InstaTweet

Release 2.1.1

Adam Korn

Oct 12, 2022

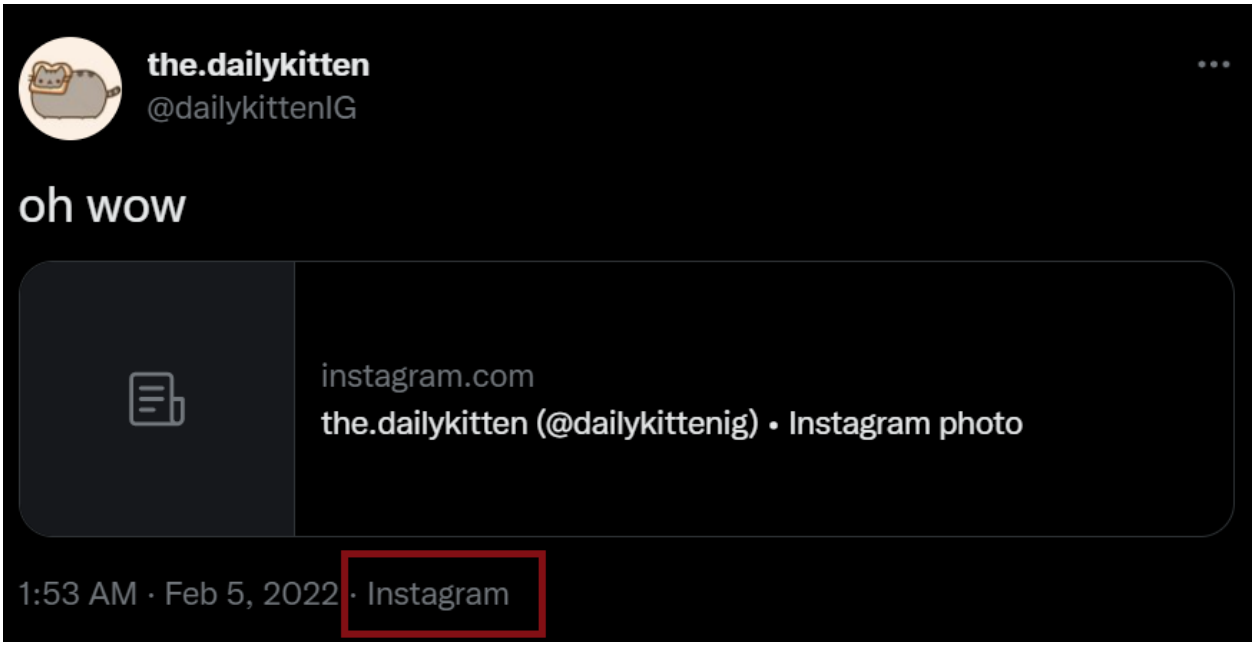
README

CHAPTER

ONE

AUTOMATICALLY REPOST CONTENT FROM INSTAGRAM TO TWITTER

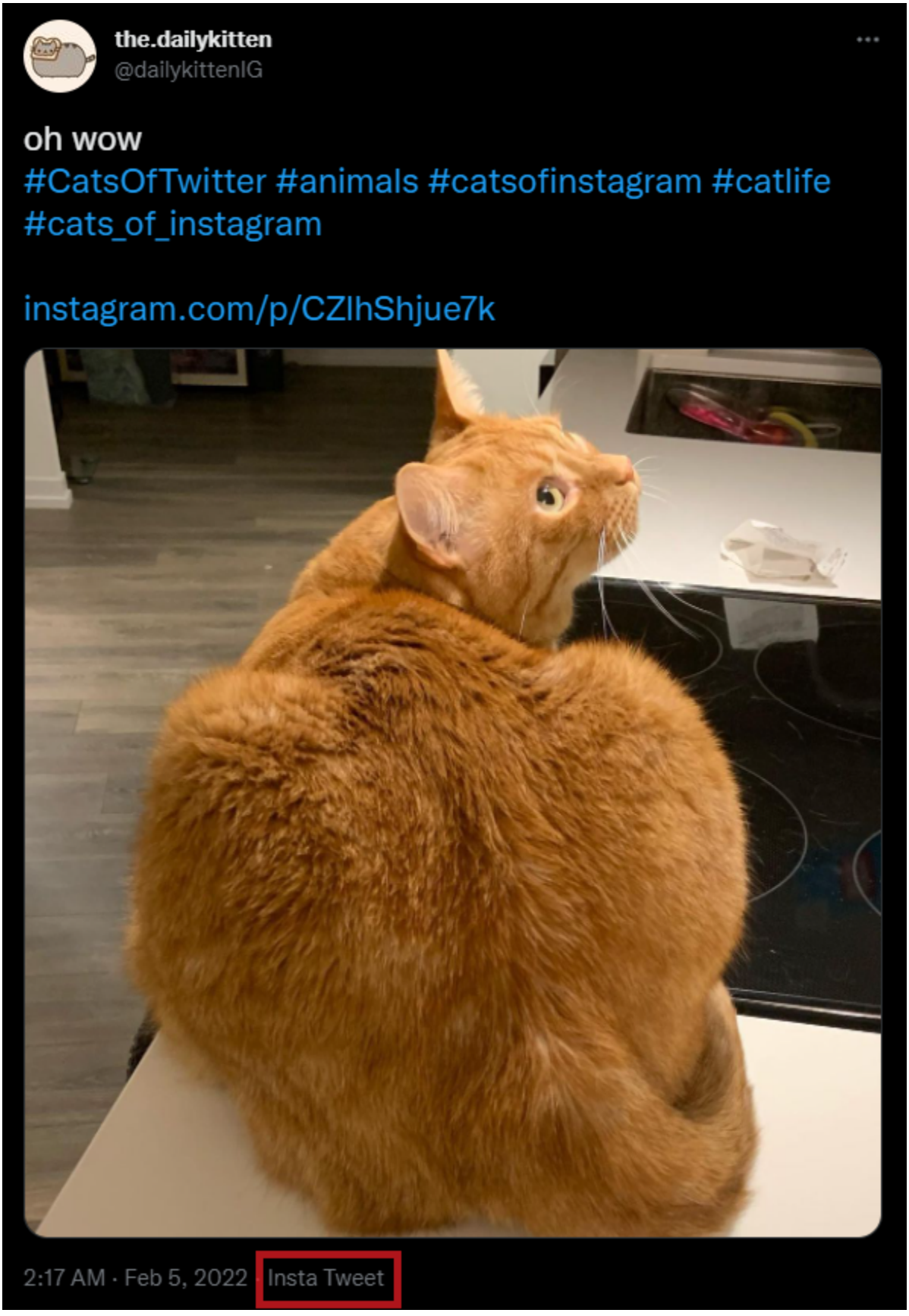
Ever tried sharing an Instagram post to Twitter, only to find out that all you tweeted was a link, and not the actual photo/video?



Humiliating

That could be literally anything. Nobody will click it.

