aiogram Documentation

Release 2.25.1

Illemius / Alex Root Junior

Aug 06, 2023

CONTENTS

1 Official aiogram resources **3**

2 Features **5**

3 Contribute **7**

4 Contents **9**

4.1 Installation Guide . . . 9
4.2 Quick start . . . 10
4.3 Migration FAQ (1.4 -> 2.0) . . . 12
4.4 Telegram . . . 17
4.5 Dispatcher . . . 170
4.6 Utils . . . 200
4.7 Examples . . . 217
4.8 Contribution . . . 244
4.9 Links . . . 244

5 Indices and tables **245**

Python Module Index **247**

Index **249**

aiogram is a pretty simple and fully asynchronous framework for Telegram Bot API written in Python 3.7 with asyncio and aiohttp. It helps you to make your bots faster and simpler.

CHAPTER

ONE

OFFICIAL AIOGRAM RESOURCES

- News: @aiogram_live
- Community: @aiogram
- Russian community: @aiogram_ru
- Pip: aiogram
- Docs: ReadTheDocs
- Source: Github repo
- Issues/Bug tracker: Github issues tracker
- Test bot: @aiogram_bot

CHAPTER

TWO

FEATURES

- Asynchronous
- Awesome
- Makes things faster
- Has FSM
- Can reply into webhook. (In other words make requests in response to updates)

CHAPTER

THREE

CONTRIBUTE

- Issue Tracker
- Source Code

CHAPTER

FOUR

CONTENTS

4.1 Installation Guide

4.1.1 Using PIP

```
$ pip install -U aiogram
```

4.1.2 Using Pipenv

```
$ pipenv install aiogram
```

4.1.3 Using Pacman

aiogram is also available in Arch Linux Repository, so you can install this framework on any Arch-based distribution like Arch Linux, Antergos, Manjaro, etc. To do this, just use pacman to install the python-aiogram package:

```
$ pacman -S python-aiogram
```

4.1.4 From sources

Development versions:

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

Or if you want to install stable version (The same with version from PyPi):

```
$ git clone https://github.com/aiogram/aiogram.git
$ cd aiogram
$ git checkout master
$ python setup.py install
```

4.1.5 Recommendations

You can speedup your bots by following next instructions:

- Use uvloop instead of default asyncio loop.

  uvloop is a fast, drop-in replacement of the built-in asyncio event loop. uvloop is implemented in Cython and uses libuv under the hood.

  Installation:

  ```
  $ pip install uvloop
  ```

- Use ujson instead of the default json module.

  UltraJSON is an ultra fast JSON encoder and decoder written in pure C with bindings for Python 2.5+ and 3.

  Installation:

  ```
  $ pip install ujson
  ```

- Use aiohttp speedups
  - Use cchardet instead of the chardet module.

    cChardet is a high speed universal character encoding detector.

    Installation:

    ```
    $ pip install cchardet
    ```

  - Use aiodns for speeding up DNS resolving.

    aiodns provides a simple way for doing asynchronous DNS resolutions.

    Installation:

    ```
    $ pip install aiodns
    ```

  - Installing speedups altogether.

    The following will get you `aiohttp` along with `cchardet`, `aiodns` and `brotlipy` in one bundle.

    Installation:

    ```
    $ pip install aiohttp[speedups]
    ```

In addition, you don't need do anything, *aiogram* automatically starts using that if it is found in your environment.

4.2 Quick start

4.2.1 Simple template

At first you have to import all necessary modules

```
import logging

from aiogram import Bot, Dispatcher, executor, types
```

Then you have to initialize bot and dispatcher instances. Bot token you can get from @BotFather

```
API_TOKEN = 'BOT TOKEN HERE'

# Configure logging
logging.basicConfig(level=logging.INFO)

# Initialize bot and dispatcher
bot = Bot(token=API_TOKEN)
dp = Dispatcher(bot)
```

Next step: interaction with bots starts with one command. Register your first command handler:

```
@dp.message_handler(commands=['start', 'help'])
async def send_welcome(message: types.Message):
    """
    This handler will be called when user sends `/start` or `/help` command
    """
    await message.reply("Hi!\nI'm EchoBot!\nPowered by aiogram.")
```

