pybot

Release 1.0.1

Aug 16, 2023

Contents:

CHAPTER 1

bot module

CHAPTER 2

botbrain module

class botbrain.**BotBrain**(*microphone*, *bot=None*)

Bases: object

BRAINDEBUG = False

getMicrophone()

notice(*channel*, *thing*)

respond(*usr*, *channel*, *message*)

say(*channel*, *thing*)

CHAPTER 3

conferror module

exception conferror.**ConfError**(*error*)

Bases: Exception

CHAPTER 4

confman module

class confman.**ConfManager**(*conf=None*)

Bases: object

Singleton class. Opens and parses a JSON-formatted conf file from (generally) the running user's home folder. Looks for .pybotrc. This allows each thread to know only its own network name, and always get back the information specified for that network from the confman.

getChannels(*net*)

getDBName(*net*)

getDBPass(*net*)

getDBType()

getDBUsername(*net*)

getIRCPass(*net*)

getNetwork()

getNetworks()

getNick(*net*)

getNumChannels(*net*)

getNumNets()

getOwner(*net*)

getPort(*net*)

getTimeout(*net*)

CHAPTER 5

db module

class db.**DB**(*bot=None*)

Bases: object

Handles connecting to the database and reading and writing data. Currently supports only MySQL/mariadb, and that probably needs to change.

age = datetime.datetime(2023, 8, 16, 22, 20, 26, 141330)

e(*sql*)

getImgs()

getSeen(*who*)

insert(*where*, *which*, *what*)

insertImg(*user*, *url*, *channel*)

isAdmin(*username*)

replace(*where*, *which*, *what*)

select(*where*, *what*)

updateSeen(*who*, *statement*, *event*)

CHAPTER 6

event module

class event.**Event**(*_type*)

Bases: object

Allows event type definition. The definition accepts a regex. Every event can be triggered by specific lines, messages, message_id or users. Eventually (see time_event branch for proof-of-concept implementation) time-sensitive events will be triggerable as well.

Each line received by the bot is passed to each module in the modules_list. If the module determines the line matches what the event cares about, the event calls each of its subscribers itself, which contains all the information the module needs to respond appropriately.

To use: e = Event("__my_type__") e.define("some_regex") bot.register_event(e, calling_module)

define(*definition=None*, *msg_definition=None*, *user_definition=None*, *message_id=None*, *mode=None*, *case_insensitive=False*)

Define ourself by general line (definition), msg_definition (what someone says in a channel or PM), user_definition (the user who said the thing), or message_id (like 376 for MOTD or 422 for no MOTD) Currently, an event is defined by only one type of definition. If one were to remove the returns after each self. set, an event could be defined and triggered by any of several definitions.

Args: definition: string. regex allowed. msg_definition: string. regex allowed. this is what someone would say in a channel. like "hello, pybot". user_definition: string. the user that said the thing. like 'hlmtre' or 'BoneKin'. message_id: the numerical ID of low-level IRC protocol stuff. 376, for example, tells clients 'hey, this is the MOTD.'

matches(*line*)

Fills out the event object per line, and returns True or False if the line matches one of our definitions. Args: line: string. The entire incoming line.

Return: boolean; True or False.

notifySubscribers(*line*)

Fills out the object with all necessary information, then notifies subscribers with itself (an event with all the line information parsed out) as an argument. Args: line: string

subscribe(*e*)

Append passed-in event to our list of subscribing modules.

Args: e: event.

CHAPTER 7

lite module

class lite.**SqliteDB**(*bot=None*)

Bases: object

e(*sql*)

getImgs()

insertImg(*user*, *url*, *channel*)

isAdmin(*username*)

CHAPTER 8

logger module

class logger.**Logger**

Bases: object

CRITICAL = 0

INFO = 2

WARNING = 1

levels = ['CRITICAL', 'WARNING', 'INFO']

write(*level*, *line*, *nick=None*, *location=None*)

Write out to the logfile of either default location or otherwise specified. Includes calling class in its logged line.

Args: level: enumerated thing from the logger class. line: string. thing to write out to the logger. nick: string. determines filename. location: where to write the logfile out to.

