PyStark

Release 2021-2022

StarkProgrammer

Jan 19, 2022

INTRODUCTION

PyStark is a fast, easy-to-use tool for creating Telegram bots in Python that is completely powered by Pyrogram, one of the best MTProto Frameworks available.

This documentation is designed primarily for absolute beginners, keeping in mind the specific needs of all major operating systems.

CHAPTER

ONE

TRY OUT PYSTARK

Not convinced to use PyStark? Try it out first, using few small steps given in the quick-start section.

- *Quick Start* - Overview to get you started quickly.

CHAPTER

TWO

HOW THE DOCUMENTATION IS ORGANIZED

- *Installation* - Overview to get you started quickly.
- *Boilerplate* - Generate a boilerplate.
- *Mandatory Variables* - Setup the needed variables for bot.
- *Customization* - Easily customize your bot.
- *Creating Plugins* - Code your own plugins
- *Run Bot* - Run your bot locally.
- *Using Databases* - Use various databases with PyStark.

CHAPTER

THREE

EASY MANTRA TO USE THIS DOCUMENTATION

Just tap on `Next` button below every page, keep reading while following the steps whenever necessary. That's it.

CHAPTER

FOUR

META

- *PyStark FAQs* - Answer to common PyStark questions.
- *ChangeLog* - ChangeLog for PyStark releases.

4.1 Quick Start

Following these steps will allow you to see **PyStark** in action as quickly as possible.

Note: Installation of Python with version 3.6 or above is required.

4.1.1 Steps

1. *Open up your terminal*.
2. Install PyStark with pip:

```
$ pip3 install pystark
```

3. Generate a boilerplate using PyStark's command-line tool.

```
$ pystark --boilerplate
```

4. Enter the newly created `boilerplate` directory.

```
$ cd boilerplate
```

5. *Open the file manager in current directory.*
6. Edit the `.env` file and fill your *API_ID*, *API_HASH* and *BOT_TOKEN*. Get the API keys from my.telegram.org and bot token from BotFather
7. Change the default values of messages in `data.py`.
8. Run the bot using python:

```
$ python3 bot.py
```

4.1.2 What does this do?

The above steps will help you set up your bot and run it. You can use the command **/start** to check if your bot is actually running.

Your bot now has four default commands:

- `/start` - Start the bot.
- `/help` - See a help message for the bot.
- `/about` - About the bot.
- `/id` - Get Telegram ID (also works in groups)

Stop the bot using `Ctrl+C`.

4.2 Installation

This guide will show you how to install PyStark. Be sure to keep an eye out for new releases and keep upgrading the library.

Contents

4.2.1 Installing PyStark

PyStark is available on PyPI and it's latest stable version can be installed using pip:

```
$ pip3 install pystark
```

4.2.2 Using Beta Version

Note: Features are almost always released as soon as they have been committed and checked. So there will be little difference even if the package is installed with pip.

When you install from the git master branch, you will be able to install the beta versions of the new features. You can do that using this command:

```
$ pip install git+https://github.com/StarkBotsIndustries/pystark.git
```

4.2.3 Upgrading pre-installed PyStark

Being a new library, we keep updating PyStark. You can check for new releases on PyPI. Thus, you will be able to use new features of PyStark. Here's how to upgrade, if you have pre-installed PyStark:

```
$ pip install --upgrade pystark
```

4.3 Generating Boilerplate

PyStark comes with a command line tool to make everything even more simpler. You can easily generate a boilerplate to get started. You can also create a boilerplate with added Heroku support. Isn't that amazing?

Contents

4.3.1 What is a boilerplate ?

Boilerplate Code or Boilerplate refers to sections of code that have to be included in many places with little or no alteration.

While using PyStark some code will be same for all bots. Our tool will help you to generate that much code, so you don't have to code and it makes it easier to use PyStark. When you will generate a boilerplate using pystark, a folder with some files will be created for you.

You can choose to generate a boilerplate with or without Heroku Support. For first-timers, I recommend try using without Heroku Support which can be run locally.

4.3.2 Generating a boilerplate to run locally

For generating a boilerplate for local deployment, run this command:

```
$ pystark --boilerplate
```

A folder named `boilerplate` will be created for you in that folder.

4.3.3 Generating a boilerplate with Heroku Support

For added Heroku support, run this command:

```
$ pystark --boilerplate-heroku
```

A folder named `boilerplate` will be created for you in that folder.

4.4 Mandatory Variables

Note: Never disclose these keys to anyone!

Contents

4.4.1 API Keys

API Keys are one of the most important needed keys to work with any MTProto Framework. They include a `API_ID` and `API_HASH`.

You can get these from my.telegram.org

4.4.2 Bot Token

Bot Token is a specific token for every telegram bot. You will get it when you create a new bot using BotFather

It should be filled as `BOT_TOKEN`

4.4.3 Filling the Variables

- **For Local Deploy** - fill them in `.env` file.
- **For Heroku Deploy** - fill them after you tap on `Deploy to Heroku` button on your repository.