If you want to handle all text messages in the chat simply add handler without filters:

```
@dp.message_handler()
async def echo(message: types.Message):
    # old style:
    # await bot.send_message(message.chat.id, message.text)

    await message.answer(message.text)
```

Last step: run long polling.

```
if __name__ == '__main__':
    executor.start_polling(dp, skip_updates=True)
```

4.2.2 Summary

```
"""
This is a echo bot.
It echoes any incoming text messages.
"""

import logging

from aiogram import Bot, Dispatcher, executor, types

API_TOKEN = 'BOT TOKEN HERE'

# Configure logging
logging.basicConfig(level=logging.INFO)

# Initialize bot and dispatcher
bot = Bot(token=API_TOKEN)
```

(continues on next page)

(continued from previous page)

```
dp = Dispatcher(bot)


@dp.message_handler(commands=['start', 'help'])
async def send_welcome(message: types.Message):
    """
    This handler will be called when user sends `/start` or `/help` command
    """
    await message.reply("Hi!\nI'm EchoBot!\nPowered by aiogram.")



@dp.message_handler()
async def echo(message: types.Message):
    # old style:
    # await bot.send_message(message.chat.id, message.text)

    await message.answer(message.text)


if __name__ == '__main__':
    executor.start_polling(dp, skip_updates=True)
```

4.3 Migration FAQ (1.4 -> 2.0)

This update make breaking changes in aiogram API and drop backward capability with previous versions of framework.

From this point aiogram supports only Python 3.7 and newer.

4.3.1 Changelog

- Used contextvars instead of *aiogram.utils.context*;
- Implemented filters factory;
- Implemented new filters mechanism;
- Allowed to customize command prefix in CommandsFilter;
- Implemented mechanism of passing results from filters (as dicts) as kwargs in handlers (like fixtures in pytest);
- Implemented states group feature;
- Implemented FSM storage's proxy;
- Changed files uploading mechanism;
- Implemented pipe for uploading files from URL;
- Implemented I18n Middleware;
- Errors handlers now should accept only two arguments (current update and exception);
- Used *aiohttp_socks* instead of *aiosocksy* for Socks4/5 proxy;
- *types.ContentType* was divided to *types.ContentType* and *types.ContentTypes*;

- Allowed to use rapidjson instead of ujson/json;
- *.current()* method in bot and dispatcher objects was renamed to *get_current()*;

4.3.2 Instructions

Contextvars

Context utility (*aiogram.utils.context*) now is removed due to new features of Python 3.7 and all subclasses of `aiogram.types.base.TelegramObject`, `aiogram.Bot` and `aiogram.Dispatcher` has *.get_current()* and *.set_current()* methods for getting/setting contextual instances of objects.

Example:

```
async def my_handler(message: types.Message):
    bot = Bot.get_current()
    user = types.User.get_current()
    ...
```

Filters

Custom filters

Now *func* keyword argument can't be used for passing filters to the list of filters instead of that you can pass the filters as arguments:

```
@dp.message_handler(lambda message: message.text == 'foo')
@dp.message_handler(types.ChatType.is_private, my_filter)
async def ...
```

(func filter is still available until v2.1)

Filters factory

Also you can bind your own filters for using as keyword arguments:

```
from aiogram.dispatcher.filters import BoundFilter

class MyFilter(BoundFilter):
    key = 'is_admin'

    def __init__(self, is_admin):
        self.is_admin = is_admin

    async def check(self, message: types.Message):
        member = await bot.get_chat_member(message.chat.id, message.from_user.id)
        return member.is_chat_admin()

dp.filters_factory.bind(MyFilter)

@dp.message_handler(is_admin=True)
async def ...
```

Customize commands prefix

Commands prefix can be changed by following one of two available methods:

```
@dp.message_handler(commands=['admin'], commands_prefix='!/')
@dp.message_handler(Command('admin', prefixes='!/'))
async def ...
```

Passing data from filters as keyword arguments to the handlers

You can pass any data from any filter to the handler by returning `dict` If any key from the received dictionary not in the handler specification the key will be skipped and and will be unavailable from the handler

Before (<=v1.4)

```
async def my_filter(message: types.Message):
    # do something here
    message.conf['foo'] = 'foo'
    message.conf['bar'] = 42
    return True

@dp.message_handler(func=my_filter)
async def my_message_handler(message: types.Message):
    bar = message.conf["bar"]
    await message.reply(f'bar = {bar}')
```