Returns: nothing.

CHAPTER 9

modules package

9.1 Subpackages

9.1.1 modules.snippets package

9.1.1.1 Submodules

9.1.1.2 modules.snippets.commandtest module

modules.snippets.commandtest.**test_function**(*bot*, *message*, *channel*)

9.1.1.3 modules.snippets.d10 module

modules.snippets.d10.**d10**(*bot*, *message*, *channel*)

9.1.1.4 modules.snippets.d12 module

modules.snippets.d12.**d12**(*bot*, *message*, *channel*)

9.1.1.5 modules.snippets.d4 module

modules.snippets.d4.**d4**(*bot*, *message*, *channel*)

9.1.1.6 modules.snippets.d6 module

modules.snippets.d6.**d4**(*bot*, *message*, *channel*)

9.1.1.7 modules.snippets.d8 module

modules.snippets.d8.**d4**(*bot*, *message*, *channel*)

9.1.1.8 modules.snippets.downtime module

9.1.1.9 modules.snippets.platypus module

modules.snippets.platypus.**platform_info**(*bot*, *message*, *channel*)

9.1.1.10 modules.snippets.reflect module

modules.snippets.reflect.**reflect_reload**(*bot*, *message*, *channel*)

9.1.1.11 modules.snippets.second_in_command_test module

9.1.1.12 modules.snippets.snippetutil module

modules.snippets.snippetutil.**reload**(*bot*, *message*, *channel*)

9.1.1.13 Module contents

9.2 Submodules

9.3 modules.basemodule module

class modules.basemodule.**BaseModule**(*events=None*, *printer_handle=None*, *bot=None*, *say=None*)

Bases: object

A base module class for deriving modules (anything you fine folk write, probably) to inherit from. The nice this is this allows you to define your own post_init and handle functions.

In your module's post_init, define and register your own events, and pass your module in.

```
def MyModule(BaseModule):
  def post_init(self):
    e = Event("__wee__")
    e.define("foo")
    self.bot.register_event(e,self)
```

Bam, you've got the things you need (a bot handle, mostly) and by extending BaseModule you implement the right things to be called without error. Elzar.

handle(*event*)

post_init()

Called after init is set up and builds out our basic module's needs. Allows you to do your own post-processing when inheriting from BaseModule.

9.4 modules.bofh module

class modules.bofh.**Bofh**(*events=None*, *printer_handle=None*, *bot=None*, *say=None*)
Bases: *modules.basemodule.BaseModule*

handle(*event*)

post_init()
Called after init is set up and builds out our basic module's needs. Allows you to do your own post-processing when inheriting from BaseModule.

9.5 modules.bonk module

class modules.bonk.**Bonk**(*events=None*, *printer_handle=None*, *bot=None*, *say=None*)
Bases: *modules.basemodule.BaseModule*

get_bonked(*bonkee=''*)

handle(*event*)

post_init()
Called after init is set up and builds out our basic module's needs. Allows you to do your own post-processing when inheriting from BaseModule.

9.6 modules.choose module

class modules.choose.**Choose**(*events=None*, *printer_handle=None*, *bot=None*, *say=None*)
Bases: *modules.basemodule.BaseModule*

handle(*event*)

post_init()
Called after init is set up and builds out our basic module's needs. Allows you to do your own post-processing when inheriting from BaseModule.

9.7 modules.ctof module

class modules.ctof.**Ctof**(*events=None*, *printer_handle=None*, *bot=None*, *say=None*)
Bases: *modules.basemodule.BaseModule*

handle(*event*)

post_init()
Called after init is set up and builds out our basic module's needs. Allows you to do your own post-processing when inheriting from BaseModule.

9.8 modules.d20 module

class modules.d20.**D20**(*events=None*, *printer_handle=None*, *bot=None*, *say=None*)
Bases: *modules.basemodule.BaseModule*

handle(*event*)

post_init()
Called after init is set up and builds out our basic module's needs. Allows you to do your own post-processing when inheriting from BaseModule.