`InstaTweet` **shares the** *actual* **content of the post. Not just a link to it.**


the.dailykitten
@dailykittenIG
oh wow
#CatsOfTwitter #animals #catsofinstagram #catlife
#cats_of_instagram
instagram.com/p/CZlhShjue7k
2:17 AM · Feb 5, 2022
Insta Tweet

With `InstaTweet`, you can rest easy knowing that, although nobody will click the link, they'll at least see what you posted.

CHAPTER

TWO

WHAT'S INSTATWEET?

InstaTweet is a customizable tool to automatically repost content from Instagram to Twitter.

Simply create a *Profile*, configure the *Mandatory Settings*, and *add_users()* to repost from

```
from InstaTweet import Profile

# Create a new (local) Profile
>>> profile = Profile('myProfile')

# Configure the required settings (at minimum)
>>> profile.twitter_keys = twitter_api_keys
>>> profile.session_id = '6011991A'

# Add at least one Instagram account to repost from
>>> profile.add_users('the.dailykitten')
```

Once configured, the *Profile* can be used to initialize and *start()* an *InstaTweet* object

```
from InstaTweet import InstaTweet

# Directly initialize with the Profile from above
>>> insta_tweet = InstaTweet profile)

# Or, save the Profile...
>>> profile.save()

Saved Local Profile myProfile

# ...then InstaTweet.load() the settings in (by Profile name)
>>> insta_tweet = InstaTweet.load profile_name="myProfile")

# Run InstaTweet by calling start()
>>> insta_tweet.start()
```

From the Docs...

InstaTweet.**start**()

InstaTweets all users that have been added to the loaded *Profile*

Each user's IG page will be scraped and compared to the `scraped` list in their *USER_MAPPING*. Posts that weren't previously scraped will be downloaded and tweeted

Note: If `InstaTweet` fails to *`download_post()`* or *`send_tweet()`*, the *`USER_MAPPING`* won't be updated

- This ensures that failed repost attempts are retried in the next call to *`start()`*

If a save file for the Profile already *`exists`*, successful reposts will trigger a call to *`save()`*

As `InstaTweet` runs, its progress will be logged to console:

```
Starting InstaTweet for Profile: myProfile
Checking posts from @the.dailykitten
...
Finished insta-tweeting for @the.dailykitten
All users have been insta-tweeted
```

CHAPTER

THREE

OKAY... BUT WHY?

But Why?

InstaTweet has two main use cases:

- To automatically share your own Instagram posts to Twitter
- To automatically tweet new content from other Instagram users

Regardless of your intention, InstaTweet will detect new posts from the users you specify, download them, and repost them to Twitter.

CHAPTER

FOUR

DOCUMENTATION

The rest of this README, the API documentation, and snippets can all be found on Read the Docs

I put a lot of time into creating the documentation for this package, it was a struggle, so it'd mean a lot to me if you could please continue reading there!

CHAPTER

FIVE

INSTALLATION

To install using pip:

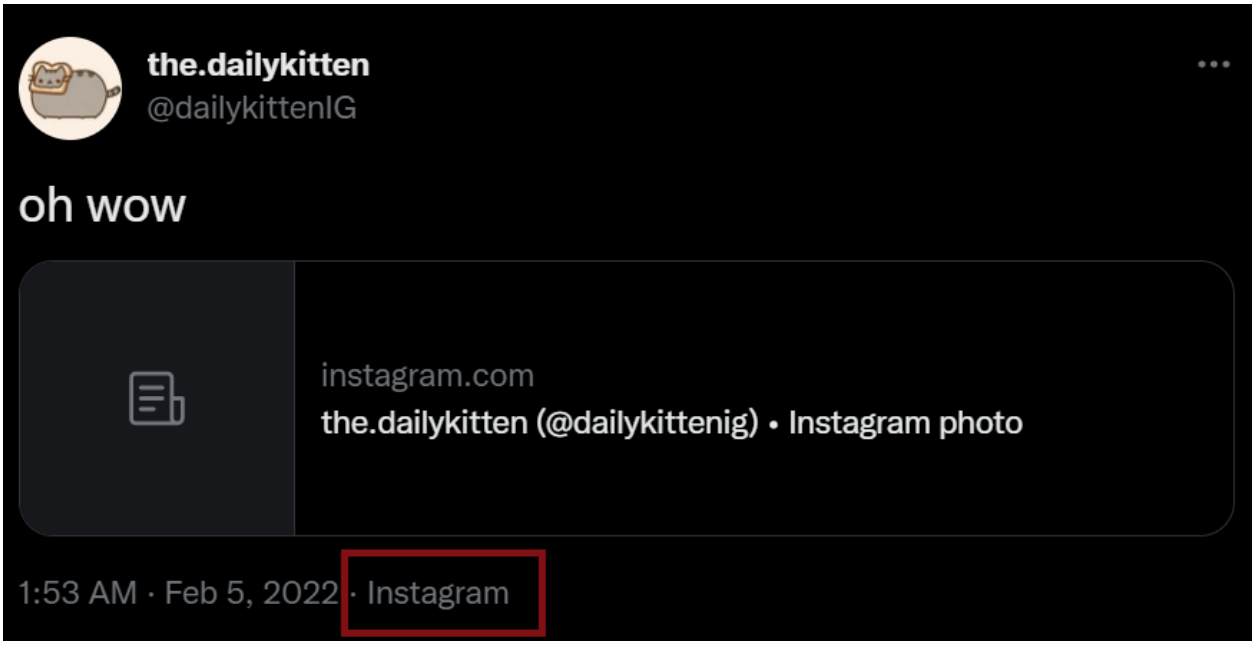
```
pip install insta-tweet
```

Please note that `InstaTweet` requires `Python >= 3.8`

5.1 InstaTweet

5.1.1 Automatically Repost Content From Instagram to Twitter

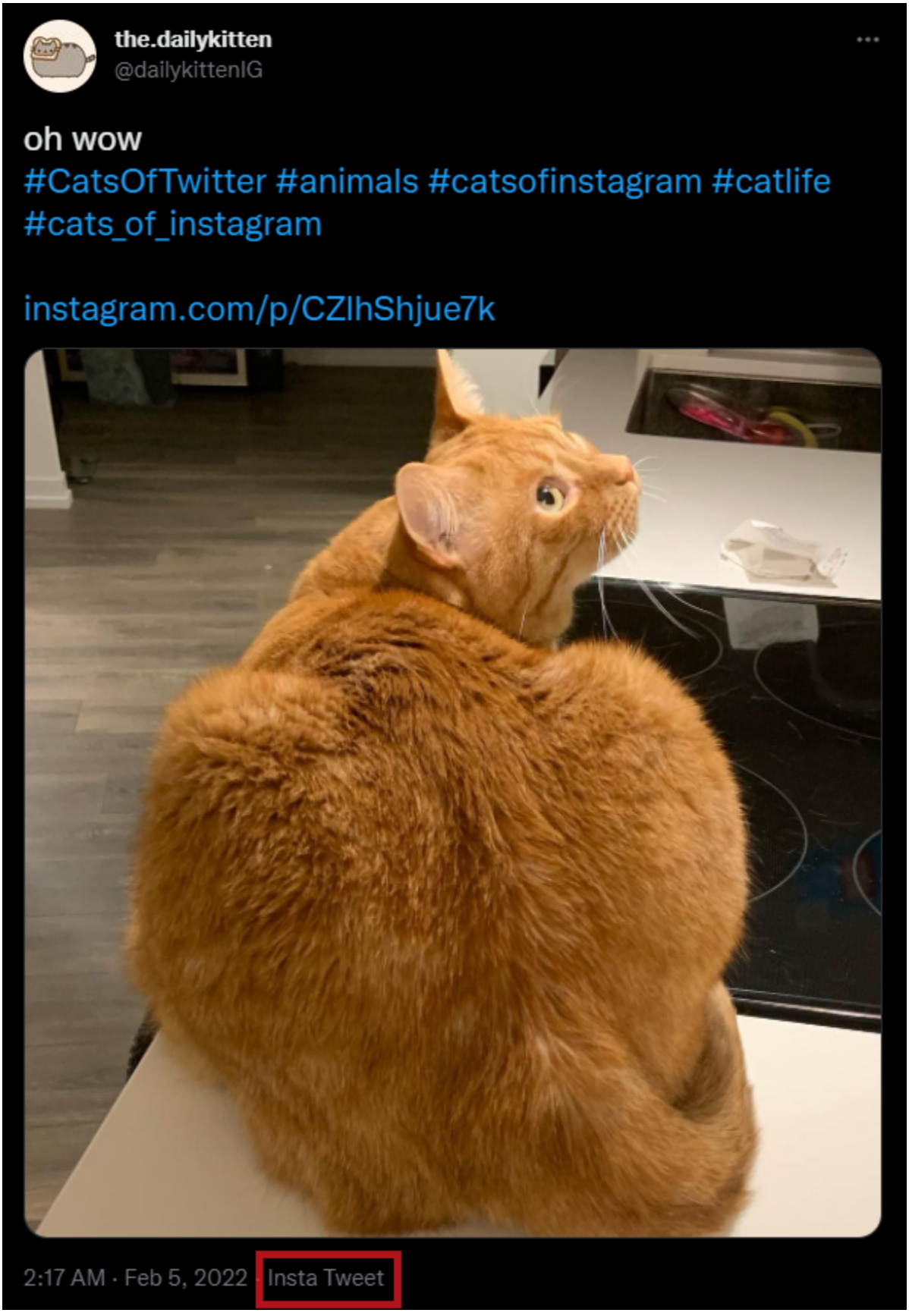
Ever tried sharing an Instagram post to Twitter, only to find out that all you tweeted was a link, and not the actual photo/video?



Humiliating

That could be literally anything. Nobody will click it.

`InstaTweet` **shares the** *actual* **content of the post. Not just a link to it.**


the.dailykitten
@dailykittenIG
oh wow
#CatsOfTwitter #animals #catsofinstagram #catlife
#cats_of_instagram
instagram.com/p/CZlhShjue7k
2:17 AM · Feb 5, 2022
Insta Tweet

With InstaTweet, you can rest easy knowing that, although nobody will click the link, they'll at least see what you posted.

5.1.2 What's InstaTweet?

InstaTweet is a customizable tool to automatically repost content from Instagram to Twitter.

Simply create a *Profile*, configure the *Mandatory Settings*, and *add_users()* to repost from

```
from InstaTweet import Profile

# Create a new (local) Profile
>>> profile = Profile('myProfile')

# Configure the required settings (at minimum)
>>> profile.twitter_keys = twitter_api_keys
>>> profile.session_id = '6011991A'

# Add at least one Instagram account to repost from
>>> profile.add_users('the.dailykitten')
```

Once configured, the *Profile* can be used to initialize and *start()* an *InstaTweet* object

```
from InstaTweet import InstaTweet

# Directly initialize with the Profile from above
>>> insta_tweet = InstaTweet profile

# Or, save the Profile...
>>> profile.save()

Saved Local Profile myProfile

# ...then InstaTweet.load() the settings in (by Profile name)
>>> insta_tweet = InstaTweet.load profile_name="myProfile"

# Run InstaTweet by calling start()
>>> insta_tweet.start()
```

From the Docs...

InstaTweet.**start**()

InstaTweets all users that have been added to the loaded *Profile*

Each user's IG page will be scraped and compared to the `scraped` list in their *USER_MAPPING*. Posts that weren't previously scraped will be downloaded and tweeted

---

Note: If InstaTweet fails to *download_post()* or *send_tweet()*, the *USER_MAPPING* won't be updated

- This ensures that failed repost attempts are retried in the next call to *start()*

If a save file for the Profile already *exists*, successful reposts will trigger a call to *save()*

As `InstaTweet` runs, its progress will be logged to console:

```
Starting InstaTweet for Profile: myProfile
Checking posts from @the.dailykitten
...
Finished insta-tweeting for @the.dailykitten
All users have been insta-tweeted
```

5.1.3 Okay... But Why?

But Why?

InstaTweet has two main use cases:

- To automatically share your own Instagram posts to Twitter
- To automatically tweet new content from other Instagram users

Regardless of your intention, InstaTweet will detect new posts from the users you specify, download them, and repost them to Twitter.

5.1.4 Documentation

The rest of this README, the API documentation, and snippets can all be found on Read the Docs

I put a lot of time into creating the documentation for this package, it was a struggle, so it'd mean a lot to me if you could please continue reading there!

5.1.5 Installation

To install using pip:

```
pip install insta-tweet
```

Please note that `InstaTweet` requires `Python >= 3.8`

5.2 Getting Started

5.2.1 InstaTweet Profiles

InstaTweet uses the *Profile* class to help manage Twitter accounts, Instagram sessions, and user maps.

class InstaTweet.profile.**Profile**(*name='default'*, *local=True*, ***kwargs*)

The *Profile* is a configuration class used extensively throughout the package

It consists of a *user_map* and an associated collection of API/web scraping *settings*

...

About the User Map

The *user_map* is a dict containing info about the users added to a *Profile*

- It's used to help detect new posts and compose tweets on a per-user basis
- Entries are created when you *add_users()*, which map the user to a *USER_MAPPING*
- The *USER_MAPPING* maintains lists of hashtags, scraped posts, and sent tweets
- The mapping is updated when you *add_hashtags()* and successfully *send_tweet()*

You can access entries in the *user_map* as follows:

- *get_user()* allows you to retrieve a full entry by username
- *get_hashtags_for()*, *get_scraped_from()*, *get_tweets_for()* provide access to lists

...

[Optional]

A unique, identifying *name* can be assigned to the Profile, which may then be used to *save()* and, in turn, *load()* its settings

- This makes it extremely easy to switch between Profiles and create templates

Saving isn't a requirement to *start()* InstaTweet, but...

