Collecting Tweets

User Timelines, User Update

Outline

- HCDE user module UserTimeline.py
  - Instantiation
  - Parameters
- HCDE user module Update.py

Using UserTimeline.py – command line

- Part of the HCDE User Module
  - Look in hcde/twitter

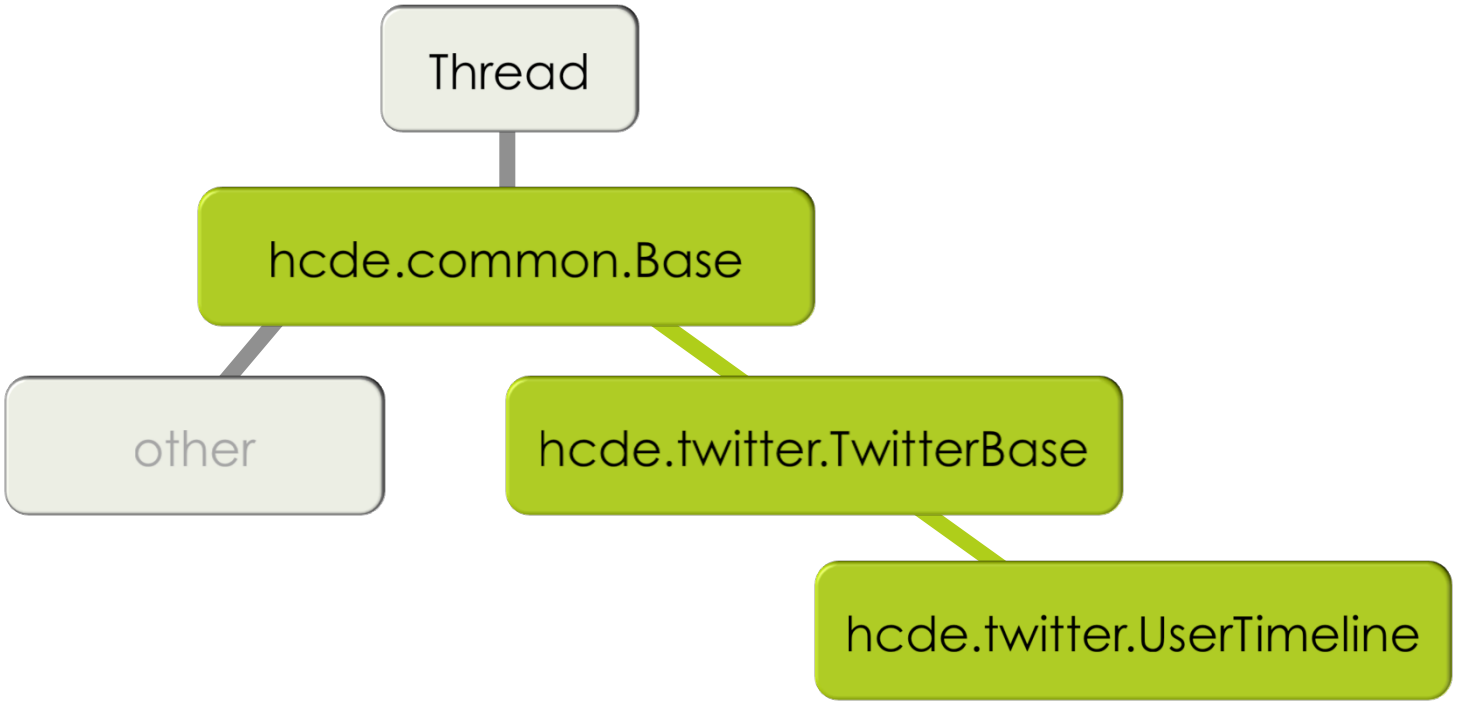
- Like the other code - to get a little help – run without command line parameters

```
python UserTimeline.py

USAGE: python UserTimeline.py -auth <appname> -user <auth_user>
[-n <username> | -id <userid>] [-c <count>] [-s <since_id>] [-m <max_id>]
[-full_tweets] [-log]
```

Reviewing UserTimeline.py code

- UserTimeline.py object hierarchy



User Timeline API Specification

- Twitter Dev Site
  - https://developer.twitter.com/en/docs/tweets/timelines/api-reference/get-statuses-user_timeline
- Take a look …

User Timeline API Specification

- Twitter Dev Site
  - https://developer.twitter.com/en/docs/tweets/timelines/api-reference/get-statuses-user_timeline
- Returns the most recent tweets of the specified user
- Returns up to 3200 tweets of the user
  - But only returns 200 at a time
- Need to rely on since_id or max_id to index through tweets

Reading the UserTimeline() code

- Reading structure
  - Import statements
  - Object definition
    - Object specific data
    - Object methods
  - Command line parameter parsing
  - main() procedure
    - An example of how to use the object

Demo (setup oauth login)

```
import sys, json
from hcde.twitter.Login import Login
from hcde.twitter.UserTimeline import UserTimeline
from hcde.twitter.auth_settings import *
# First create an authorized login object
app = "HCDE530Test01"
user = "hcdetweets" # you need to use a twitter account of your own
app_keys = TWITTER_APP_OAUTH_PAIR(app=app)
app_token_fname = TWITTER_APP_TOKEN_FNAME(app=app)
lg = Login(app_name=app, app_user=user, token_fname=app_token_fname)
lg.set_consumer_key(consumer_key=app_keys['consumer_key'])
lg.set_consumer_secret(consumer_secret=app_keys['consumer_secret'])
lg.login()
```