9.9 modules.dad module

class modules.dad.**Dad**(*events=None*, *printer_handle=None*, *bot=None*, *say=None*)
Bases: *modules.basemodule.BaseModule*

handle(*event*)

post_init()
Called after init is set up and builds out our basic module's needs. Allows you to do your own post-processing when inheriting from BaseModule.

9.10 modules.dance module

class modules.dance.**Dance**(*events=None*, *printer_handle=None*, *bot=None*, *say=None*)
Bases: *modules.basemodule.BaseModule*

handle(*event*)

post_init()
Called after init is set up and builds out our basic module's needs. Allows you to do your own post-processing when inheriting from BaseModule.

9.11 modules.debugger module

class modules.debugger.**Debugger**(*events=None*, *printer_handle=None*, *bot=None*, *say=None*)
Bases: *modules.basemodule.BaseModule*

handle(*event*)

mem_store_delete(*mem_store_key*)

post_init()
Called after init is set up and builds out our basic module's needs. Allows you to do your own post-processing when inheriting from BaseModule.

pretty(*d*, *event*, *indent=0*)

recurse(*obj*)

9.12 modules.diabeetus module

class modules.diabeetus.**Diabeetus**(*events=None*, *printer_handle=None*, *bot=None*, *say=None*)
Bases: *modules.basemodule.BaseModule*

get_glucose(*channel*)
get the glucose

handle(*event*)

post_init()
Called after init is set up and builds out our basic module's needs. Allows you to do your own post-processing when inheriting from BaseModule.

9.13 modules.disconnect_yeller module

class modules.disconnect_yeller.**Disconnect_Yeller**(*events=None*, *printer_handle=None*, *bot=None*, *say=None*)
Bases: *modules.basemodule.BaseModule*

handle(*event*)

post_init()
Called after init is set up and builds out our basic module's needs. Allows you to do your own post-processing when inheriting from BaseModule.

9.14 modules.example module

class modules.example.**Example**(*events=None*, *printer_handle=None*, *bot=None*, *say=None*)
Bases: object

handle(*event*)

9.15 modules.examplederived module

class modules.examplederived.**ExampleDerived**(*events=None*, *printer_handle=None*, *bot=None*, *say=None*)
Bases: *modules.basemodule.BaseModule*

handle(*event*)

post_init()
Called after init is set up and builds out our basic module's needs. Allows you to do your own post-processing when inheriting from BaseModule.

9.16 modules.ftoc module

class modules.ftoc.**Ftoc**(*events=None*, *printer_handle=None*, *bot=None*, *say=None*)
Bases: *modules.basemodule.BaseModule*

handle(*event*)

post_init()
Called after init is set up and builds out our basic module's needs. Allows you to do your own post-processing when inheriting from BaseModule.

9.17 modules.hello module

class modules.hello.**Hello**(*events=None*, *printer_handle=None*, *bot=None*, *say=None*)
Bases: *modules.basemodule.BaseModule*

handle(*event*)

post_init()
Called after init is set up and builds out our basic module's needs. Allows you to do your own post-processing when inheriting from BaseModule.

9.18 modules.help module

class modules.help.**Help**(*events=None*, *printer_handle=None*, *bot=None*, *say=None*)
Bases: *modules.basemodule.BaseModule*

get_help_lines()

handle(*event*)

individual_help(*cmd*, *event*)

post_init()
Called after init is set up and builds out our basic module's needs. Allows you to do your own post-processing when inheriting from BaseModule.

9.19 modules.howdy module

class modules.howdy.**Howdy**(*events=None*, *printer_handle=None*, *bot=None*, *say=None*)
Bases: *modules.basemodule.BaseModule*

handle(*event*)

post_init()
Called after init is set up and builds out our basic module's needs. Allows you to do your own post-processing when inheriting from BaseModule.

9.20 modules.isup module

class modules.isup.**Isup**(*events=None*, *printer_handle=None*, *bot=None*, *say=None*)
Bases: *modules.basemodule.BaseModule*

takes a url and determines if the site hosted there is up

handle(*event*)

post_init()
Called after init is set up and builds out our basic module's needs. Allows you to do your own post-processing when inheriting from BaseModule.

