XChatBot

Apr 26, 2020

Contents:

`XChatBot` is a xmpp bot library written in python using the nbxmpp library from Gajim

CHAPTER 1

Requirements

- python 3
- pygobject
- nbxmpp

optionally

- pipenv

CHAPTER 2

Install

2.1 With `pip`

```
pip install xchatbot
```

2.2 Using `git`

```
git clone https://git.sr.ht/~fabrixxm/xchatbot
```

then install required packages:

with pipenv:

```
pipenv --site-packages --python 3
pipenv install
```

on osx you need first to install python3 with brew:

```
brew install python3 pipenv pygobject3
```

on Arch:

```
pacman -S python-gobject python-nbxmpp
```

on Debian:

```
apt install python3-gi python3-nbxmpp
```

CHAPTER 3

Configuration

The script loads a configuration file called after the bot class name. For a bot class `EchoBot` the script will look for `./echobot.rc`, `~/.echobot.rc` and `/etc/echobot.rc` in this order, and will load the first it find.

An example config file is provided as *echobot.rc.dist*, with comments.

See *xchatbot.rc*

CHAPTER 4

Customize the bot

Subclass *xchatbot.XChatBot* class and implement your commands as method of your class.

A bot class can have public commands and private commands. Is it also possibile to define a default to handle unknown commands.

The bot is started calling the classmethod *start()*

CHAPTER 5

Commands

The bot will react to specific commands with optionals arguments.

Commands are method of the class named `cmd_commandname`. Each command method must get a `peer` (*Peer*) parameter and optionally a number of args.

A docstring should be provided that is used by `help` command to build the help message.

If the numbers of args given in the message doesn't match the function signature, an error message is returned.

By convention, command arguments are listed before description in method docstring

Here an example:

```
from xchatbot import XChatBot

# My custom bot
class MyEchoBot(XChatBot):

        # 'hello' command. Takes no arguments
        def cmd_hello(self, peer):
                """Say hello to the bot"""
                peer.send("Hello to you!")

        # 'sum' command. Takes exactly two arguments
        def cmd_sum(self, peer, a, b):
                """<a number> <another number> - Sum two numbers"""
                peer.send(str(int(a) + int(b)))

        # 'echo' command. Takes a variable number of arguments
        def cmd_echo(self, peer, *args):
                """Echo back what you typed"""
                msg = "You said: " + " ".join(args)
                peer.send(msg)

if __name__ == "__main__":
        MyEchoBot.start()
```

To test this, create a `myechobot.rc` config file and run the bot:

```
$ python myechobot.py
```

CHAPTER 6

Private commands

A command can be marked as private using the `@private` decorator.

A private command is listed in help and is executed only if the message comes from the admin JID set in config file

```
from xchatbot import XChatBot, private

class MyEchoBot(XChatBot):
        ...

        # a private command. Takes a single argument
        @private
        dev cmd_lights(self, peer, status):
            """<status:on or off> - Turn lights on or off"""
            if status == "on":
                # turn on the lights
                peer.send("Lights are now on")
            elif status == "off":
                # turn off the lights
                peer.send("Lights are now off")
            else:
                peer.send("please, 'on' or 'off'.")
```

CHAPTER 7

Default command

If no commands match the message received, *default()* method is called. Your bot class can override this method to return a default response:

```
class MyEchoBot(XChatBot):
        ...
        def default(self, peer, *args):
            peer.send("I'm sorry, I don't understand you. Write me 'help'")
```

CHAPTER 8

Logging

A pre-configured *logging.Logger* object is available as class attribute *logger*

```
class MyEchoBot(XChatBot):
        ...
        def cmd_log(self,
```

CHAPTER 9

Modules

9.1 xchatbot module

xchatbot - the Xtensible xmpp Chat Bot

Build an XMPP bot extending the base class *XChatBot*

Example

A simple bot with one public command "echo" and one private to admin command "hello" "help" command is auto-generated.

```
from xchatbot import XChatBot

class MyBot(XChatBot):
    def cmd_echo(self, peer, *args):
        "Echo back what you typed"
        msg = "You said: " + " ".join(args)
        peer.send(msg)

    @private
    def cmd_hello(self, peer, name):
        "<name> - Private command for admin user"
        peer.send("Welcome " + name)

if __name__ == "__main__":
    MyBot.start()
```

class xchatbot.**Peer**(*bot*, *jid*, *nick*, *is_groupchat=False*)

Bases: object

The peer that sent the message

jid

the peer jid

Type str

nick
the nickname

Type str

is_admin
True if the peer is a bot admin

Type bool

is_groupchat
True if the peer wrote to the bot is in a MUC

Type bool

send(*message*)
Send a message to the peer

Note: If peer is in a groupchat, the nickname is prepended to the message:

{nick}: {message}

Parameters **message** (str) – The message

class xchatbot.**XChatBot**(*jidparams*)
Bases: object

The Xtensible xmpp Chat Bot

options
options loaded from config file

Type dict

jid
bot JID

Type nbxmpp.protocol.JID

muc_nick
bot nick in MUC rooms

Type str

admin_jid
admin user JID

Type str

accept_presence
True if bot accept to be included in rooster by any users

Type bool

accept_muc_invite
True if bot accept invites to MUC rooms by any users

Type bool

logger
configured logger to be used by the bot subclass

Type logging.Logger

connect()
Connect to the server

default(*peer*, **args*)
Default command

This is called when the peer sends a command not defined in the bot.

Parameters

- **peer** (*Peer*) – The remote peer
- ***args** – arguments of the command

disconnect()
Disconnect from the server

do_help(*peer*)
Send help message to the peer

Parameters **peer** (*Peer*) – The remote peer

enter_muc(*muc_jid*)
Join a multiuser conference

Parameters **muc_jid** (str) – MUC JID

get_password(*cb*, *_mech*)
Get password

By default, calls cb() with password from config file.

Parameters **cb** (func(str)) – callback to call with the password

quit()
Disconnecty the bot and quit

Note: Deprecated. Use *disconnect()*

---

register_on_disconnect(*handler*)
Register handler that will be called on disconnect

send_groupchat_to(*to_jid*, *text*)
Send a message to a MUC by JID

Parameters

- **to_jid** (str) – MUC JID
- **text** (str) – Message text

send_message(*message*)
Send a message

Parameters **message** (nbxmpp.protocol.Message) – the message to send

send_message_to(*to_jid*, *text*)
Send a chat message to a JID

Parameters

- **to_jid** (str) – Recipient JID

- **text** (str) – Message text

send_received(*to_jid*, *message_id*)
Send a message delivery receipt

Will mark the message as received by the bot in user's client. It's sent automatically when incoming messages are parsed.

Parameters

- **to_jid** (str) – Recipient JID
- **message_id** (str) – Message id

classmethod start()
Start bot

Loads config file, connects the bot to the server, setups error and quit handlers, start GLib.MainLoop loop.

xchatbot.**store**(*name*, *data*)
store data as pickle value to file

Parameters

- **name** (str) – Filename, without extension
- **data** (str) – Python data to store

xchatbot.**load**(*name*, *default*)
Load pickled data from file.

Parameters

- **name** (str) – Filename to load, without extension
- **default** (any) – Python data returned if file is not found

Returns

Loaded data

If file is not found, the value of default parameter is returned

Return type str

xchatbot.**private**(*func*)
Decorator to mark command private for admin user

9.2 xchatbot.rc

CHAPTER 10

Indices and tables

- genindex
- modindex
- search

Python Module Index

x

Index