












 eternnoir /
pyTelegramBotAPI



 Code  Issues  Pull requests  Discussions  Actions  Projects  Wiki  Security 



Python Telegram bot api.

 GPL-2.0 license

 7.2k stars  1.9k forks  226 watching  Activity

 Public repository

 master ▾



 Branches  Tags



coder2020official ...

✓ 2 weeks ago



[View code](#)

pyTelegramBotAPI



A simple, but extensible Python implementation for the [Telegram Bot API](#).

Both synchronous and asynchronous.

Supported Bot API version: 6.8!

Official documentation

Official ru documentation

Contents 

- [Getting started](#)
- [Writing your first bot](#)
 - [Prerequisites](#)
 - [A simple echo bot](#)
- [General API Documentation](#)

- Types
- Methods
- General use of the API
 - Message handlers
 - Edited Message handler
 - Channel Post handler
 - Edited Channel Post handler
 - Callback Query handlers
 - Shipping Query Handler
 - Pre Checkout Query Handler
 - Poll Handler
 - Poll Answer Handler
 - My Chat Member Handler
 - Chat Member Handler
 - ChatJoin request handler
- Inline Mode
 - Inline handler
 - Chosen Inline handler
 - Answer Inline Query
- Additional API features
 - Middleware handlers
 - Custom filters
 - TeleBot
 - Reply markup
- Advanced use of the API
 - Using local Bot API Server
 - Asynchronous TeleBot
 - Sending large text messages
 - Controlling the amount of Threads used by TeleBot
 - The listener mechanism
 - Using web hooks
 - Logging
 - Proxy
 - Testing
- API conformance limitations
- AsyncTeleBot
- F.A.Q.
 - How can I distinguish a User and a GroupChat in message.chat?
 - How can I handle recurring ConnectionResetErrors?
- The Telegram Chat Group
- Telegram Channel
- More examples
- Code Template
- Bots using this library

Getting started

This API is tested with Python 3.8-3.11 and Pypy 3. There are two ways to install the library:

- Installation using pip (a Python package manager):

```
$ pip install pyTelegramBotAPI
```



- Installation from source (requires git):

```
$ git clone https://github.com/eternnoir/pyTelegramBotAPI.git
$ cd pyTelegramBotAPI
$ python setup.py install
```



or:

```
$ pip install git+https://github.com/eternnoir/pyTelegramBotAPI.git
```



It is generally recommended to use the first option.

While the API is production-ready, it is still under development and it has regular updates, do not forget to update it regularly by calling

```
pip install pytelegrambotapi --upgrade
```



Writing your first bot

Prerequisites

It is presumed that you [have obtained an API token with @BotFather](#). We will call this token `TOKEN`. Furthermore, you have basic knowledge of the Python programming language and more importantly [the Telegram Bot API](#).

A simple echo bot

The `TeleBot` class (defined in `_init_.py`) encapsulates all API calls in a single class. It provides functions such as `send_xyz` (`send_message`, `send_document` etc.) and several ways to listen for incoming messages.

Create a file called `echo_bot.py`. Then, open the file and create an instance of the `TeleBot` class.

```
import telebot
```



```
bot = telebot.TeleBot("TOKEN", parse_mode=None) # You can set parse_mode by default. HTML or MARKDOWN
```

Note: Make sure to actually replace `TOKEN` with your own API token.

After that declaration, we need to register some so-called message handlers. Message handlers define filters which a message must pass. If a message passes the filter, the decorated function is called and the incoming message is passed as an argument.

Let's define a message handler which handles incoming `/start` and `/help` commands.

```
@bot.message_handler(commands=['start', 'help'])
def send_welcome(message):
    bot.reply_to(message, "Howdy, how are you doing?")
```



A function which is decorated by a message handler **can have an arbitrary name, however, it must have only one parameter (the message)**.

Let's add another handler:

```
@bot.message_handler(func=lambda m: True)
def echo_all(message):
    bot.reply_to(message, message.text)
```



This one echoes all incoming text messages back to the sender. It uses a lambda function to test a message. If the lambda returns True, the message is handled by the decorated function. Since we want all messages to be handled by this function, we simply always return True.

Note: all handlers are tested in the order in which they were declared

We now have a basic bot which replies a static message to `/start` and `/help` commands and which echoes the rest of the sent messages. To start the bot, add the following to our source file:

```
bot.infinity_polling()
```



Alright, that's it! Our source file now looks like this:

```
import telebot

bot = telebot.TeleBot("YOUR_BOT_TOKEN")

@bot.message_handler(commands=['start', 'help'])
def send_welcome(message):
    bot.reply_to(message, "Howdy, how are you doing?")

@bot.message_handler(func=lambda message: True)
def echo_all(message):
    bot.reply_to(message, message.text)

bot.infinity_polling()
```



To start the bot, simply open up a terminal and enter `python echo_bot.py` to run the bot! Test it by sending commands (`/start` and `/help`) and arbitrary text messages.

General API Documentation [↗](#)

Types [↗](#)

All types are defined in `types.py`. They are all completely in line with the [Telegram API's definition of the types](#), except for the `Message's from` field, which is renamed to `from_user` (because `from` is a Python reserved token). Thus, attributes such as `message_id` can be accessed directly with `message.message_id`. Note that `message.chat` can be either an instance of `User` or `GroupChat` (see [How can I distinguish a User and a GroupChat in message.chat?](#)).