9.21 modules.jimmies module

class modules.jimmies.**Jimmies**(*events=None*, *printer_handle=None*, *bot=None*, *say=None*)
Bases: *modules.basemodule.BaseModule*

get_jimmies_status()
Randomly selects and returns a string with a "jimmies" status.

handle(*event*)

post_init()
Called after init is set up and builds out our basic module's needs. Allows you to do your own post-processing when inheriting from BaseModule.

9.22 modules.jury module

class modules.jury.**Jury**(*events=None*, *printer_handle=None*, *bot=None*, *say=None*)
Bases: *modules.basemodule.BaseModule*

handle(*event*)

post_init()
Called after init is set up and builds out our basic module's needs. Allows you to do your own post-processing when inheriting from BaseModule.

9.23 modules.kanbomodule module

class modules.kanbomodule.**KanboModule**(*events=None*, *printer_handle=None*, *bot=None*, *say=None*)
Bases: *modules.basemodule.BaseModule*

handle(*event*)

post_init()
Called after init is set up and builds out our basic module's needs. Allows you to do your own post-processing when inheriting from BaseModule.

9.24 modules.lastfm module

class modules.lastfm.**LastFM**(*events=None*, *printer_handle=None*, *bot=None*, *say=None*)
Bases: *modules.basemodule.BaseModule*

handle(*event*)

post_init()
Called after init is set up and builds out our basic module's needs. Allows you to do your own post-processing when inheriting from BaseModule.

9.25 modules.meme module

class modules.meme.**PhonyMc**
Bases: object

imgflip_password = 'None'

imgflip_userid = 'None'

class modules.meme.**meme**(*events=None*, *printer_handle=None*, *bot=None*, *say=None*)
Bases: object

check_rate(*nick*)
Check to see if the given nick has allowed enough time to pass before calling meme again. Return True and set the new last meme time if true. Warn nick and return False if not.

compare_description(*meme_name*, *user_description*)
compares two strings. if greater than 67% similarity, returns true

contains_url(*line*)
Given a string, returns True if there is a url present

create_ignore_nicks_tuple()
creates a tuple with all nicks from self.ignore_list in <>

create_meme(*meme_id*, *top_line*, *bottom_line*)
Given a meme id from imgflip and two lines, top and bottom, submit a request to imgflip for a new meme and return the URL

format_string(*line*)
Given an appropriate line, strip out <nick>. Otherwise return unmodified line

get_last_meme_time(*nick*)
Given a channel name, return the last time .meme was called in that channel, return 0 if never used

get_line(*array_of_lines*)
Given an array of lines from which to pick, randomly select an appropriate line, clean it up, and return the string.

get_random_flavor()
Change up the flavor text when returning memes. It got boring before

get_random_meme_id()
Selects a random id from the top_memes_list

get_top_memes()
Makes an API call to imgflip to get top 100 most popular memes. Returns a list of results

get_user_lines(*channel*, *nick*)
Given a specific nick and channel, create a list of all their lines in the buffer

handle(*event*)

is_valid_line(*line*)
Given a line from the qdb buffer, return True if certain conditions are met that make it good for meme selection. Return False if not

set_last_meme_time(*nick*)
Upon calling meme, set the last time it was used by that nick

9.26 modules.module module

class modules.module.**Module**(*events=None*, *printer_handle=None*, *bot=None*, *say=None*)
Bases: object

handle(*event*)

load(*modulename*)

unload(*modulename*)

unload_event(*eventname*)

9.27 modules.nicklist module

class modules.nicklist.**Nicklist**(*events=None*, *printer_handle=None*, *bot=None*, *say=None*)
Bases: *modules.basemodule.BaseModule*

handle(*event*)

post_init()
Called after init is set up and builds out our basic module's needs. Allows you to do your own post-processing when inheriting from BaseModule.

9.28 modules.part module

class modules.part.**Part**(*events=None*, *printer_handle=None*, *bot=None*, *say=None*)
Bases: *modules.basemodule.BaseModule*

This command should be used as a private message to the bot or else it will not work

handle(*event*)

post_init()
Called after init is set up and builds out our basic module's needs. Allows you to do your own post-processing when inheriting from BaseModule.

