_id** – list ID number
- **user_id** – user ID number

Returns `True` if user is member of list, `False` otherwise

is_subscribed_list(*list_id*, *user_id*)
Check if user is a subscribed of specified list

Parameters

- **list_id** – list ID number
- **user_id** – user ID number

Returns `True` if user is subscribed of list, `False` otherwise

list_members(*list_id*)
List users in a list

Parameters **list_id** – list ID number

Returns list of *User* objects

list_subscribers(*list_id*)
List subscribers of a list

Parameters **list_id** – list ID number

Returns *User* object

list_timeline(*list_id*, *since_id=None*, *max_id=None*, *count=20*)
List the tweets of specified list.

Parameters

- **list_id** – list ID number
- **since_id** – results will have ID greater than specified ID (more recent than)
- **max_id** – results will have ID less than specified ID (older than)
- **count** – number of results per page

Returns list of *Tweet* objects

lists()
List user's lists

Returns list of *List* objects

lists_memberships()
List lists which user was added

Returns list of *List* objects

lists_subscriptions()
List lists which user followed

Returns list of *List* objects

remove_list_member(*list_id*, *user_id*)
Remove a user from a list

Parameters

- **list_id** – list ID number

- **user_id** – user ID number

Returns *List* object

remove_tweet(*id*)
Delete a tweet.

Parameters **id** – ID of the tweet in question

Returns True if success, False otherwise

retweet(*id*)
Retweet a tweet.

Parameters **id** – ID of the tweet in question

Returns True if success, False otherwise

subscribe_list(*list_id*)
Subscribe to a list

Parameters **list_id** – list ID number

Returns *List* object

tweepy_api
Get the actual client object.

Returns the actual client object

tweet(*text*, *in_reply_to=None*, *filename=None*, *file=None*)
Post a new tweet.

Parameters

- **text** – the text to post
- **in_reply_to** – The ID of the tweet to reply to
- **filename** – If *file* param is not provided, read file from this path
- **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: Tweet object

unfollow(*user_id*)
Follow a user.

Parameters **user_id** – ID of the user in question

Returns The user that were unfollowed

unsubscribe_list(*list_id*)
Unsubscribe to a list

Parameters **list_id** – list ID number

Returns *List* object

update_list(*list_id*, *name=None*, *mode=None*, *description=None*)
Update a list

Parameters

- **list_id** – list ID number
- **name** – New name for the list

- **mode** – 'public' (default) or 'private'
- **description** – New description of the list

Returns The updated list object

Return type *List*

responsebot.models

Entity classes for various entity types for ResponseBot

Classes

Tweet(data)	Represents a tweet.
User(data)	Represents a user.
TweetFilter([track, follow])	Define criteria to filter tweets from Twitter's public stream.
Event(data)	Represent a user events (e.g.

class responsebot.models.**Event**(*data*)
Represent a user events (e.g. following, unfollowing, etc.). See more here and here.

__init__(*data*)

Parameters **data** (*dictionary*) – Parsed JSON data

class responsebot.models.**List**(*data*)
Represent a user list.

__init__(*data*)

Parameters **data** (*dictionary*) – Parsed JSON data

class responsebot.models.**Tweet**(*data*)
Represents a tweet. E.g. you can get a tweet's text via it's text property (tweet.text). All properties except retweeted_status, quoted_status, quoted_status_id_str, in_reply_to_status_id and in_reply_sto_status_id_str have the same name as Twitter defined them here. retweeted_status is renamed to retweeted_tweet, similar for other properties above.

__init__(*data*)

Parameters **data** (*dictionary*) – Parsed JSON data

class responsebot.models.**TweetFilter**(*track=[]*, *follow=[]*)
Define criteria to filter tweets from Twitter's public stream. See *track* and *follow* parameters in here.

__init__(*track=[]*, *follow=[]*)

Parameters

- **track** – A list of keywords to follow (each could also be a @mention or a #hashtag)
- **follow** – A list of user ID strings to follow

Returns

match_tweet(*tweet*, *user_stream*)
Check if a tweet matches the defined criteria

Parameters **tweet** (*Tweet*) – The tweet in question

Returns True if matched, False otherwise

class responsebot.models.**User**(*data*)
Represents a user. E.g. you can get a user's screen name via it's screen_name property (user.screen_name). All properties except status have the same name as Twitter defined them here. status is renamed to tweet.

__init__(*data*)

Parameters **data** (*dictionary*) – Parsed JSON data

responsebot.common.exceptions

Exceptions and errors used by Tweet Bot

Exceptions

APIError	Generic API error.
APIQuotaError	Exception indicate some API quota breached.
AuthenticationError	Error to indicate Tweet Bot failed to authenticate with the provided credentials.
MissingConfigError	Exception to indicate a required configuration for Tweet Bot is not found from config file or CLI.
NotFreeToTweetError	Error to indicate Tweeter fails to post some status due to Twitter API's status update rate limit.

exception responsebot.common.exceptions.**APIError**
Generic API error.

exception responsebot.common.exceptions.**APIQuotaError**
Exception indicate some API quota breached.

exception responsebot.common.exceptions.**AuthenticationError**
Error to indicate Tweet Bot failed to authenticate with the provided credentials.

exception responsebot.common.exceptions.**AutomatedRequestError**
Error to indicate a request is deemed automated by Twitter.

exception responsebot.common.exceptions.**CharacterLimitError**
Error to indicate your tweet reached the character limit.

exception responsebot.common.exceptions.**DailyStatusUpdateError**
Error to indicate your account reached the daily status update limit.

exception responsebot.common.exceptions.**MissingConfigError**
Exception to indicate a required configuration for Tweet Bot is not found from config file or CLI.

exception responsebot.common.exceptions.**NotFreeToTweetError**
Error to indicate Tweeter fails to post some status due to Twitter API's status update rate limit.

exception responsebot.common.exceptions.**OverCapacityError**
Error to indicate Twitter is currently over capacity.

exception responsebot.common.exceptions.**ResponseBotError**
Generic response bot error.

exception responsebot.common.exceptions.**StatusDuplicateError**
Error to indicate your status is a duplicate.

exception responsebot.common.exceptions.**UnknownAPIError**
Unknown error when executing API.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

r

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

R

S

T

U