The Message object also has a `content_type` attribute, which defines the type of the Message. `content_type` can be one of the following strings: `text`, `audio`, `document`, `photo`, `sticker`, `video`, `video_note`, `voice`, `location`, `contact`, `new_chat_members`, `left_chat_member`, `new_chat_title`, `new_chat_photo`, `delete_chat_photo`, `group_chat_created`, `supergroup_chat_created`, `channel_chat_created`, `migrate_to_chat_id`, `migrate_from_chat_id`, `pinned_message`, `web_app_data`.

You can use some types in one function. Example:

```
content_types=["text", "sticker", "pinned_message", "photo", "audio"]
```

Methods [↗](#)

All [API methods](#) are located in the TeleBot class. They are renamed to follow common Python naming conventions. E.g. `getMe` is renamed to `get_me` and `sendMessage` to `send_message`.

General use of the API [↗](#)

Outlined below are some general use cases of the API.

Message handlers [↗](#)

A message handler is a function that is decorated with the `message_handler` decorator of a TeleBot instance. Message handlers consist of one or multiple filters. Each filter must return `True` for a certain message in order for a message handler to become eligible to handle that message. A message handler is declared in the following way (provided `bot` is an instance of TeleBot):

```
@bot.message_handler(filters)
def function_name(message):
    bot.reply_to(message, "This is a message handler")
```



`function_name` is not bound to any restrictions. Any function name is permitted with message handlers. The function must accept at most one argument, which will be the message that the function must handle. `filters` is a list of keyword arguments. A filter is declared in the following manner: `name=argument`. One handler may have multiple filters. TeleBot supports the following filters:

name	argument(s)	Condition
content_types	list of strings (default <code>['text']</code>)	True if <code>message.content_type</code> is in the list of strings.
regexp	a regular expression as a string	True if <code>re.search(regexp_arg)</code> returns <code>True</code> and <code>message.content_type == 'text'</code> (See Python Regular Expressions)
commands	list of strings	True if <code>message.content_type == 'text'</code> and <code>message.text</code> starts with a command that is in the list of strings.
chat_types	list of chat types	True if <code>message.chat.type</code> in your filter
func	a function (lambda or function reference)	True if the lambda or function reference returns <code>True</code>

Here are some examples of using the filters and message handlers:

```
import telebot
bot = telebot.TeleBot("TOKEN")
```



```

# Handles all text messages that contains the commands '/start' or '/help'.
@bot.message_handler(commands=['start', 'help'])
def handle_start_help(message):
    pass

# Handles all sent documents and audio files
@bot.message_handler(content_types=['document', 'audio'])
def handle_docs_audio(message):
    pass

# Handles all text messages that match the regular expression
@bot.message_handler(regex="SOME_REGEX")
def handle_message(message):
    pass

# Handles all messages for which the lambda returns True
@bot.message_handler(func=lambda message: message.document.mime_type == 'text/plain', content_types=
def handle_text_doc(message):
    pass

# Which could also be defined as:
def test_message(message):
    return message.document.mime_type == 'text/plain'

@bot.message_handler(func=test_message, content_types=['document'])
def handle_text_doc(message):
    pass

# Handlers can be stacked to create a function which will be called if either message_handler is el.
# This handler will be called if the message starts with '/hello' OR is some emoji
@bot.message_handler(commands=['hello'])
@bot.message_handler(func=lambda msg: msg.text.encode("utf-8") == SOME_FANCY_EMOJI)
def send_something(message):
    pass

```

Important: all handlers are tested in the order in which they were declared

Edited Message handler [↗](#)

Handle edited messages @bot.edited_message_handler(filters) # <- passes a Message type object to your function

Channel Post handler [↗](#)

Handle channel post messages @bot.channel_post_handler(filters) # <- passes a Message type object to your function

Edited Channel Post handler [↗](#)

Handle edited channel post messages @bot.edited_channel_post_handler(filters) # <- passes a Message type object to your function

Callback Query Handler [↗](#)

Handle callback queries

```

@bot.callback_query_handler(func=lambda call: True)
def test_callback(call): # <- passes a CallbackQuery type object to your function
    logger.info(call)

```



Shipping Query Handler

Handle shipping queries `@bot.shipping_query_handler()` # <- passes a `ShippingQuery` type object to your function

Pre Checkout Query Handler

Handle pre checkout queries `@bot.pre_checkout_query_handler()` # <- passes a `PreCheckoutQuery` type object to your function

Poll Handler

Handle poll updates `@bot.poll_handler()` # <- passes a `Poll` type object to your function

Poll Answer Handler

Handle poll answers `@bot.poll_answer_handler()` # <- passes a `PollAnswer` type object to your function

My Chat Member Handler

Handle updates of a the bot's member status in a chat `@bot.my_chat_member_handler()` # <- passes a `ChatMemberUpdated` type object to your function

Chat Member Handler

Handle updates of a chat member's status in a chat `@bot.chat_member_handler()` # <- passes a `ChatMemberUpdated` type object to your function *Note: "chat_member" updates are not requested by default. If you want to allow all update types, set `allowed_updates` in `bot.polling()` / `bot.infinity_polling()` to `util.update_types`*

Chat Join Request Handler

Handle chat join requests using: `@bot.chat_join_request_handler()` # <- passes `ChatInviteLink` type object to your function

Inline Mode

More information about [Inline mode](#).

Inline handler

Now, you can use `inline_handler` to get inline queries in telebot.

```
@bot.inline_handler(lambda query: query.query == 'text')
def query_text(inline_query):
    # Query message is text
```



Chosen Inline handler

Use `chosen_inline_handler` to get `chosen_inline_result` in telebot. Don't forgot add the `/setinlinefeedback` command for @Botfather.