- To *get_new_posts()*, InstaTweet makes comparisons with the `scraped` list in the *user_map*
- Saving this list ensures you don't *send_tweet()* for a post more than once

...

Important

If you do *save()* your profile, the save location is determined by the value of *Profile.local*

- Local saves are made to the *LOCAL_DIR*, as pickle files
- Remote saves are made to a database (via the *db* module) as pickle bytes

You MUST configure the *DATABASE_URL* environment variable to save/load remotely

- InstaTweet uses `SQLAlchemy` to create a *DBConnection* – any db it supports is compatible
- See the *db* module for more information

5.2.2 Profile Settings

All settings can be configured in two ways:

1. By passing them as keyword arguments when initializing a *Profile*
2. By setting them directly as object attributes after the *Profile* object is created

Mandatory Settings

- *session_id* — Instagram sessionid cookie, obtained by logging in on a desktop browser
- *twitter_keys* — Twitter API keys with v1.1 endpoint access

Mandatory Settings with Default Values

- *name* (="default") — the profile name; if non-default, it must be unique
- *local* (=True) — indicates if the profile should be saved locally (default) or on a remote database
- user_agent=USER_AGENT — user agent to use when making requests to Instagram; currently hardcoded

Entirely Optional Settings

- proxy_key — Environment variable to retrieve proxies from when making requests to Instagram/Twitter
- *user_map* — Fully formatted dictionary of IG usernames mapped to their USER_MAPPING

5.2.3 Creating a Profile

```
from InstaTweet import Profile

# Initialize a profile with arguments
p = Profile(
    name='myProfile',
    session_id='6011991A'
)

# Initialize a profile with no arguments
q = Profile()
q.name = 'myProfile'
q.session_id = '6011991A'
```

All settings can be accessed via the *config* dict. If you just want to look, call *view_config()*

```
# View and compare configuration settings
>>> q.view_config()
>>> print(f'Same Config: {p.config==q.config}')
```

Output:

```
name : myProfile
local : True
session_id : 6011991A
twitter_keys : {'Consumer Key': 'string', 'Consumer Secret': 'string', 'Access Token':
↪'string', 'Token Secret': 'string'}
user_agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like␣
↪Gecko) Chrome/102.0.5005.63 Safari/537.36
proxy_key : None
user_map : {}
Same Config: True
```

...

Validating Profile Settings

- Property setters validate data types for the *Mandatory Settings*
- Requirements aren't strictly enforced until *start()* is called, which will first *validate()* the profile settings

5.2.4 Populating the User Map

The User Map

The *user_map* allows a *Profile* to maintain a history of package-related activity for its added IG users

Users are mapped to their *USER_MAPPING*, which contains their associated lists of:

```
USER_MAPPING = {'hashtags': [], 'scraped': [], 'tweets': []}
```

- `hashtags` — the user's associated hashtag list (for use when composing tweets)
- `scraped` — the list of posts that have been scraped from the user (only the post id)
- `tweets` — the list of sent tweets containing media scraped from that user (limited data)

The mapping gets updated each time *InstaTweet* successfully scrapes and tweets a post from the user

Adding Users

Use the *add_users()* method to add one or more Instagram users to a *Profile*'s *user_map*

```
from InstaTweet import Profile

# Add one user at a time
>>> p = Profile('myProfile')
>>> p.add_users('the.dailykitten', send_tweet=True)

Added Instagram user @the.dailykitten to the user map

# Add multiple users at once
>>> usernames = ['dailykittenig', 'the.daily.kitten.ig']
>>> p.add_users usernames)
```

(continues on next page)

(continued from previous page)

```
Added Instagram user @dailykittenig to the user map
Added Instagram user @the.daily.kitten.ig to the user map
```

The *get_user()* method can be used to retrieve the full *USER_MAPPING* of an added user

```
>> p.get_user('the.dailykitten')

{'hashtags': [], 'scraped': [-1], 'tweets': []}
```

Adding Hashtags

You can *add_hashtags()* for each user in the *user_map*

- They'll be chosen from at random when composing tweets based on one of their *posts*
- For more info, see *pick_hashtags()*, *build_tweet()* and *send_tweet()*

```
# Add a single hashtag for a specific user
>>> p.add_hashtags(user='dailykittenig', hashtags='cats')

Added hashtags for @dailykittenig

# Add multiple hashtags at once
>>> users = ['the.dailykitten','the.daily.kitten.ig']
>>> hashtags = ['kittygram', 'kittycat']

>>> for user in users:
...     p.add_hashtags(user, hashtags)

Added hashtags for @the.dailykitten
Added hashtags for @the.daily.kitten.ig

>>> p.view_config()
```

Output:

```
name : myProfile
local : True
session_id : 6011991A
twitter_keys : {'Consumer Key': 'string', 'Consumer Secret': 'string', 'Access Token':
↪'string', 'Token Secret': 'string'}
user_agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like␣
↪Gecko) Chrome/102.0.5005.63 Safari/537.36
proxy_key : None
user_map : {'the.dailykitten': {'hashtags': ['kittygram', 'kittycat'], 'scraped': [-1],
↪'tweets': []}, 'dailykittenig': {'hashtags': ['cats'], 'scraped': [], 'tweets': []},
↪'the.daily.kitten.ig': {'hashtags': ['kittygram', 'kittycat'], 'scraped': [], 'tweets
↪': []}}
```

User Map Access Methods

User Map Access Methods

The *Profile* has several methods that allow for easy access to the *user_map*

- *get_user()* provides access to a particular user's *USER_MAPPING*
- *get_scraped_from()* returns the list of posts scraped from a specified user
- *get_hashtags_for()* returns the list of hashtags to use in tweets for the specified user
- *get_tweets_for()* returns a list of tweets that use the specified user's scraped content

All lists returned by these methods can be modified in place. For example:

```
# View the list of hashtags by username
>> print p.get_hashtags_for('the.daily.kitten.ig')

['kittygram', 'kittycat']

# Retrieve and modify the list
>> p.get_hashtags_for('the.daily.kitten.ig').append('kittypoop')
>> print p.get_hashtags_for('the.daily.kitten.ig')

['kittygram', 'kittycat', 'kittypoop']
```

5.2.5 Saving a Profile

Saving a Profile

When you *save()* your *Profile*, the current or specified *name* will be used to create or update a save file in the location specified by *local*

From the Docs...

`Profile.`**`save`**(*name=None*, *alert=True*)

Pickles and saves the *Profile* using the specified or currently set name.

Parameters

- **name** (*Optional[str]*) – name to save the *Profile* under; replaces the current *name*
- **alert** (*bool*) – set to `True` to print a message upon successful save

Return type
bool

`InstaTweet.profile.Profile.`**`local =True`**

Indicates if saves should be made locally (`True`) or on a remote database (`False`)

Return type
bool

- Local saves are made to the *LOCAL_DIR*, as pickle files

- Remote saves are made to a database (via the *db* module) as pickle bytes

---

Important!!

