ErinaBot API reference

Release 0.1

Dec 12, 2020

Contents

CHAPTER 1

ErinaBot

Discord bot with Natural Language Processing and other basic stuff.

Tip: I already coded a nice bot example so check the github repo out!

ErinaBot.**conversation = <ErinaBot._conversation.Conversation object>**
ErinaBot.Conversation instance.

ErinaBot.**db = Database(MongoClient(host=['localhost:27017'], document_class=dict, tz_aware=**
Mongo database instance.

ErinaBot.**intention**(*func*)
Use this decorator to declare intention handlers. The intention handler function name must be the same as intention defined in intentions.yml by example:

intentions.yml

```
-
    -
        - what time is it
        - show time
        - system time
    - show_time
```

Your intention handler for *show_time* should looks like:

```
import time
import ErinaBot as erina
from datetime import datetime

ACCESS_TOKEN = "your_atoken_here"
```

(continues on next page)

(continued from previous page)

```
client = discord.Client()

erina.conversation.load_dictionary("intentions.yml")

@erina.intention
async def show_time(ctx, args):
    """
        **Show system time**

        You can ask me for show system time :smiley:
    """
    date = datetime.utcfromtimestamp(time.time())
    date = date.strftime('%Y-%m-%d %H:%M')

    await ctx.channel.send("Server time: %s" %(date))
```

Tip: As you can notice you can document your intention handler in *python like doc* and it will be showed as help when bot receive a *help* command.

Intention handlers will receive two parameters **ctx** which is a discord.Message instance and **args** that is an *ErinaBot.Arguments* instance which contains arguments (string, numbers, url's) in received messages.

ErinaBot.**music = <ErinaBot._music_player.MusicPlayer object>**

ErinaBot.MusicPlayer instance.

CHAPTER 2

Language

2.1 Conversation

class ErinaBot.**Conversation**

This class handle the received messages and process them to recognize the message intention based on the levenshtein distance between input message and loaded intentions.

Important: You will use this class throw **ErinaBot.conversation** instance.

_Conversation__clear_string(*text*)

Removes strings between quotes also removes punctuations and non ascii characters and also removes bot's name (eri) and youtube url's in order to increase recognition accurate.

Parameters **text** (*str*) – String to clear.

clear_context(*ctx*)

Clears the context value for the especified context.

Parameters **ctx** (*discord.Message*) – Context.

get_context(*ctx*)

Gets the context value for the especified context.

Parameters **ctx** (*discord.Message*) – Context.

Returns The context value.

Return type str

get_context_var(*ctx*, *var*)

Gets the value of the especified context var.

Parameters

- **ctx** (*discord.Message*) – Context.
- **var** (*str*) – Var name.

Returns Var value.

Return type mixed

load_dictionary(*file*)
Loads a dictionary of intentions or dialogs. Must be a .yml file. see *intentions.yml* and *dialogs.yml* for reference.

Parameters **file** (*str*) – Intentions or Dialogs disctionary path.

recognize(*msg*)
Reconize the intention of the given string and call the appropriate intention handler. If the recognition result is a dialog not and intention it will send the dialog answer.

Note: If the intention handler is not defined will not throw an error it will just log an alert.

How to use it:

```
# Bot initialization, intention dictionary load
# and intention definition here

@client.event
async def on_message(message):
    if message.author == client.user:
        return

    if (not client.user in message.mentions)\
        and (not message.mention_everyone)\
        and (not erina.conversation.talking_to_me(message.content)):
        return

    # recognize the incoming message
    await erina.conversation.recognize(message)

client.run(ACCESS_TOKEN)
```

Parameters **msg** (*discord.Message*) – Message to recognize.

set_context(*ctx*, *value*)
Sets the context value for the especified context.

Parameters

- **ctx** (*discord.Message*) – Context.
- **value** (*str*) – Context value.

set_context_var(*ctx*, *var*, *val*)
Creates a context var for the especified context.

Parameters

- **ctx** (*discord.Message*) – Context.
- **var** (*str*) – Var name.
- **val** (*mixed*) – Var value, it could be whatever you want.

talking_to_me(*text*)
Look for the bot's name (eri) in the given string.

Parameters **text** (*str*) – String to search in.

Returns True if the bot's name is in the given string.

Return type boolean

2.2 Arguments

class ErinaBot.**Arguments**(*content*)

This class is used to represent arguments (strings, numbers and urls) inside messages content.

Note: *String* is anything inside quotes.

string(*str*)
An string in the given message content.

number(*int*)
A number in the given message content.

yt_url(*str*)
A YouTube url in the given message content.

You will receive an intance object of this class in your intention handlers. If there is no arguments in the message content class attributes will be *None*.

Quick example of reading arguments:

```
# ...

@erina.intention
async def some_nice_intention(ctx, args):
    if args.string:
        await ctx.channel.send("String argument: %s" %(args.string))

    if args.number:
        await ctx.channel.send("Numeric argument: %i" %(args.number))

    if args.yt_url:
        await ctx.channel.send("YouTube url: %s" %(args.yt_url))
```

__init__(*content*)

Search for arguments in the given string.

Parameters **content** (*str*) – Message content.

CHAPTER 3

Music

3.1 MusicPlayer

class ErinaBot.**MusicPlayer**

Music utilities like download search and play. You will use this class throw ErinaBot.music object.

queues (*dict*)
Dictionary that contains *channel* -> *music queue*.

download_yt_video (*url*)

Downloads a YouTube video.