4.4.4 Non-mandatory Variables

- `CMD_PREFIXES` - prefixes for commands (defaults to "/"). For multiple prefixes, specify multiple together like "/.*"
- `OWNER_ID` - Your Telegram ID
- `TIMEZONE` - "Asia/Kolkata"
- `DATABASE_URL` - for PostgreSQL database
- `REDIS_URL` - for Redis database (public endpoint)
- `REDIS_PASSWORD` - for Redis database

4.5 Creating Plugins

Some Python knowledge is required to create plugins in general. Therfore, I highly recommend you to learn Python first.

Note:

- All plugins must be added to the **plugins** folder.
- Plugins must end with .py extension

Here's a sample code for a new plugin

```
# Import class 'Stark' in every plugin
from pystark import Stark, Message

# use 'Stark.cmd' decorator to create commands
# @Stark.cmd('name', owner_only=False, extra_filters=None, group=0) - defaults

@Stark.cmd('sample')  # or @Stark.command('sample')
async def sample_function(bot: Stark, msg: Message):
    # 'msg.react()' is 'msg.reply()' with del_in added argument
    await msg.react('This will be the reply when /sample is sent.')
```

But anyway, you can create easier plugins like text plugins with no python knowledge whatsoever.

```
from pystark import Stark


@Stark.cmd('command_name')
async def text_plugin(bot, msg):
    text = 'your text here'
    await msg.react(text)
```

For example, below plugin has a command `/greet` and the bot will reply with *Welcome to the Bot*

```
from pystark import Stark
```

(continues on next page)

(continued from previous page)

```
@Stark.cmd('greet')
async def text_plugin(bot, msg):
    text = 'Welcome to the Bot'
    await msg.react(text)
```

4.6 Customization

There are a lot of customization options in PyStark to customize the behavior of your bot.

Contents

4.6.1 Change the default messages

PyStark comes with in-built plugins like `start` and `help`. But what if you want to have different messages than the in-built ones? They are easily customizable.

After you have finished generating a boilerplate, you will see a file named `data.py`. You can change it's values to change the default messages.

Special Keywords - You may want to mention user or bot in `start` or `help` messages. You can use special keywords to do that. They will be replaced at runtime and will be different for all users.

- `{user}` - User's first name
- `{bot}` - Bot's name
- `{user_mentions}` - User mention as a hyperlink
- `{bot_mentions}` - Bot mention as a hyperlink
- `{owner}` - Owner mention (only works if `OWNER_ID` is set else `@StarkBots`)

So let's say your `start` message is set to *Hi {user}* and your first name on telegram is *Stark* then bot will send *Hi Stark*.

4.6.2 Remove the default plugins

PyStark comes with four in-built plugins. To remove this you need to edit `bot.py`. Use `default_plugins=False` while calling the `activate` function.

You will see this:

```
Stark().activate()
```

Change that to this:

```
Stark().activate(default_plugins=False)
```

4.6.3 Rename the plugins directory

You may notice that if you rename the plugins directory, the plugins won't load. To fix this you need to pass the name of your plugins directory to the `activate` function. Open `bot.py`.

You will see this:

```
Stark().activate()
```

Change that to this:

```
Stark().activate(plugins="name of plugins folder")
```

Let's say I renamed the `plugins` folder to `files`. Then I should do this:

```
Stark().activate(plugins="files")
```

4.7 Run Bot Locally

You can run your bot using simple python.

- First go to your folder using `cd`
- Then run `bot.py`

```
$ python3 bot.py
```

- You can also run your bot in any IDE like PyCharm or VS Code. Just run the `bot.py` file.

4.8 Using Databases

You can use any database you wish with PyStark, but we have provided a simple default setup for some databases, such as PostgreSQL and Redis, to make them even easier to use. By following this guide, you will have a basic understanding of how to use them.

Note: This feature is still in beta. There are a lot of things to add like alembic support for sqlalchemy, etc and this is just a pre-release.

Contents

4.8.1 PostgreSQL (using sqlalchemy)

- **Database URL** - You need to add `DATABASE_URL` to `.env`. If you are using Heroku boilerplate, leave it to **Heroku** and **pystark**. Otherwise, you can get a Database URL from ElephantSQL
- **Creating Tables** - You need to create all the tables with all columns you need. In Python, using Classes.