More information : [collecting-feedback](#)

```
@bot.chosen_inline_handler(func=lambda chosen_inline_result: True)
def test_chosen(chosen_inline_result):
    # Process all chosen_inline_result.
```



Answer Inline Query

```
@bot.inline_handler(lambda query: query.query == 'text')
def query_text(inline_query):
    try:
        r = types.InlineQueryResultArticle('1', 'Result', types.InputTextMessageContent('Result mes:
        r2 = types.InlineQueryResultArticle('2', 'Result2', types.InputTextMessageContent('Result m
        bot.answer_inline_query(inline_query.id, [r, r2])
    except Exception as e:
        print(e)
```



Additional API features

Middleware Handlers

A middleware handler is a function that allows you to modify requests or the bot context as they pass through the Telegram to the bot. You can imagine middleware as a chain of logic connection handled before any other handlers are executed. Middleware processing is disabled by default, enable it by setting `apihelper.ENABLE_MIDDLEWARE = True`.

```
apihelper.ENABLE_MIDDLEWARE = True
```



```
@bot.middleware_handler(update_types=['message'])
def modify_message(bot_instance, message):
    # modifying the message before it reaches any other handler
    message.another_text = message.text + ':changed'


@bot.message_handler(commands=['start'])
def start(message):
    # the message is already modified when it reaches message handler
    assert message.another_text == message.text + ':changed'
```

There are other examples using middleware handler in the [examples/middleware](#) directory.

Class-based middlewares

There are class-based middlewares. Basic class-based middleware looks like this:

```
class Middleware(BaseMiddleware):
    def __init__(self):
        self.update_types = ['message']
    def pre_process(self, message, data):
        data['foo'] = 'Hello' # just for example
        # we edited the data. now, this data is passed to handler.
        # return SkipHandler() -> this will skip handler
        # return CancelUpdate() -> this will cancel update
    def post_process(self, message, data, exception=None):
        print(data['foo'])
        if exception: # check for exception
            print(exception)
```



Class-based middleware should have to functions: post and pre process. So, as you can see, class-based middlewares work before and after handler execution. For more, check out in [examples](#)

Custom filters

Also, you can use built-in custom filters. Or, you can create your own filter.

Example of custom filter

Also, we have examples on them. Check this links:

You can check some built-in filters in source [code](#)

Example of [filtering by id](#)

Example of [filtering by text](#)

If you want to add some built-in filter, you are welcome to add it in custom_filters.py file.

Here is example of creating filter-class:

```
class IsAdmin(telebot.custom_filters.SimpleCustomFilter):
    # Class will check whether the user is admin or creator in group or not
    key='is_chat_admin'
    @staticmethod
    def check(message: telebot.types.Message):
        return bot.get_chat_member(message.chat.id,message.from_user.id).status in ['administrator']

# To register filter, you need to use method add_custom_filter.
bot.add_custom_filter(IsAdmin())

# Now, you can use it in handler.
@bot.message_handler(is_chat_admin=True)
def admin_of_group(message):
    bot.send_message(message.chat.id, 'You are admin of this group!')
```

☰ README.md

TeleBot [↗](#)

```
import telebot

TOKEN = '<token_string>'
tb = telebot.TeleBot(TOKEN)    #create a new Telegram Bot object

# Upon calling this function, TeleBot starts polling the Telegram servers for new messages.
# - interval: int (default 0) - The interval between polling requests
# - timeout: integer (default 20) - Timeout in seconds for long polling.
# - allowed_updates: List of Strings (default None) - List of update types to request
tb.infinity_polling(interval=0, timeout=20)

# getMe
user = tb.get_me()

# setWebhook
tb.set_webhook(url="http://example.com", certificate=open('mycert.pem'))
# unset webhook
tb.remove_webhook()

# getUpdates
updates = tb.get_updates()
# or
updates = tb.get_updates(1234,100,20) #get_Updates(offset, limit, timeout):

# sendMessage
tb.send_message(chat_id, text)

# editMessageText
tb.edit_message_text(new_text, chat_id, message_id)
```

```

# forwardMessage
tb.forward_message(to_chat_id, from_chat_id, message_id)

# All send_xyz functions which can take a file as an argument, can also take a file_id instead of a
# sendPhoto
photo = open('/tmp/photo.png', 'rb')
tb.send_photo(chat_id, photo)
tb.send_photo(chat_id, "FILEID")

# sendAudio
audio = open('/tmp/audio.mp3', 'rb')
tb.send_audio(chat_id, audio)
tb.send_audio(chat_id, "FILEID")

## sendAudio with duration, performer and title.
tb.send_audio(CHAT_ID, file_data, 1, 'eternnoir', 'pyTelegram')

# sendVoice
voice = open('/tmp/voice.ogg', 'rb')
tb.send_voice(chat_id, voice)
tb.send_voice(chat_id, "FILEID")

# sendDocument
doc = open('/tmp/file.txt', 'rb')
tb.send_document(chat_id, doc)
tb.send_document(chat_id, "FILEID")

# sendSticker
sti = open('/tmp/sti.webp', 'rb')
tb.send_sticker(chat_id, sti)
tb.send_sticker(chat_id, "FILEID")

# sendVideo
video = open('/tmp/video.mp4', 'rb')
tb.send_video(chat_id, video)
tb.send_video(chat_id, "FILEID")

# sendVideoNote
videonote = open('/tmp/videonote.mp4', 'rb')
tb.send_video_note(chat_id, videonote)
tb.send_video_note(chat_id, "FILEID")

# sendLocation
tb.send_location(chat_id, lat, lon)

# sendChatAction
# action_string can be one of the following strings: 'typing', 'upload_photo', 'record_video', 'upload_video',
# 'record_audio', 'upload_audio', 'upload_document' or 'find_location'.
tb.send_chat_action(chat_id, action_string)

# getFile
# Downloading a file is straightforward
# Returns a File object
import requests
file_info = tb.get_file(file_id)

file = requests.get('https://api.telegram.org/file/bot{0}/{1}'.format(API_TOKEN, file_info.file_path))