Now (v2.0)

```
async def my_filter(message: types.Message):
    # do something here
    return {'foo': 'foo', 'bar': 42}

@dp.message_handler(my_filter)
async def my_message_handler(message: types.Message, bar: int):
    await message.reply(f'bar = {bar}')
```

Other

Filters can also be used as logical expressions:

```
Text(equals='foo') | Text(endswith='Bar') | ~Text(contains='spam')
```

States group

You can use States objects and States groups instead of string names of the states. String values is still also be available.

Writing states group:

```
from aiogram.dispatcher.filters.state import State, StatesGroup

class UserForm(StatesGroup):
    name = State()  # Will be represented in storage as 'Form:name'
    age = State()  # Will be represented in storage as 'Form:age'
    gender = State()  # Will be represented in storage as 'Form:gender'
```

After that you can use states as *UserForm.name* and etc.

FSM storage's proxy

Now *Dispatcher.current_context()* can't be used as context-manager.

Implemented *FSMContext.proxy()* method which returns asynchronous *FSMContextProxy* context manager and can be used for more simply getting data from the storage.

FSMContextProxy load all user-related data on initialization and dump it to the storage when proxy is closing if any part of the data was changed.

Usage:

```
@dp.message_handler(commands=['click'])
async def cmd_start(message: types.Message, state: FSMContext):
    async with state.proxy() as proxy:  # proxy = FSMContextProxy(state); await proxy.
↪load()
        proxy.setdefault('counter', 0)
        proxy['counter'] += 1
        return await message.reply(f"Counter: {proxy['counter']}")
```

This method is not recommended in high-load solutions in reason named "race-condition".

File uploading mechanism

Fixed uploading files. Removed *BaseBot.send_file* method. This allowed to send the *thumb* field.

Pipe for uploading files from URL

Known issue when Telegram can not accept sending file as URL. In this case need to download file locally and then send.

In this case now you can send file from URL by using pipe. That means you download and send the file without saving it.

You can open the pipe and use for uploading by calling *types.InputFile.from_file(<URL>)*

Example:

```
URL = 'https://docs.aiogram.dev/en/dev-2.x/_static/logo.png'


@dp.message_handler(commands=['image, img'])
async def cmd_image(message: types.Message):
    await bot.send_photo(message.chat.id, types.InputFile.from_url(URL))
```

I18n Middleware

You can internalize your bot by following next steps:

(Code snippets in this example related with *examples/i18n_example.py*)

First usage

1. Extract texts

```
pybabel extract i18n_example.py -o locales/mybot.pot
```

2. Create **.po* files. For e.g. create *en*, *ru*, *uk* locales.
3. Translate texts
4. Compile translations

```
pybabel compile -d locales -D mybot
```

Updating translations

When you change the code of your bot you need to update *po* & *mo* files:

1. Regenerate pot file:

```
pybabel extract i18n_example.py -o locales/mybot.pot
```

2. Update po files

```
pybabel update -d locales -D mybot -i locales/mybot.pot
```

3. Update your translations
4. Compile *mo* files

```
pybabel compile -d locales -D mybot
```

Error handlers

Previously errors handlers had to have three arguments *dispatcher*, *update* and *exception* now *dispatcher* argument is removed and will no longer be passed to the error handlers.

Content types

Content types helper was divided to *types.ContentType* and *types.ContentTypes*.

In filters you can use *types.ContentTypes* but for comparing content types you must use *types.ContentType* class.

4.4 Telegram

4.4.1 Bot object

Low level API

Subclass of this class used only for splitting network interface from all of API methods.

class aiogram.bot.base.**BaseBot**(*token: String, loop: Optional[Union[BaseEventLoop, AbstractEventLoop]] = None, connections_limit: Optional[Integer] = None, proxy: Optional[String] = None, proxy_auth: Optional[BasicAuth] = None, validate_token: Optional[Boolean] = True, parse_mode: Optional[String] = None, disable_web_page_preview: Optional[Boolean] = None, protect_content: Optional[Boolean] = None, timeout: Optional[Union[Integer, Float, ClientTimeout]] = None, server:* TelegramAPIServer *= TelegramAPIServer(base='https://api.telegram.org/bot{token}/{method}', file='https://api.telegram.org/file/bot{token}/{path}')*)

Bases: `object`

Base class for bot. It's raw bot.