Parameters **url** (*str*) – Url for the video to download.

Returns each value will be False if the download fails.

Return type tuple(song_path, song_title, song_thumbnail)

get_queue (*text_channel*, *voice_channel*, *loop*)

Gets the music queue for the given voice_channel. If the given voice_channel doesn't have a music queue it creates one.

Parameters

- **text_channel** (*discord.TextChannel*) – text_channel for the queue (if queue needs to be created).
- **voice_channel** (*discord.VoiceClient*) – voice_channel for the queue.
- **loop** (*asyncio.AbstractEventLoop*) – we get this from *client.loop*

Returns Music queue for the given voice_channel.

Return type *ErinaBot.MusicQueue*

get_queue_info (*voice_channel*)

Gets the queued songs for the given voice_channel.

Parameters **voice_channel** (*discord.VoiceClient*) – Gets the queued songs of this channel.

Returns Array of dictionaries (each dictionary contains: ['source'] and ['metadata'] keys).

Return type array[dict]

get_voice_channel(*client*, *author*)

Gets the voice channel for the given user (who sends a message requesting a song).

Parameters

- **client** (*discord.Client*) – Discord client, we gonna search the voice_channel here.
- **author** (*discord.User*) – User who requested the song.

Returns Voice channel (False if user is no connected to a voice channel).

Return type discord.VoiceClient

list_downloaded_songs()

Get a list of the downloaded songs.

Returns Names of the donwloaded songs (only name not full path).

Return type array[str]

play(*client*, *ctx*, *voice_channel*, *metadata*)

Enqueue a song for the given voice_channel.

Song metadata must be a dictionary like this:

```
metadata = {
    'path':'song/file/path.mp3',
    'title':'song title',
    'thumbnail':'song thumbnail url',
    'url':'yt url',
    'requested_by': ctx.author.mention
}
```

Note: You get those parameters (path, title, thumbnail) from ErinaBot.download_yt_video()

Parameters

- **client** (*discord.Client*) – Needed to create the queue if it doesn't exists.
- **ctx** (*discord.Message*) – Needed to create the queue if it doesn't exists.
- **voice_channel** (*discord.VoiceClient*) – The song will be played here.
- **metadata** (*dict*) – Song metadata.

remove_queue(*voice_channel*)

Removes queue for the given voice_channel.

Parameters **voice_channel** (*discord.VoiceClient*) – It will removes the queue for this channel.

search_yt_video(*query*)

Search for songs (any video actually) for the given search query.

Note: Results are limited to 10.

Parameters **query** (*str*) – Search for videos of this in YouTube.

Returns Array dictionaries of the results (each dictionary contains ['url'] and ['name'] keys)

Return type array[dict]

set_volume(*voice_channel*, *volume*)
Sets the music volume for the given channel (if it is playing music).

Parameters

- **voice_channel** (*discord.VoiceClient*) – Set the volumen for this channel.
- **volume** (*float*) – Volume from 0.0 to 1.0.

3.2 MusicQueue

class ErinaBot.**MusicQueue**(*text_channel*, *voice_channel*, *loop*)
This class is used for MusicPlayer to manage songs queues for the music play feature. This class uses asyncio Queues.

queue (*asyncio.Queue(maxsize=50)*)
Songs queue

text_channel (*discord.TextChannel*)
Queue text channel (user requested songs from this text channel)

voice_channel (*discord.VoiceClient*)
Queue voice channel (user who requested songs is in this voice channel)

volume (*float*)
Volume for the queue songs (float from 0.0 to 1.0, 1.0 by default)

_MusicQueue__destroy()
Disconnects voice channel and stops task loop.

_MusicQueue__queue_worker()
Queue worker waits for songs in the queue and plays them in voice channel. When a song is putted to the queue it plays it and also sends a embed message with song info.

It waits 3 minutes for songs. If there is no songs putted in those 3 minutes automatically leaves the voice_channel and stop the queue task.

__init__(*text_channel*, *voice_channel*, *loop*)
Initializes the Music Queue.

Parameters

- **text_channel** (*discord.TextChannel*) – Text channel for the queue (when a song starts it sends song's info to this channel).
- **voice_channel** (*discord.VoiceClient*) – Songs will be played here.
- **loop** (*asyncio.AbstractEventLoop*) – We need this to create tasks (we get it from client.loop).

CHAPTER 4

Miscellaneous

4.1 ErinaBot Utilities

Utilities for the example implementation.

Sorry but I'm spanish speaker so example bot implementations are in taco.

ErinaBot.utils.**covid_cases**(*search*)
Search for latest covid statistics for the given country.

Parameters **search** (*str*) – Country name.

Returns Covid cases or error message if it couldn't find the country statistics.

Return type str

ErinaBot.utils.**get_joke**()
Gets a random joke.

Returns Joke.

Return type str

ErinaBot.utils.**get_meme**()
Gets a random meme.

Returns Meme url.

Return type str

ErinaBot.utils.**get_nudes**()
(° °)

Returns file and thumbnail urls.

Return type tuple(file_url, preview_url)

Python Module Index

e

Index

Symbols

A

C

D

E

G

I

L

M

N

P

Q

R

S

T

V

Y