You MUST configure the *DATABASE_URL* environment variable to save/load remotely

InstaTweet uses `SQLAlchemy` to create a *DBConnection*

- Any `SQLAlchemy`-supported database is therefore also supported by `InstaTweet`
- See the *db* module for more information

Although you don't *need* to *save()* the Profile to *start()* InstaTweet, it's highly suggested since:

- It's an easy way to group API settings together
- It keeps track of previously scraped & tweeted posts, which is used to detect new posts

Example: Save a Profile

Note: You can specify a new *name* for the profile in the call to *save()*

```
from InstaTweet import Profile

>>> p = Profile('myProfile')
>>> p.save()

Saved Local Profile myProfile

>>> q = Profile()
>>> q.save('aProfile')

Saved Local Profile aProfile

# Try to save under a name that's already used...
>>> q.save('myProfile')

FileExistsError: Local save file already exists for profile named "myProfile"
Please choose another name, load the profile, or delete the file.

>>> Profile.profile_exists("aProfile")
True
```

5.3 Schedule InstaTweet

You can easily schedule InstaTweet using the provided scheduler script

```
#scheduler.py
from InstaTweet import InstaTweet

PROFILES = ['aProfile', 'myProfile']
LOCAL = True


def run(profile_name: str, local: bool = LOCAL):
    """Loads and InstaTweets a profile

    :param profile_name: the name of the :class:`~.Profile`
    :param local: if the profile is saved locally or in a SQLAlchemy supported database
    """
    insta_tweet = InstaTweet.load(profile_name, local=local)
    insta_tweet.start()


if __name__ == '__main__':
    for profile in PROFILES:
        run(profile, local=LOCAL)
```

CHAPTER

SIX

INSTATWEET README

6.1 InstaTweet Package

Below, you'll find the documentation for each class/module in the `InstaTweet` package.

6.1.1 The `InstaTweet` class

Not to be confused with the package, the `InstaTweet` class is the main point of interaction for a user of the package

- It uses a fully configured *`Profile`* to initialize an *`InstaClient`* and *`TweetClient`*, which are used to scrape and tweet posts when *`start()`* is called

...

class InstaTweet.instatweet.**InstaTweet**(*profile*)

Bases: `object`

Uses the settings from a Profile to do the actual InstaTweeting

You might be wondering, what's InstaTweeting? According to TDK Dictionary:

InstaTweet (*verb*):

To load a *`Profile`* scrape *`posts`* from its Instagram users *`download_post()`* & *`send_tweet()`* for any new content update the *`user_map`* *`save()`* the profile if it *`exists`*

Example Sentence

Oh, you lost 700 Twitter followers after you shared your IG post? Well maybe if people actually saw the picture and not just the caption your tweet would've been less creepy. You should've InstaTweeted it.

__init__(*profile*)

Initializes *InstaTweet* using a fully configured *Profile*

The *Profile* will be used to initialize an *InstaClient* and *TweetClient*

Note: InstaTweet won't *validate()* the Profile settings until you call *start()*

Parameters
profile (Profile) – the *Profile* to use for InstaTweeting

classmethod load(*profile_name*, *local=True*)

Loads a profile by name

Parameters

- **profile_name** (*str*) – name of the Profile to load
- **local** (*bool*) – whether the profile is saved locally (default) or remotely on a database

Return type
InstaTweet

get_proxies()

Retrieve proxies using the loaded Profile's proxy_key

Return type
Optional[dict]

get_insta_client()

Initializes an *InstaClient* using the loaded *Profile* settings

Return type
InstaClient

get_tweet_client()

Initializes an *TweetClient* using the loaded *Profile* settings

Return type
TweetClient

start()

InstaTweets all users that have been added to the loaded *Profile*

Each user's IG page will be scraped and compared to the scraped list in their *USER_MAPPING*. Posts that weren't previously scraped will be downloaded and tweeted

Note: If InstaTweet fails to *download_post()* or *send_tweet()*, the *USER_MAPPING* won't be updated

- This ensures that failed repost attempts are retried in the next call to *start()*

If a save file for the Profile already *exists*, successful reposts will trigger a call to *save()*

get_new_posts(*username*)

Scrapes recent posts from an Instagram user and returns all posts that haven't been tweeted yet

NOTE: If a user's `scraped` list is empty, no posts will be returned.

Instead, the user is "initialized" as follows:

- Their `scraped` list will be populated with the ID's from the most recent posts
- These IDs are then used in future calls to the method to determine which posts to tweet

Parameters
username – the IG username to scrape posts from

Returns
a list of posts that haven't been tweeted yet, or nothing at all (if user is only initialized)

Return type
Optional[*List*[InstaPost]]

6.1.2 The `Profile` class

class InstaTweet.profile.**Profile**(*name='default'*, *local=True*, ***kwargs*)

Bases: `object`

The *`Profile`* is a configuration class used extensively throughout the package

It consists of a *`user_map`* and an associated collection of API/web scraping *settings*

...

About the User Map

The *`user_map`* is a dict containing info about the users added to a *`Profile`*

- It's used to help detect new posts and compose tweets on a per-user basis
- Entries are created when you *`add_users()`*, which map the user to a *`USER_MAPPING`*
- The *`USER_MAPPING`* maintains lists of hashtags, scraped posts, and sent tweets
- The mapping is updated when you *`add_hashtags()`* and successfully *`send_tweet()`*

You can access entries in the *`user_map`* as follows:

- *`get_user()`* allows you to retrieve a full entry by username
- *`get_hashtags_for()`*, *`get_scraped_from()`*, *`get_tweets_for()`* provide access to lists

...

[Optional]

A unique, identifying *`name`* can be assigned to the Profile, which may then be used to *`save()`* and, in turn, *`load()`* its settings

- This makes it extremely easy to switch between Profiles and create templates

Saving isn't a requirement to *`start()`* InstaTweet, but...

- To *`get_new_posts()`*, InstaTweet makes comparisons with the `scraped` list in the *`user_map`*
- Saving this list ensures you don't *`send_tweet()`* for a post more than once

...