```

Reply markup [↗](#)

All `send_xyz` functions of `TeleBot` take an optional `reply_markup` argument. This argument must be an instance of `ReplyKeyboardMarkup`, `ReplyKeyboardRemove` or `ForceReply`, which are defined in `types.py`.

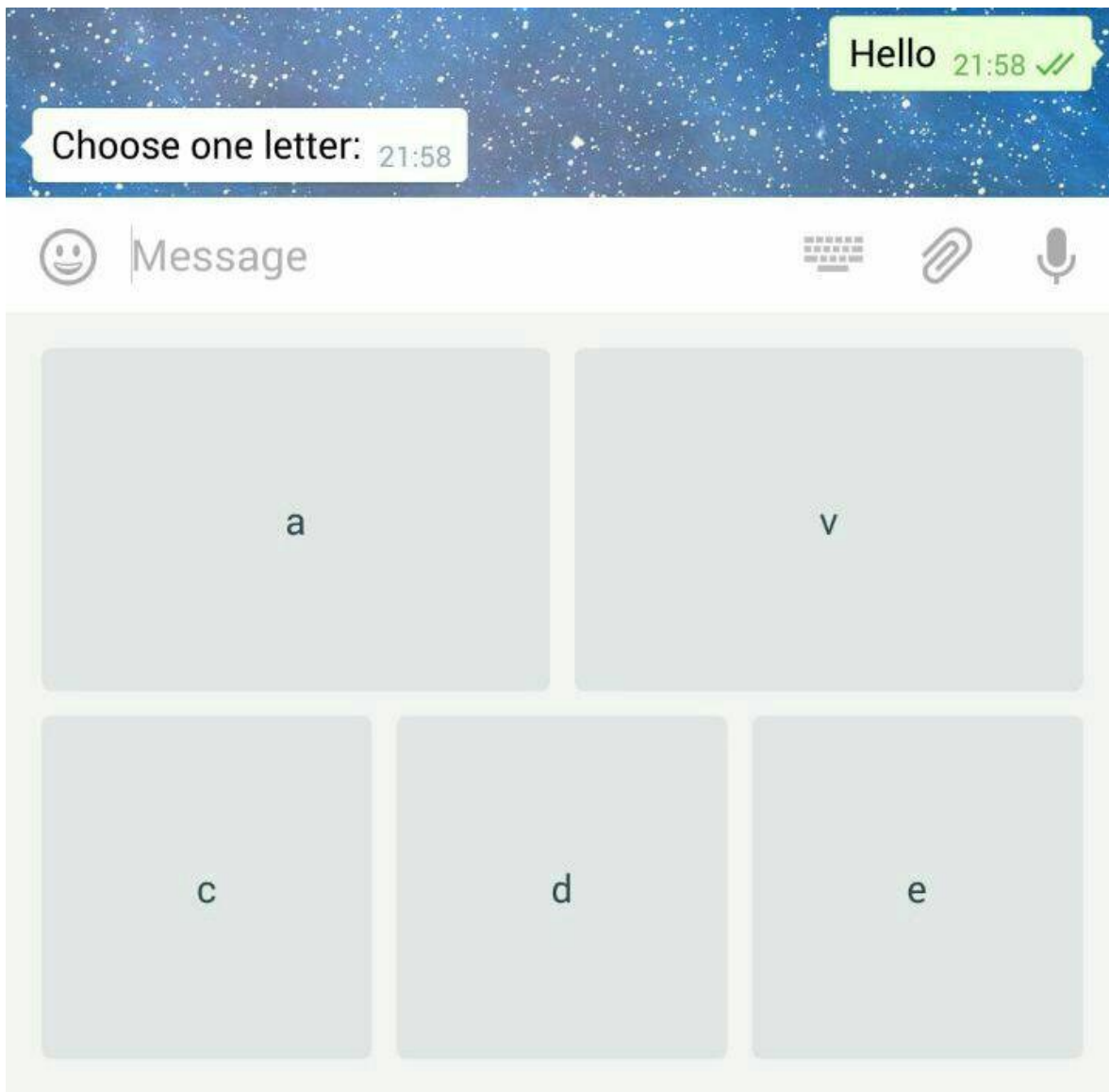


```
from telebot import types

# Using the ReplyKeyboardMarkup class
# It's constructor can take the following optional arguments:
# - resize_keyboard: True/False (default False)
# - one_time_keyboard: True/False (default False)
# - selective: True/False (default False)
# - row_width: integer (default 3)
# row_width is used in combination with the add() function.
# It defines how many buttons are fit on each row before continuing on the next row.
markup = types.ReplyKeyboardMarkup(row_width=2)
itembtn1 = types.KeyboardButton('a')
itembtn2 = types.KeyboardButton('v')
itembtn3 = types.KeyboardButton('d')
markup.add(itembtn1, itembtn2, itembtn3)
tb.send_message(chat_id, "Choose one letter:", reply_markup=markup)

# or add KeyboardButton one row at a time:
markup = types.ReplyKeyboardMarkup()
itembtna = types.KeyboardButton('a')
itembtnv = types.KeyboardButton('v')
itembtnc = types.KeyboardButton('c')
itembtnd = types.KeyboardButton('d')
itembtne = types.KeyboardButton('e')
markup.row(itembtna, itembtnv)
markup.row(itembtnc, itembtnd, itembtne)
tb.send_message(chat_id, "Choose one letter:", reply_markup=markup)
```