Below is a code example for a table named `users` with 3 columns named `user_id`, `name`, and `aim`:

```
# Import 'Base' and 'Session' already made by pystark
from pystark.database.postgres import Base, Session
# Import basic sqlalchemy classes
from sqlalchemy import Column, Integer, String


# Every class should inherit from 'Base'
class Users(Base):
    __tablename__ = "users"
    __table_args__ = {'extend_existing': True}
    user_id = Column(Integer, primary_key=True)  # sql primary key (pk)
    name = Column(String)
    aim = Column(String)

    def __init__(self, user_id, name, aim=None):
        self.user_id = user_id
        self.name = name
        self.aim = aim


# Create Table
Users.__table__.create(checkfirst=True)
```

- **Querying Tables** - You can query tables using `Session` object or the in-built pystark functions.
  - *Using Session object*
  - *Using in-built functions*

Querying Postgres Tables

In this guide, you will learn how to query using postgres tables while using PyStark

Note: This feature is still in beta. There are a lot of things to add like default classes, alembic support, etc and this is just a pre-release.

Contents

Using the in-built functions

PyStark provides some default functions to query postgres tables. These functions allow you to query tables using table name (`__tablename__` attribute), that is, a string instead of a class. Therefore, you do not need to import classes.

Note: All the in-built functions are asynchronous.

Table 1: In-built Functions

Name	Function
all_db	*Get All Rows*
get_db	*Get a Particular Row*
count_db	*Get Number of Rows*
set_db	*Set/Update value of a key in a Row*
delete_db	*Delete a Row*

- **Get All Rows**

```
from pystark.database.postgres import all_db


# Get all rows from "users" table as dicts.
async def get_users():
    all_data = await all_db("users")
    print(all_data)
```

- **Get a Particular Row**

```
from pystark.database.postgres import get_db


# Get row using primary key from "users" table.
async def get_user():
    user_id = 500123456  # primary key
    get_data = await get_db("users", user_id)
    print(get_data)
```

- **Get Number of Rows**

```
from pystark.database.postgres import count_db


# Get number of rows in "users" table.
async def user_count():
    count = await count_db("users")
    print(count)
```

- **Set/Update value of a key in a Row**

```
from pystark.database.postgres import set_db
```

(continues on next page)

(continued from previous page)

```
# set/update key, value pairs in "users" table.
async def set_data():
    user_id = 500123456  # primary key
    key_to_change = "aim"
    new_value = "programmer"
    set_data = await set_db("users", user_id, key_to_change, new_value)
    print("Set")
```

- **Delete a Row**

```
from pystark.database.postgres import delete_db


# Delete a row using primary key from "users" table.
async def delete_user():
    user_id = 500123456
    delete_data = await delete_db("users", user_id)
    print("Deleted")
```

Using the Regular Way (Session object)

You can query tables using the `Session` object which is the regular way in sqlalchemy.

```
# import 'Session' object
from pystark.database.postgres import Session
# import Python class for respective table
# let's say it is in 'users_sql.py' inside 'database' folder.
from database.users_sql import Users


# This function gives total 'rows', that is total user ids in 'users' table.
def num_users():
    users = Session.query(Users).count()
    # close session after all queries are made.
    Session.close()
    return users


# This function returns 'name' and 'aim' for users by using 'user_id'
def get_name_and_aim(user_id):
    query = Session.query(Users).get(user_id)
    name = query.name  # get name
    aim = query.aim  # get aim
    Session.close()
    return (name, aim)


# This function sets name and aim for users by using 'user_id'
def set_name_and_aim(user_id, name, aim):
```

(continues on next page)

(continued from previous page)

```
    query = Session.query(Users).get(user_id)
    query.name = name  # set name
    query.aim = aim  # set aim
    Session.commit()  # use this after setting anything.
    # Now you don't need to 'Session.close()' as you used 'Session.commit()' already.

# Etc
```

4.8.2 Redis (using redis-py)

- **Variables** - You need to set `REDIS_URL (public endpoint) and REDIS_PASSWORD by creating a database at redislabs.com
- **Setting and Getting key-value pairs**

```
from pystark.database.redis_db import redis

redis.set('Agra', 'Taj Mahal')
```

```
redis.get('Agra')
```

```
b'Taj Mahal'
```

4.8.3 MongoDB

Coming soon.

4.9 Frequently Asked Questions

Contents

4.9.1 What is PyStark?

- PyStark is a spoon-feeding not-even-library based on Pyrogram.
- Pyrogram is a spoon-feeding MTProto Framework made in Python.
- Python is a spoon-feeding programming language.

4.9.2 Where to run commands?

This means that you are an absolute beginner.

- If you are using `Windows`, tap on Start button and search for **cmd** or **Command Prompt**.
- If you are using `MacOS` or `Linux`, search for **Terminal**.

4.9.3 How to open file manager in current directory?

- For `Windows` use the start command:

```
$ start .
```

- For `MacOS` use the open command:

```
$ open .
```

- For `Linux` use the xdg-open command:

```
$ xdg-open .
```

The dot (.) after command is required to open in current directory.

4.10 ChangeLog

Latest Version - `v0.2.5`

`v0.2.5`

- Added in-built functions to query postgres tables - *Read More*
- Added ChangeLog to docs (this webpage)
- Improve documentation using sphinx-toolbox

`v0.2.4`

- This Documentation was created