Instructions how to get Bot token is found here: https://core.telegram.org/bots#3-how-do-i-create-a-bot

Parameters

- **token** (`str`) – token from @BotFather
- **loop** (Optional Union `asyncio.BaseEventLoop`, `asyncio.AbstractEventLoop`) – event loop
- **connections_limit** (`int`) – connections limit for aiohttp.ClientSession
- **proxy** (`str`) – HTTP proxy URL
- **proxy_auth** (Optional `aiohttp.BasicAuth`) – Authentication information
- **validate_token** (`bool`) – Validate token.
- **parse_mode** (`str`) – You can set default parse mode
- **disable_web_page_preview** (`bool`) – You can set default disable web page preview parameter

- **protect_content** (typing.Optional[base.Boolean]) – Protects the contents of sent messages from forwarding and saving
- **timeout** (typing.Optional[typing.Union[base.Integer, base.Float, aiohttp.ClientTimeout]]) – Request timeout
- **server** (TelegramAPIServer) – Telegram Bot API Server endpoint.

Raise
when token is invalid throw an *aiogram.utils.exceptions.ValidationError*

request_timeout(*timeout: Union[Integer, Float, ClientTimeout]*)

Context manager implements opportunity to change request timeout in current context

Parameters
timeout (typing.Optional[typing.Union[base.Integer, base.Float, aiohttp.ClientTimeout]]) – Request timeout

Returns

close()

Close all client sessions

async request(*method: String*, *data: Optional[Dict] = None*, *files: Optional[Dict] = None*, ***kwargs*) → Union[List, Dict, Boolean]

Make an request to Telegram Bot API

https://core.telegram.org/bots/api#making-requests

Parameters

- **method** (str) – API method
- **data** (dict) – request parameters
- **files** (dict) – files

Returns
result

Return type
Union[List, Dict]

Raise
aiogram.exceptions.TelegramApiError

async download_file(*file_path: ~aiogram.types.base.String*, *destination: ~typing.Optional[~typing.Union[~aiogram.types.base.InputFile, ~pathlib.Path]] = None*, *timeout: ~typing.Optional[~aiogram.types.base.Integer] = <object object>*, *chunk_size: ~typing.Optional[~aiogram.types.base.Integer] = 65536*, *seek: ~typing.Optional[~aiogram.types.base.Boolean] = True*, *destination_dir: ~typing.Optional[~typing.Union[str, ~pathlib.Path]] = None*, *make_dirs: ~typing.Optional[~aiogram.types.base.Boolean] = True*) → Union[BytesIO, FileIO]

Download file by file_path to destination file or directory

if You want to automatically create destination (io.BytesIO) use default value of destination and handle result of this method.

At most one of these parameters can be used: :param destination:, :param destination_dir:

Parameters

- **file_path** (str) – file path on telegram server (You can get it from aiogram.types.File)
- **destination** – filename or instance of io.IOBase. For e. g. io.BytesIO
- **timeout** – Integer
- **chunk_size** – Integer
- **seek** – Boolean - go to start of file when downloading is finished.
- **destination_dir** – directory for saving files
- **make_dirs** – Make dirs if not exist

Returns
destination

async send_file(*file_type*, *method*, *file*, *payload*) → Union[Dict, Boolean]

Send file

https://core.telegram.org/bots/api#inputfile

Parameters

- **file_type** – field name
- **method** – API method
- **file** – String or io.IOBase
- **payload** – request payload

Returns
response

Telegram Bot

This class based on *aiogram.bot.base.BaseBot*

class aiogram.bot.bot.**Bot**(*token: String*, *loop: Optional[Union[BaseEventLoop, AbstractEventLoop]] = None*, *connections_limit: Optional[Integer] = None*, *proxy: Optional[String] = None*, *proxy_auth: Optional[BasicAuth] = None*, *validate_token: Optional[Boolean] = True*, *parse_mode: Optional[String] = None*, *disable_web_page_preview: Optional[Boolean] = None*, *protect_content: Optional[Boolean] = None*, *timeout: Optional[Union[Integer, Float, ClientTimeout]] = None*, *server: TelegramAPIServer = TelegramAPIServer(base='https://api.telegram.org/bot{token}/{method}', file='https://api.telegram.org/file/bot{token}/{path}')*)