9.29 modules.pimp module

class modules.pimp.**Pimp**(*events=None*, *printer_handle=None*, *bot=None*, *say=None*)
Bases: *modules.basemodule.BaseModule*

handle(*event*)

post_init()
Called after init is set up and builds out our basic module's needs. Allows you to do your own post-processing when inheriting from BaseModule.

9.30 modules.qdb module

class modules.qdb.**QDB**(*events=None*, *printer_handle=None*, *bot=None*, *say=None*)
Bases: object

add_buffer(*event=None*, *debug=False*)
Takes a channel name and line passed to it and stores them in the bot's mem_store dict for future access. The dict will have channel as key. The value to that key will be a list of formatted lines of activity. If the buffer size is not yet exceeded, lines are just added. If the buffer is maxed out, the oldest line is removed and newest one inserted at the beginning.

add_recently_submitted(*q_id*, *submission*)
Takes a string, submission, and adds it to the list of recent submissions. Also we do length checking, only keep record of the previous MAX_HISTORY_SIZE quotes.

delete(*user*, *post_id=''*, *passcode=''*)
A special function that allows certain users to delete posts

format_line(*event*)
Takes an event and formats a string appropriate for quotation from it

get_qdb_submission(*channel=None*, *start_msg=''*, *end_msg=''*, *strict=False*)
Given two strings, start_msg and end_msg, this function will assemble a submission for the QDB. start_msg is a substring to search for and identify a starting line. end_msg similarly is used to search for the last desired line in the submission. This function returns a string ready for submission to the QDB if it finds the desired selection. If not, it returns None.

handle(*event*)

recently_submitted(*submission*)
Checks to see if the given submission is string is at least 75% similar to the strings in the list of recently submitted quotes. Returns the id of the quote if it was recently submitted. If not, returns -1.

strip_formatting(*msg*)
Uses regex to replace any special formatting in IRC (bold, colors) with nothing

submit(*qdb_submission*, *debug=False*)
Given a string, qdb_submission, this function will upload the string to hlmtre's qdb server. Returns a string with status of submission. If it worked, includes a link to new quote.

9.31 modules.r6 module

class modules.r6.**R6**(*events=None*, *printer_handle=None*, *bot=None*, *say=None*)
Bases: *modules.basemodule.BaseModule*

Takes specified stats from r6tab and prints them to irc channel

api_get(*name*)
Needed to set user agent so request would not be blocked, without this a 503 status code is returned

handle(*event*)

post_init()
Called after init is set up and builds out our basic module's needs. Allows you to do your own post-processing when inheriting from BaseModule.

print_stats(*ids*, *js*, *choice*)

9.32 modules.recap module

class modules.recap.**recap**(*events=None*, *printer_handle=None*, *bot=None*, *say=None*)
Bases: *modules.basemodule.BaseModule*

check_rate(*channel*)
Check to see if the given channel has allowed enough time to pass before calling recap again. Return True and set the new time limit if true. Return False if not.

contains_url(*line*)
Given a string, returns True if there is a url present

create_ignore_nicks_tuple()
creates a tuple with all nicks from self.ignore_list in <>

dramatize_line(*line*)
Pass a valid line in, return line with some random type of dramatic formatting

get_episode()
Return a list with two elements: a random show title and episode name

get_lines(*channel*)
Given a channel, searches the qdb buffer for 4 random, suitable lines.

get_timediff(*channel*)
Return how much time remains in the function lockdown

handle(*event*)

post_init()
Called after init is set up and builds out our basic module's needs. Allows you to do your own post-processing when inheriting from BaseModule.

reset_timer(*channel*)
If there's an error getting a recap, call this to reset lockdown timer

scramble_nick(*nick*)
Given a valid nick in the format <nickname>, scramble a vowel in the nick to avoid beeping the user

valid_line(*line*)
Returns True if a given line matches all requirements for validity: Not an action line, longer than minimum length, not spoken by ignored nicks, no URLs