The last example yields this result:



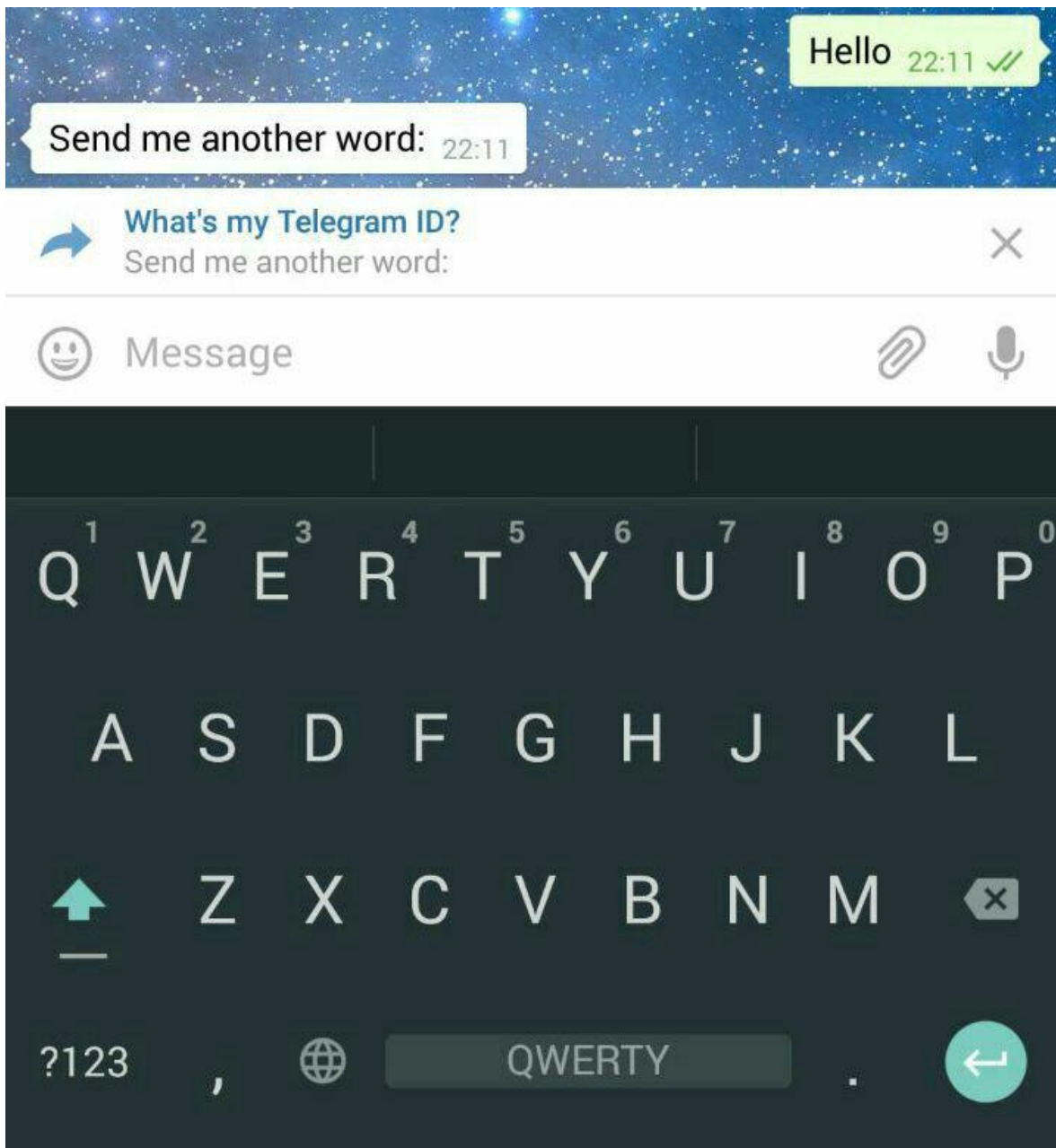
```
# ReplyKeyboardRemove: hides a previously sent ReplyKeyboardMarkup
# Takes an optional selective argument (True/False, default False)
markup = types.ReplyKeyboardRemove(selective=False)
tb.send_message(chat_id, message, reply_markup=markup)
```



```
# ForceReply: forces a user to reply to a message
# Takes an optional selective argument (True/False, default False)
markup = types.ForceReply(selective=False)
tb.send_message(chat_id, "Send me another word:", reply_markup=markup)
```



ForceReply:



Working with entities [↗](#)

This object represents one special entity in a text message. For example, hashtags, usernames, URLs, etc. Attributes:

- type
- url
- offset
- length
- user

Here's an Example: `message.entities[num].<attribute>`

Here `num` is the entity number or order of entity in a reply, for if incase there are multiple entities in the reply/message.

`message.entities` returns a list of entities object.