Important

If you do *save()* your profile, the save location is determined by the value of *Profile.local*

- Local saves are made to the *LOCAL_DIR*, as pickle files
- Remote saves are made to a database (via the *db* module) as pickle bytes

You MUST configure the *DATABASE_URL* environment variable to save/load remotely

- InstaTweet uses `SQLAlchemy` to create a *DBConnection* – any db it supports is compatible
- See the *db* module for more information

USER_MAPPING = {'hashtags': [], 'scraped': [], 'tweets': []}

Template for an entry in the `user_map`

LOCAL_DIR = '/home/docs/checkouts/readthedocs.org/user_builds/instatweet/checkouts/develop/docs/source/profiles'

Directory where local profiles are saved

__init__(*name='default'*, *local=True*, ***kwargs*)

Create a new *Profile*

Note: *Profile* creation is mandatory to use the `InstaTweet` package

- Required as a parameter to initialize an *InstaTweet* object
- Naming and saving it is ideal, but not necessary to *start()* InstaTweet

---

Parameters

- **name** (*str*) – unique profile name
- **local** (*bool*) – indicates if profile is being saved locally or on a remote database
- **kwargs** – see below

Keyword Arguments

- ***session_id*** **(str)**
  Instagram `sessionid` cookie, obtained by logging in through browser
- ***twitter_keys*** **(dict)**
  Twitter API Keys with v1.1 endpoint access (see *DEFAULT_KEYS* for a template)
- ***user_agent*** **(str) – Optional**
  The user agent to use for requests; uses a currently working hardcoded agent if not provided
- ***proxy_key*** **(str) – Optional**
  Environment variable to retrieve proxies from

- **user_map**
  dict: Mapping of added Instagram users and their *USER_MAPPING*

Profile Creation Tips

- All attributes can be passed as arguments at initialization or set directly afterwards
- Property setters validate data types for the *Mandatory Settings*
- The *Profile* as a whole is validated by *validate()*

---

user_map
dict: Mapping of added Instagram users and their *USER_MAPPING*

classmethod load(*name*, *local=True*)
Loads an existing profile from a locally saved pickle file or remotely stored pickle bytes

Parameters

- **name** (*str*) – the name of the *Profile* to load
- **local** (*bool*) – whether the profile is saved locally (default, True) or remotely on a database

Return type
Profile

classmethod from_json(*json_str*)
Creates a profile from a JSON formatted string of config settings

Return type
Profile

classmethod from_dict(*d*)
Creates a profile from a dictionary of config settings

Return type
Profile

static profile_exists(*name*, *local=True*)
Checks locally/remotely to see if a *Profile* with the specified name has an existing save file

Whenever the *name* is changed, its property setter calls this method to ensure you don't accidentally overwrite a save that already *exists*

Parameters

- **name** (*str*) – the name of the *Profile* to check for
- **local** (*bool*) – the location (local/remote) to check for an existing save

Return type
bool

static get_local_path(*name*)
Returns filepath of where a local profile would be saved

Return type
str

add_users(*users*, *send_tweet=False*)

Add Instagram user(s) to the *user_map* for subsequent monitoring

---

Note: By default, newly added users won't have their posts tweeted the first time they're scraped

- The IDs of the ~12 most recent posts are stored in the `scraped` list
- Any new posts from that point forward will be tweeted

You can override this by setting `send_tweet=True`

- This causes their ~12 most recent posts to be scraped AND tweeted

---

Parameters

- **users** (*Iterable*) – Instagram username(s) to automatically scrape and tweet content from
- **send_tweet** (*bool*) – choose if tweets should be sent on the first scrape, or only for new posts going forward

add_hashtags(*user*, *hashtags*)

Add hashtag(s) to a user in the *user_map*, which will be randomly chosen from when composing Tweets

Parameters

- **user** (*str*) – the user in the user map to add hashtags to
- **hashtags** (*Iterable*) – hashtags to choose from and include in any Tweets where content comes from this user

save(*name=None*, *alert=True*)

Pickles and saves the *Profile* using the specified or currently set name.

Parameters

- **name** (*Optional[str]*) – name to save the *Profile* under; replaces the current *name*
- **alert** (*bool*) – set to `True` to print a message upon successful save

Return type
bool

validate()

Checks to see if the Profile is fully configured for InstaTweeting

Raises
ValueError – if the *session_id*, *twitter_keys*, or *user_map* are invalid

to_pickle()

Serializes profile to a pickled byte string

Return type
bytes

to_json()

Serializes profile to a JSON formatted string

Return type
str

`to_dict()`

Serializes profile to a dict

Return type
dict

`view_config()`

Prints the *config* dict to make it legible

`property config: dict`

Returns a dictionary containing important configuration settings

`property exists: bool`

Returns True if a local save file or database record exists for the currently set profile name

`property is_default: bool`

Check if profile *name* is set or not

`property profile_path: str`

If *local* is `True`, returns the file path for where this profile would be/is saved

`get_user(user)`

Returns the specified user's dict entry in the *user_map*

Return type
dict

`get_scraped_from(user)`

Returns a list of posts that have been scraped from the specified user

Return type
list

`get_tweets_for(user)`

Returns a list of tweets that use the specified user's scraped content

Return type
list

`get_hashtags_for(user)`

Returns the hashtag list for the specified user

Return type
list

`property local: bool`

Indicates if saves should be made locally (`True`) or on a remote database (`False`)

Return type
bool

`property name: str`

A name for the Profile

The *name* is used differently depending on the value of *local*

- `local==True`: the name determines the *profile_path* (path where it would save to)
- `local==False`: the name is used as the primary key in the *Profiles* database table

...

Profile Names Must Be Unique

When you set or change the *name*, a property setter will make sure no *profile_exists()* with that name before actually updating it

- This ensures that you don't accidentally overwrite a different Profile's save data

...

Raises

- **FileExistsError** – if *local* ==True and a save is found in the *LOCAL_DIR*
- **ResourceWarning** – if *local* ==False and a database row is found by query_profile()

property session_id: str

Instagram sessionid cookie, obtained by logging in through a browser

Tip

If you log into your account with a browser you don't use, the session cookie will last longer

property twitter_keys: dict

Twitter developer API keys with v1.1 endpoint access. See *DEFAULT_KEYS*

6.1.3 The db module

The *db* module contains the *DBConnection* class and the *Profiles* database table

The Database Table

In the *Profiles* database table each row corresponds to a unique *Profile*

The table only has two fields per row:

- *name*: primary key for lookups/insertions
- *config*: stores the Profile as pickle bytes via *to_pickle()*

How is profile data saved to the database?

When a *Profile* calls *save()* and has *local* = False, it will *connect()* to the database specified by the DABATASE_URL environment variable and use it to *query_profile()* settings

- If the *profile_exists()* in the database already, its *config* data will be updated
- Otherwise, the *DBConnection* will *save_profile()* data in a new table row

Important!!

You MUST configure the *DATABASE_URL* environment variable to save/load remotely

- InstaTweet uses SQLAlchemy to create a *DBConnection* – any db it supports is compatible
- See the *db* module for more information

One Last Thing!

The *DBConnection* is meant to be used as a context manager

```
with DBConnection() as db:
    # Do Something
```

- A SESSION is created/destroyed when saving, loading, and InstaTweeting a *Profile*

If you don't want that, here's instructions on *Persisting The DBConnection*

...