9.33 modules.redditinfo module

9.34 modules.replace module

class modules.replace.**Replace**(*events=None*, *printer_handle=None*, *bot=None*, *say=None*)
Bases: *modules.basemodule.BaseModule*

add_buffer(*event=None*, *debug=False*)
Takes a channel name and line passed to it and stores them in the bot's mem_store dict for future access. The dict will have channel as key. The value to that key will be a list of formatted lines of activity. If the buffer size is not yet exceeded, lines are just added. If the buffer is maxed out, the oldest line is removed and newest one inserted at the beginning.

format_line(*event*)
Takes an event and formats a string appropriate for quotation from it

get_replacement_message(*channel=None*, *find_msg=''*)
Looks through the mem_store to find the most recent message containing find_msg

handle(*event*)

post_init()
Called after init is set up and builds out our basic module's needs. Allows you to do your own post-processing when inheriting from BaseModule.

9.35 modules.replay module

class modules.replay.**Replay**(*events=None*, *printer_handle=None*, *bot=None*, *say=None*)
Bases: object

get_replacement_message(*channel=None*, *find_msg=''*)
Looks through the mem_store to find the most recent message containing find_msg

handle(*event*)

is_number(*e*)

9.36 modules.seen module

class modules.seen.**Seen**(*events=None*, *printer_handle=None*, *bot=None*, *say=None*)
Bases: *modules.basemodule.BaseModule*

handle(*event*)

mem_store_init()

post_init()
Called after init is set up and builds out our basic module's needs. Allows you to do your own post-processing when inheriting from BaseModule.

9.37 modules.shortener module

class modules.shortener.**Shortener**(*events=None*, *printer_handle=None*, *bot=None*, *say=None*)
Bases: *modules.basemodule.BaseModule*

handle(*event*)

post_init()
Called after init is set up and builds out our basic module's needs. Allows you to do your own post-processing when inheriting from BaseModule.

reddit_link(*link*)

9.38 modules.tell module

class modules.tell.**Notice**(*subj*, *obj*, *message*)
Bases: object

class modules.tell.**Tell**(*events=None*, *printer_handle=None*, *bot=None*, *say=None*)
Bases: *modules.basemodule.BaseModule*

handle(*event*)

post_init()
Because of the way this module works we have to make sure to set our event like we normally would with __.tell__, but we cannot define our event with "^.tell" like we normally would as it will only look for that line to trigger the event and the user being told will never receive his message since the bot is only looking for .tell and not the user in the PRIVMSG

We will set the .tell trigger in our handle function "if event.msg.startswith(".tell"):" and set define to PRIVMSG so it searches all lines from users. While simultaneously looking for the .tell trigger from the user.

This is because we actually need 2 things for this module to work.

1.) The user needs to be able to leave a message for someone else using ".tell someuser <Insert message here>"

2.) The user who the .tell message is directed towards will be determined by the PRIVMSG definition. This is determined in the "else" block that searches every line not starting with .tell. If the user matches the stored user from the previous tell trigger, the event will be triggered and pybot will spit out text into the proper channel every time the intended user says something in chat until the buffer is out of .tell events.

9.39 modules.told module

class modules.told.**Told**(*events=None*, *printer_handle=None*, *bot=None*, *say=None*)

Bases: *modules.basemodule.BaseModule*

get_told_status(*target*)

Randomly selects and returns a string with a "told" status.

handle(*event*)

post_init()

Called after init is set up and builds out our basic module's needs. Allows you to do your own post-processing when inheriting from BaseModule.