`message.entities[0].type` would give the type of the first entity

Refer [Bot Api](#) for extra details

Advanced use of the API [↗](#)

Using local Bot API Sever [↗](#)

Since version 5.0 of the Bot API, you have the possibility to run your own [Local Bot API Server](#). pyTelegramBotAPI also supports this feature.

```
from telebot import apihelper

apihelper.API_URL = "http://localhost:4200/bot{0}/{1}"
```



Important: Like described [here](#), you have to log out your bot from the Telegram server before switching to your local API server. in pyTelegramBotAPI use `bot.log_out()`

Note: 4200 is an example port

Asynchronous TeleBot [↗](#)

New: There is an asynchronous implementation of telebot. To enable this behaviour, create an instance of AsyncTeleBot instead of TeleBot.

```
tb = telebot.AsyncTeleBot("TOKEN")
```



Now, every function that calls the Telegram API is executed in a separate asynchronous task. Using AsyncTeleBot allows you to do the following:

```
import telebot

tb = telebot.AsyncTeleBot("TOKEN")

@tb.message_handler(commands=['start'])
async def start_message(message):
    await bot.send_message(message.chat.id, 'Hello!')
```



See more in [examples](#)

Sending large text messages [↗](#)

Sometimes you must send messages that exceed 5000 characters. The Telegram API can not handle that many characters in one request, so we need to split the message in multiples. Here is how to do that using the API:

```
from telebot import util
large_text = open("large_text.txt", "rb").read()

# Split the text each 3000 characters.
# split_string returns a list with the splitted text.
splitted_text = util.split_string(large_text, 3000)

for text in splitted_text:
    tb.send_message(chat_id, text)
```



Or you can use the new `smart_split` function to get more meaningful substrings:

```
from telebot import util
large_text = open("large_text.txt", "rb").read()
# Splits one string into multiple strings, with a maximum amount of `chars_per_string` (max. 4096)
# Splits by last '\n', '. ' or ' ' in exactly this priority.
# smart_split returns a list with the splitted text.
splitted_text = util.smart_split(large_text, chars_per_string=3000)
for text in splitted_text:
    tb.send_message(chat_id, text)
```



Controlling the amount of Threads used by TeleBot [↗](#)

The TeleBot constructor takes the following optional arguments:

- threaded: True/False (default True). A flag to indicate whether TeleBot should execute message handlers on it's polling Thread.

The listener mechanism [↗](#)

As an alternative to the message handlers, one can also register a function as a listener to TeleBot.

NOTICE: handlers won't disappear! Your message will be processed both by handlers and listeners. Also, it's impossible to predict which will work at first because of threading. If you use threaded=False, custom listeners will work earlier, after them handlers will be called. Example:

```
def handle_messages(messages):
    for message in messages:
        # Do something with the message
        bot.reply_to(message, 'Hi')

bot.set_update_listener(handle_messages)
bot.infinity_polling()
```



Using web hooks [↗](#)

When using webhooks telegram sends one Update per call, for processing it you should call `process_new_messages([update.message])` when you receive it.

There are some examples using webhooks in the [examples/webhook_examples](#) directory.

Logging [↗](#)

You can use the Telebot module logger to log debug info about Telebot. Use `telebot.logger` to get the logger of the TeleBot module. It is possible to add custom logging Handlers to the logger. Refer to the [Python logging module page](#) for more info.

```
import logging

logger = telebot.logger
telebot.logger.setLevel(logging.DEBUG) # Outputs debug messages to console.
```



Proxy [↗](#)

For sync:

You can use proxy for request. `apihelper.proxy` object will use by call `requests` `proxies` argument.

```
from telebot import apihelper

apihelper.proxy = {'http': 'http://127.0.0.1:3128'}
```



If you want to use socket5 proxy you need install dependency `pip install requests[socks]` and make sure, that you have the latest version of `gunicorn`, `PySocks`, `pyTelegramBotAPI`, `requests` and `urllib3`.

```
apihelper.proxy = {'https': 'socks5://userproxy:password@proxy_address:port'}
```



For async:

```
from telebot import asyncio_helper

asyncio_helper.proxy = 'http://127.0.0.1:3128' #url
```



Testing [↗](#)

You can disable or change the interaction with real Telegram server by using

```
apihelper.CUSTOM_REQUEST_SENDER = your_handler
```



parameter. You can pass there your own function that will be called instead of `requests.request`.

For example:

```
def custom_sender(method, url, **kwargs):
    print("custom_sender. method: {}, url: {}, params: {}".format(method, url, kwargs.get("params")
    result = util.CustomRequestResponse({'ok': true, "result": {"message_id": 1, "date": 1, "chat": {
    return result
```



Then you can use API and proceed requests in your handler code.

```
apihelper.CUSTOM_REQUEST_SENDER = custom_sender
tb = TeleBot("test")
res = tb.send_message(123, "Test")
```



Result will be:

```
custom_sender. method: post, url: https://api.telegram.org/botololo/sendMessage, params:
{'chat_id': '123', 'text': 'Test'}
```

API conformance limitations [↗](#)

- **+** Bot API 4.5 - No nested MessageEntities and Markdown2 support
- **+** Bot API 4.1 - No Passport support
- **+** Bot API 4.0 - No Passport support

AsyncTeleBot [↗](#)

Asynchronous version of telebot [↗](#)

We have a fully asynchronous version of TeleBot. This class is not controlled by threads. Asyncio tasks are created to execute all the stuff.