Bases: *BaseBot*, DataMixin, ContextInstanceMixin

Base bot class

Instructions how to get Bot token is found here: https://core.telegram.org/bots#3-how-do-i-create-a-bot

Parameters

- **token** (str) – token from @BotFather
- **loop** (Optional Union asyncio.BaseEventLoop, asyncio.AbstractEventLoop) – event loop
- **connections_limit** (int) – connections limit for aiohttp.ClientSession

- **proxy** (`str`) – HTTP proxy URL
- **proxy_auth** (Optional `aiohttp.BasicAuth`) – Authentication information
- **validate_token** (`bool`) – Validate token.
- **parse_mode** (`str`) – You can set default parse mode
- **disable_web_page_preview** (`bool`) – You can set default disable web page preview parameter
- **protect_content** (`typing.Optional[base.Boolean]`) – Protects the contents of sent messages from forwarding and saving
- **timeout** (`typing.Optional[typing.Union[base.Integer, base.Float, aiohttp.ClientTimeout]]`) – Request timeout
- **server** (`TelegramAPIServer`) – Telegram Bot API Server endpoint.

Raise
when token is invalid throw an *`aiogram.utils.exceptions.ValidationError`*

property me: *`User`*

Alias for self.get_me() but lazy and with caching.

Returns
`aiogram.types.User`

async download_file_by_id(*file_id: base.String, destination: Optional[base.InputFile, pathlib.Path] = None, timeout: base.Integer = 30, chunk_size: base.Integer = 65536, seek: base.Boolean = True, destination_dir: Optional[Union[str, pathlib.Path]] = None, make_dirs: Optional[base.Boolean] = True*)

Download file by file_id to destination file or directory

if You want to automatically create destination (`io.BytesIO`) use default value of destination and handle result of this method.

At most one of these parameters can be used: :param destination:, :param destination_dir:

Parameters

- **file_id** – str
- **destination** – filename or instance of `io.IOBase`. For e. g. `io.BytesIO`
- **timeout** – int
- **chunk_size** – int
- **seek** – bool - go to start of file when downloading is finished
- **destination_dir** – directory for saving files
- **make_dirs** – Make dirs if not exist

Returns
destination

async get_updates(*offset: Optional[Integer] = None, limit: Optional[Integer] = None, timeout: Optional[Integer] = None, allowed_updates: Optional[List[String]] = None*) → List[*Update*]

Use this method to receive incoming updates using long polling (wiki).

Notes 1. This method will not work if an outgoing webhook is set up. 2. In order to avoid getting duplicate updates, recalculate offset after each server response.

Source: https://core.telegram.org/bots/api#getupdates

Parameters

- **offset** (typing.Optional[base.Integer]) – Identifier of the first update to be returned
- **limit** (typing.Optional[base.Integer]) – Limits the number of updates to be retrieved
- **timeout** (typing.Optional[base.Integer]) – Timeout in seconds for long polling
- **allowed_updates** (typing.Union[typing.List[base.String], None]) – List the types of updates you want your bot to receive

Returns
An Array of Update objects is returned

Return type
typing.List[types.Update]

async set_webhook(*url: String*, *certificate: Optional[InputFile] = None*, *ip_address: Optional[String] = None*, *max_connections: Optional[Integer] = None*, *allowed_updates: Optional[List[String]] = None*, *drop_pending_updates: Optional[Boolean] = None*, *secret_token: Optional[str] = None*) → Boolean

Use this method to specify a url and receive incoming updates via an outgoing webhook. Whenever there is an update for the bot, we will send an HTTPS POST request to the specified url, containing a JSON-serialized Update. In case of an unsuccessful request, we will give up after a reasonable amount of attempts. Returns True on success.

If you'd like to make sure that the Webhook request comes from Telegram, we recommend using a secret path in the URL, e.g. *https://www.example.com/<token>*. Since nobody else knows your bot's token, you can be pretty sure it's us.