InstaTweet.db.**DATABASE_URL**

The Database URL to use, obtained from the DATABASE_URL environment variable

...

class InstaTweet.db.**Profiles**(***kwargs*)

Database table used for storing *Profile* settings

The table currently has only 2 fields, for the *name* and pickle bytes of the profile

name

The *Profile* name

config

The pickle bytes from *Profile.to_pickle()*

...

class InstaTweet.db.**DBConnection**

Database Connection class with context management ooh wow

Uses SQLAlchemy to connect and interact with the database specified by *DATABASE_URL* environment variable

Sample Usage:

```
def poop_check():
    with DBConnection() as db
        if db.query_profile name="POOP").first():
            raise FileExistsError('DELETE THIS NEPHEW.....')
```

SESSION

The currently active session; closed on object exit

Type
scoped_session

ENGINE

The engine for the currently set *DATABASE_URL*; reused after first connection

Type
Engine

static connect()

Creates a scoped_session and assigns it to *DBConnection.SESSION*

query_profile(*name*)

Queries the database for a *Profile* by its name

Parameters
name (*str*) – the profile name (ie. the *Profile.name*)

Returns
the Query NOT the *Profile*

Return type
Query

load_profile(*name*)

Loads a profile from the database by name

Parameters
name (*str*) – the profile name (ie. the *Profile.name*)

Raises
LookupError – if the database has no profile saved with the specified name

Return type
Profile

save_profile(*profile*, *alert=True*)

Saves a *Profile* to the database by either updating an existing row or inserting a new one

Parameters

- **profile** (Profile) – the *Profile* to save
- **alert** (*bool*) – if True, will print a message upon successfully saving

Return type
bool

delete_profile(*name*, *alert=True*)

Deletes a *Profile* from the database by name

Parameters

- **name** (*str*) – the profile name (ie. the *Profile.name*)
- **alert** (*bool*) – if True, will print a message upon successfully deleting

Return type
bool

6.1.4 The `TweetClient` class

class InstaTweet.tweetclient.**TweetClient**(*profile*, *proxies=None*)

Bases: `object`

MAX_HASHTAGS = 5

DEFAULT_KEYS = {'Access Token': 'string', 'Consumer Key': 'string', 'Consumer Secret': 'string', 'Token Secret': 'string'}

__init__(*profile*, *proxies=None*)

Initialize TweetClient using a *Profile*

Basically just a wrapper for tweepy. It uses the settings of a profile to initialize the API and send tweets

Parameters

- **profile** (`Profile`) – the profile to use when initializing a `tweepy.API` object
- **proxies** (*Optional[dict]*) – optional proxies to use when making API requests

get_api()

Initializes a `API` object using the API keys of the loaded *Profile*

Return type
API

static get_oauth(*api_keys*)

Initializes and returns an `OAuth1UserHandler` object from tweepy using the specified API keys

Parameters
api_keys (*dict*) – Twitter developer API keys with v1.1 endpoint access

Return type
OAuth1UserHandler

send_tweet(*post*, *hashtags=None*)

Composes and sends a Tweet using an already-downloaded Instagram post

Parameters

- **post** (`InstaPost`) – the post to tweet
- **hashtags** (*Optional[list[str]]*) – a list of hashtags to randomly chose from and include in the tweet

Return type
bool

How Tweets are Composed and Sent

The *InstaPost.filepath* – set by *download_post()* – is used as the media source

The body of the tweet is then generated by *build_tweet()* as folows:

- The `InstaPost.caption` is used as a starting point
- If you've *add_hashtags()* for the user, will randomly *pick_hashtags()* to include
- Lastly, the `InstaPost.permalink` is added to the end

`upload_media`(*post*)

Uploads the media from an already-downloaded Instagram post to Twitter

Note: If the post is a carousel, only the first 4 photos/videos will be uploaded

Parameters
post (InstaPost) – the Instagram post to use as the media source

Returns
the list of uploaded media ids (if API upload was successful) or `False`

Return type
Union[list, bool]

`build_tweet`(*post*, *hashtags=None*)

Uses an *InstaPost* to build the body text of a tweet

Parameters

- **post** (InstaPost) – the post that's being tweeted; the caption and link are used
- **hashtags** (*Optional[list[str]]*) – optional list of hashtags to randomly pick from and include

Returns
the text to use for the tweet

Return type
str

`static pick_hashtags`(*hashtags*)

Randomly picks hashtags from the provided list and returns them as a single string

The number of hashtags chosen will either be 1 less than the length of the list (to avoid using the same tags in every tweet), or the value of *MAX_HASHTAGS*, whichever is smaller

Parameters
hashtags (*list[str]*) – a list of hashtags to randomly choose from

Example

```
from InstaTweet import TweetClient

>> TweetClient.pick_hashtags(['cat','dog','woof'])
"#woof #cat\n"
```

Return type
str

Note: A newline is added to help with formatting & character counting in *build_tweet()*

6.1.5 The InstaClient class

InstaTweet.instaclient.**USER_AGENT**

Hardcoded user agent proven to work with the *get_user()* endpoint introduced in v2.0.0b13

InstaClient.**DOWNLOAD_DIR**

[*Optional*] – Directory to temporarily download media to

class InstaTweet.instaclient.**InstaClient**(*session_id*, *user_agent=USER_AGENT*, *proxies=None*)

Bases: object

Minimalistic class for scraping/downloading Instagram user/media data

__init__(*session_id*, *user_agent=USER_AGENT*, *proxies=None*)

Initialize an *InstaClient* with an Instagram sessionid cookie (at minimum)

Note: As of v2.0.0b13, the endpoint used by *get_user()* seems to require a specific *USER_AGENT* You can override the hardcoded one if you'd like, but you'll likely get a "useragent mismatch" response

---

Parameters

- **session_id** (*str*) – valid Instagram sessionid cookie from a browser
- **user_agent** (*str*) – user agent to use in requests made by the class
- **proxies** (*Optional[dict]*) – proxies to use in requests made by the class

request(*url*)

Sends a request using the *cookies*, *headers*, and proxies

Parameters
url (*str*) – the Instagram URL to send the request to

Return type
Response

get_user(*username*)

Scrapes an Instagram user's profile and wraps the response

Parameters
username (*str*) – the username of the IG user to scrape (without the @)

Returns
an *InstaUser* object, which wraps the response data

Return type
InstaUser

download_post(*post*, *filepath=None*)

Downloads the media from an Instagram post

Parameters

- **post** (InstaPost) – the *InstaPost* of the post to download
- **filepath** (*Optional[str]*) – the path to save the downloaded media; if None, saves to the *DOWNLOAD_DIR*

Return type
bool

property headers: dict
Headers to use in *request()*

property cookies: dict
Cookies to use in *request()*

6.1.6 The InstaUser class

class InstaTweet.instauser.**InstaUser**(*data*, *client=None*)
Bases: object

Minimalistic API response wrapper for an Instagram profile

__init__(*data*, *client=None*)
Initialize an *InstaUser*

Parameters
- **data** (*dict*) – the API response JSON to use as source data
- **client** (*Optional[InstaClient]*) – API client to use; only required for *get_more_posts()*

property id: int
Instagram User ID

property posts: [InstaPost]
Returns the list of posts scraped from the Instagram user

property media_data: dict

property user_data: dict

get_more_posts()
Requests the next page of posts

If the user *has_more_posts*, they'll be added to the *posts* list

Returns
`True` if the request was successful, otherwise `False`

Return type
bool

property has_more_posts: bool
Returns `True` if more posts can be scraped using *get_more_posts()*

property end_cursor: str
Cursor used in request by *get_more_posts()*

property page_info: dict

6.1.7 The InstaPost class

class InstaTweet.instapost.**InstaPost**(*post_data*)