EchoBot [↗](#)

Echo Bot example on AsyncTeleBot:

```
# This is a simple echo bot using the decorator mechanism.
# It echoes any incoming text messages.

from telebot.async_telebot import AsyncTeleBot
import asyncio
bot = AsyncTeleBot('TOKEN')

# Handle '/start' and '/help'
@bot.message_handler(commands=['help', 'start'])
async def send_welcome(message):
    await bot.reply_to(message, """\
Hi there, I am EchoBot.
I am here to echo your kind words back to you. Just say anything nice and I'll say the exact same tl
""")

# Handle all other messages with content_type 'text' (content_types defaults to ['text'])
@bot.message_handler(func=lambda message: True)
async def echo_message(message):
    await bot.reply_to(message, message.text)

asyncio.run(bot.polling())
```

As you can see here, keywords are await and async.

Why should I use async? [↗](#)

Asynchronous tasks depend on processor performance. Many asynchronous tasks can run parallelly, while thread tasks will block each other.

Differences in AsyncTeleBot [↗](#)

AsyncTeleBot is asynchronous. It uses aiohttp instead of requests module.

Examples [↗](#)

See more examples in our [examples](#) folder

F.A.Q. [↗](#)

How can I distinguish a User and a GroupChat in message.chat? [↗](#)

Telegram Bot API support new type Chat for message.chat.

- Check the `type` attribute in `Chat` object:



```
if message.chat.type == "private":
    # private chat message

if message.chat.type == "group":
    # group chat message

if message.chat.type == "supergroup":
    # supergroup chat message

if message.chat.type == "channel":
    # channel message
```

How can I handle recurring ConnectionResetErrors? [↗](#)

Bot instances that were idle for a long time might be rejected by the server when sending a message due to a timeout of the last used session. Add `apihelper.SESSION_TIME_TO_LIVE = 5 * 60` to your initialisation to force recreation after 5 minutes without any activity.

The Telegram Chat Group [↗](#)

Get help. Discuss. Chat.

- Join the [pyTelegramBotAPI Telegram Chat Group](#)

Telegram Channel [↗](#)

Join the [News channel](#). Here we will post releases and updates.

More examples [↗](#)

- [Echo Bot](#)
- [Deep Linking](#)
- [next_step_handler Example](#)

Code Template [↗](#)

Template is a ready folder that contains architecture of basic project. Here are some examples of template:

- [AsyncTeleBot template](#)
- [TeleBot template](#)

Bots using this library [↗](#)


- [SiteAlert bot \(source\)](#) by *ilteoood* - Monitors websites and sends a notification on changes
- [TelegramLoggingBot](#) by *aRandomStranger*
- [Telegram LMGTFY_bot](#) by *GabrielRF* - Let me Google that for you.
- [Telegram Proxy Bot](#) by *mrgigabyte*
- [RadRetroRobot](#) by *Tronikart* - Multifunctional Telegram Bot RadRetroRobot.
- [League of Legends bot \(source\)](#) by *i32ropie*
- [NeoBot](#) by *@NeoRanger*
- [ColorCodeBot \(source\)](#) - Share code snippets as beautifully syntax-highlighted HTML and/or images.

- [ComedoresUGRbot \(source\)](#) by [alejandrocq](#) - Telegram bot to check the menu of Universidad de Granada dining hall.
- [proxybot](#) - Simple Proxy Bot for Telegram. by p-hash
- [DonantesMalagaBot](#) - DonantesMalagaBot facilitates information to Malaga blood donors about the places where they can donate today or in the incoming days. It also records the date of the last donation so that it helps the donors to know when they can donate again. - by vfranch
- [DuttyBot](#) by [Dmytryi Striletskyi](#) - Timetable for one university in Kiev.
- [wat-bridge](#) by [rmed](#) - Send and receive messages to/from WhatsApp through Telegram
- [filmratingbot\(source\)](#) by [jcolladosp](#) - Telegram bot using the Python API that gets films rating from IMDb and metacritic
- [Send2Kindlebot \(source\)](#) by [GabrielRF](#) - Send to Kindle service.
- [RastreioBot \(source\)](#) by [GabrielRF](#) - Bot used to track packages on the Brazilian Mail Service.
- [Spbu4UBot\(link\)](#) by [EeOneDown](#) - Bot with timetables for SPbU students.
- [SmartySBot\(link\)](#) by [0xVK](#) - Telegram timetable bot, for Zhytomyr Ivan Franko State University students.
- [LearnIt\(link\)](#) - A Telegram Bot created to help people to memorize other languages' vocabulary.
- [Bot-Telegram-Shodan](#) by [rubenleon](#)
- [VigoBusTelegramBot \(GitHub\)](#) - Bot that provides buses coming to a certain stop and their remaining time for the city of Vigo (Galicia - Spain)
- [kaishnik-bot \(source\)](#) by [airatk](#) - bot which shows all the necessary information to KNTRU-KAI students.
- [Robbie \(source\)](#) by [@FacuM](#) - Support Telegram bot for developers and maintainers.
- [AsadovBot \(source\)](#) by [@DesExcile](#) - Catalog of poems by Eduard Asadov.
- [thesaurus_com_bot \(source\)](#) by [@LeoSvalov](#) - words and synonyms from [dictionary.com](#) and [thesaurus.com](#) in the telegram.
- [InfoBot \(source\)](#) by [@irevenko](#) - An all-round bot that displays some statistics (weather, time, crypto etc...)
- [FoodBot \(source\)](#) by [@Fliego](#) - a simple bot for food ordering
- [Sporty \(source\)](#) by [@0xnu](#) - Telegram bot for displaying the latest news, sports schedules and injury updates.
- [JoinGroup Silencer Bot \(source\)](#) by [@zeph1997](#) - A Telegram Bot to remove "join group" and "removed from group" notifications.
- [TasksListsBot \(source\)](#) by [@Pablo-Davila](#) - A (tasks) lists manager bot for Telegram.
- [MyElizaPsychologistBot \(source\)](#) by [@Pablo-Davila](#) - An implementation of the famous Eliza psychologist chatbot.
- [Frcstbot \(source\)](#) by [Mrsqd](#). A Telegram bot that will always be happy to show you the weather forecast.
- [MineGramBot](#) by [ModischFabrications](#). This bot can start, stop and monitor a minecraft server.
- [Tabletop DiceBot](#) by [dexpiper](#). This bot can roll multiple dices for RPG-like games, add positive and negative modifiers and show short descriptions to the rolls.
- [BarnameKon](#) by [Anvaari](#). This Bot make "Add to google calendar" link for your events. It give information about event and return link. It work for Jalali calendar and in Tehran Time. [Source code](#)
- [Translator bot](#) by [Areeg Fahad](#). This bot can be used to translate texts.
- [Digital Cryptocurrency bot](#) by [Areeg Fahad](#). With this bot, you can now monitor the prices of more than 12 digital Cryptocurrency.
- [Anti-Tracking Bot](#) by [Leon Heess \(source\)](#). Send any link, and the bot tries its best to remove all tracking from the link you sent.
- [Developer Bot](#) by [Vishal Singh \(source code\)](#) This telegram bot can do tasks like GitHub search & clone, provide c++ learning resources ,Stackoverflow search, Codeforces(profile visualizer,random problems)
- [oneIPO bot](#) by [Aadithya](#) & [Amol Soans](#) This Telegram bot provides live updates , data and documents on current and upcoming IPOs(Initial Public Offerings)
- [CoronaGraphsBot \(source\)](#) by [TrevorWinstral](#) - Gets live COVID Country data, plots it, and briefs the user