9.40 modules.twitterposter module

class modules.twitterposter.**TwitterPoster**(*events=None*, *printer_handle=None*, *bot=None*, *say=None*)

Bases: *modules.basemodule.BaseModule*

class PhonyPt

Bases: object

access_token = ''

access_token_secret = ''

api_key = ''

api_secret = ''

handle(*event*)

post_init()

Called after init is set up and builds out our basic module's needs. Allows you to do your own post-processing when inheriting from BaseModule.

pt = <modules.twitterposter.TwitterPoster.PhonyPt object>

user_to_track = 'bhhorg'

9.41 modules.tzone module

class modules.tzone.**Tzone**(*events=None*, *printer_handle=None*, *bot=None*, *say=None*)
Bases: *modules.basemodule.BaseModule*

handle(*event*)

post_init()
Called after init is set up and builds out our basic module's needs. Allows you to do your own post-processing when inheriting from BaseModule.

request_api(*location*)
Takes the location provided and determines whether its a valid request and will return either the time of the location or a message instructing you how to the make the proper call

9.42 modules.uptime module

class modules.uptime.**Uptime**(*events=None*, *printer_handle=None*, *bot=None*, *say=None*)
Bases: *modules.basemodule.BaseModule*

handle(*event*)

post_init()
Called after init is set up and builds out our basic module's needs. Allows you to do your own post-processing when inheriting from BaseModule.

9.43 modules.vyos module

Works only in hlmtre's specifically configured environment and when his house has not burned down

class modules.vyos.**Vyos**(*events=None*, *printer_handle=None*, *bot=None*, *say=None*)
Bases: *modules.basemodule.BaseModule*

handle(*event*)

ping(*nick*)

post_init()
Called after init is set up and builds out our basic module's needs. Allows you to do your own post-processing when inheriting from BaseModule.

9.44 modules.weather module

class modules.weather.**Weather**(*events=None*, *printer_handle=None*, *bot=None*, *say=None*)
Bases: *modules.basemodule.BaseModule*

get_api_request(*x*, *y*)
Simple form the query string and return it.

get_conditions(*query*, *channel*)
given a fully formed query to the OpenWeatherMap API, format an output string

get_lat_long_from_bing(*location*)
go grab the latitude/longitude from bing's really excellent location API. Returns: tuple of x,y coordinates - (0,0) on error

handle(*event*)

post_init()
Called after init is set up and builds out our basic module's needs. Allows you to do your own post-processing when inheriting from BaseModule.

9.45 modules.welcome module

class modules.welcome.**Welcome**(*events=None*, *printer_handle=None*, *bot=None*, *say=None*)
Bases: object

handle(*event*)

9.46 modules.youtube module

class modules.youtube.**Youtube**(*events=None*, *printer_handle=None*, *bot=None*, *say=None*)
Bases: *modules.basemodule.BaseModule*

handle(*event*)

post_init()
Called after init is set up and builds out our basic module's needs. Allows you to do your own post-processing when inheriting from BaseModule.

print_video_title(*event*, *url*, *video_tag*)

9.47 modules.yth module

class modules.yth.**YTH**(*events=None*, *printer_handle=None*, *bot=None*, *say=None*)
Bases: *modules.basemodule.BaseModule*

handle(*event*)

post_init()
Called after init is set up and builds out our basic module's needs. Allows you to do your own post-processing when inheriting from BaseModule.

9.48 Module contents

CHAPTER 10

pybot module

CHAPTER 11

stats module

class stats.**Stats**

Bases: object

db = <db.DB object>

CHAPTER 12

util module

class util.**bcolors**
Bases: object

Allows for prettyprinting to the console for debugging.

CYAN = '\x1b[36m'

ENDC = '\x1b[0m'

FAIL = '\x1b[91m'

GREEN = '\x1b[32m'

HEADER = '\x1b[95m'

OKBLUE = '\x1b[94m'

OKGREEN = '\x1b[92m'

WARNING = '\x1b[93m'

YELLOW = '\x1b[33m'

util.**commands**(**command_list*)

util.**depends**(*self*, *module_name*)

util.**parse_line**(*line*)
returns an object with a nice set of line-pulled-apart members

util.**strip_nick**(*nick*)
Clean up nicks of their op levels (&Schooly_D, ~BoneKin, etc)

CHAPTER 13

version module

CHAPTER 14

webwriter module

class webwriter.**WebWriter**

Bases: object

Handles writing out to teh home folder of the running user. Creates a very simple web page that just lists all the images posted in a channel the bot's in. Probably in trouble for cp because of teh 4cdn links.

CHAPTER 15

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

U

V

W

Index

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y