Bases: object

Minimalistic API response wrapper for an Instagram post

__init__(*post_data*)

Initialize an *InstaPost*

Parameters
post_data (*dict*) – the JSON response data of a single Instagram post, found within the *user_data*

id

The post id

is_video

Indicates if the post is a video or photo

filepath

str: Path of downloaded media, set by *download_post()*

tweet_data

dict: Limited data from a successful tweet based off this post, set by *send_tweet()*

property children: list

If the post is a carousel, returns a list of child *InstaPost*'s

property permalink: str

property shortcode: str

property caption: str

property media_url: str

The direct URL to the actual post content

Returns
the video_url if the post is a video, otherwise the *thumbnail_url*

property thumbnail_url: str

property is_downloaded: bool

Checks the *filepath* to see if the post has been downloaded yet

property is_carousel: bool

property filename: str

Concatenates *id* + *filetype* to create the default filename, for use when saving the post

For Example::
>> print(post.filename) "2868062811604347946.mp4"

property filetype: str

Filetype of the post, based on the value of *is_video*

property owner: dict

property timestamp: Union[datetime, str]

add_tweet_data(*tweet*)

Used by *TweetClient* to add minimal tweet data after the post has been tweeted

Parameters
tweet (*Status*) – a Status object from a successfully sent tweet

Return type
bool

6.1.8 The utils module

This module contains a few helper functions that are used throughout the package

InstaTweet.utils.**get_agents**()

Scrapes a list of user agents. Returns a default list if the scrape fails.

Note: Deprecated since 2.0.0b13, but might be useful when new endpoint gets patched

Return type
list

InstaTweet.utils.**get_agent**(*index=0*)

Returns a single user agent string from the specified index of the AGENTS list

Note: Deprecated since 2.0.0b13, but might be useful when new endpoint gets patched

Return type
str

InstaTweet.utils.**get_proxies**(*env_key*)

Retrieve proxies from an environment variable

Return type
Optional[dict]

InstaTweet.utils.**get_root**()

Return type
Path

InstaTweet.utils.**get_filepath**(*filename*, *filetype='txt'*)

Return type
str

Just want to get started?

If you don't care about the details and just want to get this running… I get you.

You'll only need to be familiar with

- The *Profile*, which is used to configure all *settings*

- The *InstaTweet* class, which is used to *start()* the "main script"
- The *db* module, but only if you plan to save data remotely

Otherwise, the other classes are pretty self explanatory

Tip:

- *InstaClient* sends requests to Instagram
- *InstaPost* and *InstaUser* wrap responses from Instagram
- *TweetClient* wraps the `tweepy.API` to *send_tweet()* based off an *InstaPost*
- The *db* module contains the *Profiles* database table and the *DBConnection* class

6.2 InstaTweet Snippets

6.2.1 About the User Map

About the User Map

The *user_map* is a dict containing info about the users added to a *Profile*

- It's used to help detect new posts and compose tweets on a per-user basis
- Entries are created when you *add_users()*, which map the user to a *USER_MAPPING*
- The *USER_MAPPING* maintains lists of hashtags, scraped posts, and sent tweets
- The mapping is updated when you *add_hashtags()* and successfully *send_tweet()*

You can access entries in the *user_map* as follows:

- *get_user()* allows you to retrieve a full entry by username
- *get_hashtags_for()*, *get_scraped_from()*, *get_tweets_for()* provide access to lists

6.2.2 Saving a Profile

Saving a Profile

When you *save()* your *Profile*, the current or specified *name* will be used to create or update a save file in the location specified by *local*

From the Docs…

`Profile.save`(*name=None*, *alert=True*)

Pickles and saves the *Profile* using the specified or currently set name.

Parameters

- **name** (*Optional[str]*) – name to save the *Profile* under; replaces the current *name*

- **alert** (*bool*) – set to `True` to print a message upon successful save

Return type
bool

`InstaTweet.profile.Profile.local =True`

Indicates if saves should be made locally (`True`) or on a remote database (`False`)

Return type
bool

- Local saves are made to the *LOCAL_DIR*, as pickle files
- Remote saves are made to a database (via the *db* module) as pickle bytes

---

Important!!

You MUST configure the *DATABASE_URL* environment variable to save/load remotely

InstaTweet uses `SQLAlchemy` to create a *DBConnection*

- Any `SQLAlchemy`-supported database is therefore also supported by `InstaTweet`
- See the *db* module for more information

6.2.3 Persisting The DBConnection

You can assign a `DBConnection()` to a variable if you want a persistent connection

Here's how:

```
from InstaTweet import DBConnection

# When __enter__ is called for the first time, the engine is set
>>> with DBConnection() as db:
...     pass

# Since the database URL is constant,
#__exit__() doesn't remove the ENGINE class var
>>> print(db.ENGINE)
Engine(postgresql://...)

# The SESSION class var is cleared upon __exit__ though
>>> print(db.SESSION)
None
```

To connect to the database and create a new session, call *connect()*

It will persist until you somehow trigger a call to `__exit__()`

```
# Using the DBConnection object from above
# Call connect() to create a new connection
>>> db.connect()

# Now it can be used like a regular object, and the
```

(continues on next page)

(continued from previous page)

```
# connection will persist until you trigger a call to __exit__()
>>> profile = db.load_profile('myProfile')
>>> profile.view_config()
```

Output:

```
name : myProfile
local : True
session_id :
twitter_keys : {'Consumer Key': 'string', 'Consumer Secret': 'string', 'Access Token':
↪'string', 'Token Secret': 'string'}
user_agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like␣
↪Gecko) Chrome/102.0.5005.63 Safari/537.36
proxy_key : None
user_map : {}
```

yea

6.2.4 Schedule InstaTweet

You can easily schedule InstaTweet using the provided scheduler script

```
#scheduler.py
from InstaTweet import InstaTweet

PROFILES = ['aProfile', 'myProfile']
LOCAL = True


def run(profile_name: str, local: bool = LOCAL):
    """Loads and InstaTweets a profile

    :param profile_name: the name of the :class:`~.Profile`
    :param local: if the profile is saved locally or in a SQLAlchemy supported database
    """
    insta_tweet = InstaTweet.load(profile_name, local=local)
    insta_tweet.start()


if __name__ == '__main__':
    for profile in PROFILES:
        run(profile, local=LOCAL)
```

6.3 Indices and tables

- genindex
- modindex
- search

PYTHON MODULE INDEX

i

INDEX

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V