Source: https://core.telegram.org/bots/api#setwebhook

Parameters

- **url** (base.String) – HTTPS url to send updates to. Use an empty string to remove webhook integration
- **certificate** (typing.Optional[base.InputFile]) – Upload your public key certificate so that the root certificate in use can be checked. See our self-signed guide for details: https://core.telegram.org/bots/self-signed
- **ip_address** (typing.Optional[base.String]) – The fixed IP address which will be used to send webhook requests instead of the IP address resolved through DNS
- **max_connections** (typing.Optional[base.Integer]) – Maximum allowed number of simultaneous HTTPS connections to the webhook for update delivery, 1-100. Defaults to 40. Use lower values to limit the load on your bot's server, and higher values to increase your bot's throughput.
- **allowed_updates** (typing.Optional[typing.List[base.String]]) – A list of the update types you want your bot to receive. For example, specify ["message", "edited_channel_post", "callback_query"] to only receive updates of these types. See Update for a complete list of available update types. Specify an empty list to receive all updates regardless of type (default). If not specified, the previous setting will be used. Please note that this parameter doesn't affect updates created before the call to the setWebhook, so unwanted updates may be received for a short period of time.

- **drop_pending_updates** (typing.Optional[base.Boolean]) – Pass True to drop all pending updates
- **secret_token** (typing.Optional[str]) – A secret token to be sent in a header "X-Telegram-Bot-Api-Secret-Token" in every webhook request, 1-256 characters. Only characters A-Z, a-z, 0-9, _ and - are allowed. The header is useful to ensure that the request comes from a webhook set by you.

Returns
Returns true

Return type
base.Boolean

async delete_webhook(*drop_pending_updates: Optional[Boolean] = None*) → Boolean

Use this method to remove webhook integration if you decide to switch back to getUpdates. Returns True on success.

Source: https://core.telegram.org/bots/api#deletewebhook

Parameters
drop_pending_updates (typing.Optional[base.Boolean]) – Pass True to drop all pending updates

Returns
Returns True on success

Return type
base.Boolean

async get_webhook_info() → *WebhookInfo*

Use this method to get current webhook status. Requires no parameters.

If the bot is using getUpdates, will return an object with the url field empty.

Source: https://core.telegram.org/bots/api#getwebhookinfo

Returns
On success, returns a WebhookInfo object

Return type
types.WebhookInfo

async get_me() → *User*

A simple method for testing your bot's auth token. Requires no parameters.

Source: https://core.telegram.org/bots/api#getme

Returns
Returns basic information about the bot in form of a User object

Return type
types.User

async log_out() → Boolean

Use this method to log out from the cloud Bot API server before launching the bot locally. You **must** log out the bot before running it locally, otherwise there is no guarantee that the bot will receive updates. After a successful call, you will not be able to log in again using the same token for 10 minutes. Returns True on success. Requires no parameters.

Source: https://core.telegram.org/bots/api#logout

Returns
Returns True on success

Return type
`base.Boolean`

`close_bot()` → Boolean

Use this method to close the bot instance before moving it from one local server to another. You need to delete the webhook before calling this method to ensure that the bot isn't launched again after server restart. The method will return error 429 in the first 10 minutes after the bot is launched. Returns True on success. Requires no parameters.

Source: https://core.telegram.org/bots/api#close

Returns
Returns True on success

Return type
`base.Boolean`

`async send_message(`*chat_id: Union[Integer, String], text: String, parse_mode: Optional[String] = None, entities: Optional[List[*MessageEntity*]] = None, disable_web_page_preview: Optional[Boolean] = None, message_thread_id: Optional[Integer] = None, disable_notification: Optional[Boolean] = None, protect_content: Optional[Boolean] = None, reply_to_message_id: Optional[Integer] = None, allow_sending_without_reply: Optional[Boolean] = None, reply_markup: Optional[Union[*InlineKeyboardMarkup*, *ReplyKeyboardMarkup*, *ReplyKeyboardRemove*, *ForceReply*]] = None*`)` → *Message*

Use this method to send text messages.

