Twitch Bot

Author - Joseph Seering

Language - Python (Tested on Python Version 2.7.9)

This is an IRC (Internet Relay Chat) Bot that can be used to listen on different activities happening in different Twitch Channels. It can connect to multiple channels at once.

How To Run The Bot

Linux and Mac OSX will have Python support by default.
Opening the terminal and typing in *python —version* will tell you the version of Python you're running currently. Windows usually does not come with Python installed by default. You can follow the tutorial available at http://www.howtogeek.com/197947/how-to-install-python-on-windows/ to get Python installed on Windows

Once Python is installed, the bot can be fired up by going into the root folder (where the 4 Python files exist) via the Terminal or Command Prompt and executing *python Runv4.py*

This will run the script and start to print details about the channel. This document will walk you through how the channel's chat and other details can be retrieved.

Structure

Note: # in Python means comments

The code consists of 4 Python scripts:
- Readv3.py
- Runv4.py
- Settingsv2.py
- Socketv2.py

Settingsv2.py

This is script where you tell the bot your credentials. Twitch chat uses IRC which makes use of a technique called OAuth for authentication. You will need an OAuth token for this to work. By default, the script uses the username "experimentalgamedesign" and generates its corresponding OAuth token. Open http://www.twitchapps.com/tmi/ to generate an OAuth token for your Twitch handle. Replace "experimentalgamedesign" with your handle and the *PASS* will be the new OAUTH token.

The host and port need not be changed.

The last variable in this script is the channel list. As mentioned before, this bot can connect to multiple channels. You may specify that in this list.

Socketv2.py

This script is used to open a socket and connect to Twitch. The method openSocket() does exactly that. It fetches the values of HOST, PORT, PASS and IDENT from Settingsv2.py and connects to CHANNELLIST (the list of channels specified)

This script also has a method to send a message to a channel (based on index in CHANNELLIST)

Runv4.py

This is the script that is being executed. It makes use of Settingsv2.py and Socketv2.py. Hence the earlier explanation of the other two. This script heavily makes use of Readv3.py as well.

This script runs in a while loop for 7200 seconds (hardcoded as of now). It receives chunks off the buffer that is received by the socket and writes it to a variable *temp*. Now temp consists of a part of the twitch chat.

temp is iterated on, using a for loop to figure out what is happening in the chat currently.

Sample output

```
@badges=;color=;display-name=Kuroh02;emotes=;id=efa6d5e2-
c78a-40b1-80b4-32772ac70bef;mod=0;room-id=23660075;subscriber=0;turbo=0;user-
id=47685081;user-type= :kuroh02!kuroh02@kuroh02.tmi.twitch.tv PRIVMSG
#esl_overwatch :welcome in any esport lol
```

PRIVMSG - Even though slightly deceptive, this actually means the chat seen publicly by the chatroom. In the above example, “welcome in any esport lol” is the text entered by the viewer. If the *line* contains ‘PRIVMSG’, that means its a public chat message. You can parse that text using different methods written in the **Readv4.py** file like getUser(), getMessage() and so on.

PING - IRC makes use of a PING PONG protocol in order to ensure the client is always connected. When the server send’s a PING to the client, it needs to respond with a PONG. This implementation is necessary to make sure the client stays connected. You can read up on this topic here http://www.irchelp.org/irchelp/new2irc.html (if interested)

CLEARCHAT - This is used to get the handles of the banned users and the name of the channel from *line.*

PRIVMSG and CLEARCHAT are written into a log file - outputlog.csv. This can be referred to later as its hard to read the logs from the terminal.

Readv3.py

This script consists of all the methods that are used to parse a *line* from Runv4.py.

getUser():

This method takes in *line* as an argument and returns the username/handle.
The first check is to see if the input is from a user and not from Twitch itself.
If not, the line is parsed to get the username/handle.

Example:

Lets consider the previous sample output:

```
@badges=;color=;display-name=Kuroh02;emotes=;id=efa6d5e2-
c78a-40b1-80b4-32772ac70bef;mod=0;room-id=23660075;subscriber=0;turbo=0;user-
```

id=47685081;user-type= :kuroh02!kuroh02@kuroh02.tmi.twitch.tv PRIVMSG
#esl_overwatch :welcome in any esport lol
separate = line.split("!", 1)

The code splits the *line* into two based on the separator '!' and assigns it to *separate.*
Now *separate consists of two parts "@badges=;color=;display-name=Kuroh02;emotes=;id=efa6d5e2-c78a-40b1-80b4-32772ac70bef;mod=0;room-id=23660075;subscriber=0;turbo=0;user-id=47685081;user-type= :kuroh02"*

and

"kuroh02@kuroh02.tmi.twitch.tv PRIVMSG #esl_overwatch :welcome in any esport lol"

user = separate[1].split("@", 1)[0]

The second part of *separate* (separate[1]) is split into two based on the separator "@" and the first part of that is assigned to user. Thus we get "kuroh02" as the user.

All other methods of this script make use of similar logic but splits and parses *line* according to the location of the required field in the text.