- [ETHLectureBot \(source\)](#) by [TrevorWinstral](#) - Notifies ETH students when their lectures have been uploaded
- [Vlun Finder Bot](#) by [Resinprotein2333](#). This bot can help you to find The information of CVE vulnerabilities.
- [ETHGasFeeTrackerBot \(Source\)](#) by [DevAdvik](#) - Get Live Ethereum Gas Fees in GWEI
- [Google Sheet Bot](#) by [JoachimStanislaus](#). This bot can help you to track your expenses by uploading your bot entries to your google sheet.
- [GrandQuiz Bot](#) by [Carlosma7](#). This bot is a trivia game that allows you to play with people from different ages. This project addresses the use of a system through chatbots to carry out a social and intergenerational game as an alternative to traditional game development.
- [Diccionario de la RAE \(source\)](#) This bot lets you find difinitions of words in Spanish using [RAE's dictionary](#). It features direct message and inline search.
- [remoteTelegramShell](#) by [EnriqueMoran](#). Control your LinuxOS computer through Telegram.
- [Commerce Telegram Bot](#). Make purchases of items in a store with an Admin panel for data control and notifications.
- [Pyfram-telegram-bot](#) Query wolframalpha.com and make use of its API through Telegram.
- [TranslateThisVideoBot](#) This Bot can understand spoken text in videos and translate it to English
- [Zyprexa \(source\)](#) Zyprexa can solve, help you solve any mathematical problem you encounter and convert your regular mathematical expressions into beautiful imagery using LaTeX.
- [Bincodet-telegram-bot](#) by [tusharhero](#) - Makes [bincodes](#) from text provides and also converts them back to text.
- [hydrolib_bot](#) Toolset for Hydrophilia tabletop game (game cards, rules, structure...).
- [Gugumoe-bot \(source\)](#) by [咕谷酱](#) GuXiaojiang is a multi-functional robot, such as OSU game information query, IP test, animation screenshot search and other functions.
- [Feedback-bot](#) A feedback bot for user-admin communication. Made on AsyncTeleBot, using [template](#).
- [TeleServ](#) by [ablakely](#) This is a Telegram to IRC bridge which links as an IRC server and makes Telegram users appear as native IRC users.
- [Simple Store Bot](#) by [Anton Glyzin](#) This is a simple telegram-store with an admin panel. Designed according to a template.
- [Media Rating Bot \(source\)](#)by [CommanderCRM](#). This bot aggregates media (movies, TV series, etc.) ratings from IMDb, Rotten Tomatoes, Metacritic, TheMovieDB, FilmAffinity and also provides number of votes of said media on IMDb.
- [Spot Seek Bot \(source\)](#) by [Arashnm80](#). This is a free & open source telegram bot for downloading tracks, albums or playlists from spotify.
- [CalendarIT Bot \(source\)](#)by [CodeByZen](#). A simple, but extensible Python Telegram bot, can post acquainted with what is happening today, tomorrow or what happened 20 years ago to channel.
- [DownloadMusicBOT](#) by [Francisco Griman](#) - It is a simple bot that downloads audio from YouTube videos on Telegram.
- [AwesomeChatGPTBot](#) - Simple ChatGTP-3.5 bot. It is FREE and can remember chat history for a while With pre-defined roles!

Want to have your bot listed here? Just make a pull request. Only bots with public source code are accepted.

Releases 61

 4.13.0 - Bot API 6.8 Latest
on Aug 20

+ 60 releases

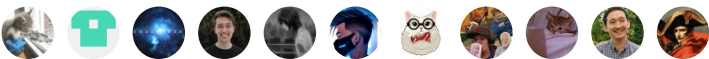
Packages

No packages published

Used by 39.6k



Contributors 203



+ 192 contributors

Languages

● Python 100.0%