Demo (setup UserTimeline)

```
# Assuming imports and lg (Login object from prior page)
utl = UserTimeline()
utl.set_auth_obj(obj=lg) # all Twitter requests require auth
utl.set_user_agent(agent="ie")
utl.set_throttling(True)
utl.set_count(c=100)
#utl.set_extended_tweet_mode(True)
#utl.set_username("ThePSF")
utl.set_username("jimmyfallon")
#utl.set_username("timoreilly")
# NOW WHAT?
# If you just want to make a single request just
utl.make_request()
# easy, no threading, just a request
```

Demo (getting timeline result)

```
# Assuming UserTimeline() object and search from prior page
# WHERE IS THE RESULT?
# Check for messages
print utl.messages()
# Get the message from the message queue
resp = utl.get_message()
# The response is a list of dictionary items, json response was converted
print len(resp)
#
for t in resp:
          print json.dumps(t,indent=4,sort_keys=True)
# Because it matters, lets check the first and last items of the response
print json.dumps(resp[0],indent=4,sort_keys=True)
print json.dumps(resp[99],indent=4,sort_keys=True)
```

Paging

- From Jimmy Fallon's timeline (reading the JSON text)
  - First tweet returned (at resp[0])

    ```
    "created_at": "Thu Feb 08 13:38:16 +0000 2018"
    "id": 961595033081217024
    ```

  - Last tweet returned (at resp[99])

    ```
    "created_at": "Tue Jan 09 14:17:23 +0000 2018"
    "id": 950733241031577606
    ```

- For this user 100 tweets was about 1 month of activity
- Tweets returned in reverse chronological order

Demo (getting the next page)

```
# Keep pretty much the same parameters, but set the max_id to our “oldest” ID
utl.set_max_id(950733241031577606)
# Now just make the next request
utl.make_request()
print utl.messages()
resp = utl.get_message()
print len(resp)
print json.dumps(resp[0],indent=4,sort_keys=True)
print json.dumps(resp[99],indent=4,sort_keys=True)
```

Reading Next Page

- From Jimmy Fallon's timeline (reading the JSON text)
  - First tweet returned

```
"created_at": "Tue Jan 09 14:17:23 +0000 2018"
"id": 950733241031577606,
```

  - Last tweet returned

```
"created_at": "Fri Nov 24 13:09:08 +0000 2017"
"id": 934046223874641920
```

- Note the first item was the one we set for max_id
  - Now we have seen that one twice
- This time 100 tweets, just over 1 *month*

Summarize

- Consider Search.py and UserTimeline.py
  - What's the same?
  - What's different?

Update.py

- Update.py allows you to programmatically tweet content on behalf of a given user.
- Update allows you to
  - Tweet
  - Tweet an @ reply
  - Retweet
  - Direct Message

Update.py

- Update is different for a least two reasons:
  - Update.py requires a “POST” http request
    - Search.py and UserTimeline.py use “GET” requests
  - Update.py requires OAuth key/token pairs with WRITE permissions
    - Search.py and UserTimeline.py only need READ

Update.py Code

- Not going to review the Twitter Dev API for Update
  - You've seen this for at least two others – same idea
- Not going to review/walkthrough the code
  - You've seen this at least twice – same idea
- But I will do a little demo
  - Remember much of this can be easily done at the command line with: `python Update.py …`

Demo (setup oauth login)

```
import sys, json
from hcde.twitter.Login import Login
from hcde.twitter.Update import Update
from hcde.twitter.auth_settings import *
# First create an authorized login object
app = "HCDE530Test02" # for Update we need R+W permissions!!!
user = "hcdetweets" # use your own twitter user
app_keys = TWITTER_APP_OAUTH_PAIR(app=app)
app_token_fname = TWITTER_APP_TOKEN_FNAME(app=app)
lg = Login(app_name=app, app_user=user, token_fname=app_token_fname)
lg.set_consumer_key(consumer_key=app_keys['consumer_key'])
lg.set_consumer_secret(consumer_secret=app_keys['consumer_secret'])
lg.login()
```

Demo (do Update)

```
# Assuming imports and lg (Login object from prior page)
upd = Update()
upd.set_auth_obj(obj=lg) # all Twitter requests require auth
upd.set_user_agent(agent="ie")
upd.set_status(status="Status message for today Feb 8 2018. I think the olympics
start today - well tonight – depending on your timezone - with curling, ski
jumping, downhill and luge training runs. #WinterOlympics2018")
# There are really two simple ways to do this, one is the standard
#upd.make_request()
# The other way is a convenience routine that just calls make_request
upd.update_status()
# While this code is based on a Thread object, and can be run like
# a standard thread, there is really no reason to do that
```

Demo (do Update, check)

```
# Still valuable to check and retrieve any messages
print upd.messages()
resp = upd.get_message()
# In this case the returned message is just a single dictionary of
# what was tweeted – or an error message
print json.dumps(resp,indent=4,sort_keys=True)
```