Source: https://core.telegram.org/bots/api#sendmessage

Parameters

- **chat_id** (`typing.Union[base.Integer, base.String]`) – Unique identifier for the target chat or username of the target channel
- **message_thread_id** (`typing.Optional[base.Integer]`) – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **text** (`base.String`) – Text of the message to be sent
- **parse_mode** (`typing.Optional[base.String]`) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.
- **entities** (`typing.Optional[typing.List[types.MessageEntity]]`) – List of special entities that appear in message text, which can be specified instead of parse_mode
- **disable_web_page_preview** (`typing.Optional[base.Boolean]`) – Disables link previews for links in this message
- **disable_notification** (`typing.Optional[base.Boolean]`) – Sends the message silently. Users will receive a notification with no sound
- **protect_content** (`typing.Optional[base.Boolean]`) – Protects the contents of sent messages from forwarding and saving
- **reply_to_message_id** (`typing.Optional[base.Integer]`) – If the message is a reply, ID of the original message

- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user

Returns
On success, the sent Message is returned

Return type
types.Message

async forward_message(*chat_id: Union[Integer, String], from_chat_id: Union[Integer, String], message_id: Integer, message_thread_id: Optional[Integer] = None, disable_notification: Optional[Boolean] = None, protect_content: Optional[Boolean] = None*) → *Message*

Use this method to forward messages of any kind.

Source: https://core.telegram.org/bots/api#forwardmessage

Parameters

- **chat_id** (typing.Union[base.Integer, base.String]) – Unique identifier for the target chat or username of the target channel
- **message_thread_id** (typing.Optional[base.Integer]) – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **from_chat_id** (typing.Union[base.Integer, base.String]) – Unique identifier for the chat where the original message was sent
- **disable_notification** (typing.Optional[base.Boolean]) – Sends the message silently. Users will receive a notification with no sound
- **protect_content** (typing.Optional[base.Boolean]) – Protects the contents of the forwarded message from forwarding and saving
- **message_id** (base.Integer) – Message identifier in the chat specified in from_chat_id

Returns
On success, the sent Message is returned

Return type
types.Message

async copy_message(*chat_id: Union[Integer, String], from_chat_id: Union[Integer, String], message_id: Integer, caption: Optional[String] = None, parse_mode: Optional[String] = None, caption_entities: Optional[List[*MessageEntity*]] = None, message_thread_id: Optional[Integer] = None, disable_notification: Optional[Boolean] = None, protect_content: Optional[Boolean] = None, reply_to_message_id: Optional[Integer] = None, allow_sending_without_reply: Optional[Boolean] = None, reply_markup: Optional[Union[*InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply*]] = None*) → MessageId

Use this method to copy messages of any kind. The method is analogous to the method forwardMessages, but the copied message doesn't have a link to the original message. Returns the MessageId of the sent message on success.

Source: https://core.telegram.org/bots/api#copymessage

Parameters

- **chat_id** (typing.Union[base.Integer, base.String]) – Unique identifier for the target chat or username of the target channel (in the format @channelusername)
- **message_thread_id** (typing.Optional[base.Integer]) – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **from_chat_id** (typing.Union[base.Integer, base.String]) – Unique identifier for the chat where the original message was sent (or channel username in the format @channelusername)
- **message_id** (base.Integer) – Message identifier in the chat specified in from_chat_id
- **caption** (typing.Optional[base.String]) – New caption for media, 0-1024 characters after entities parsing. If not specified, the original caption is kept
- **parse_mode** (typing.Optional[base.String]) – Mode for parsing entities in the new caption. See formatting options for more details: https://core.telegram.org/bots/api#formatting-options
- **caption_entities** (typing.Optional[typing.List[types.MessageEntity]]) – List of special entities that appear in the new caption, which can be specified instead of parse_mode
- **disable_notification** (typing.Optional[base.Boolean]) – Sends the message silently. Users will receive a notification with no sound
- **protect_content** (typing.Optional[base.Boolean]) – Protects the contents of sent messages from forwarding and saving
- **reply_to_message_id** (typing.Optional[base.Integer]) – If the message is a reply, ID of the original message
- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.

Returns

On success, the sent Message is returned

Return type

types.Message

async send_photo(*chat_id: Union[Integer, String], photo: Union[InputFile, String], caption: Optional[String] = None, parse_mode: Optional[String] = None, caption_entities: Optional[List[MessageEntity]] = None, message_thread_id: Optional[Integer] = None, disable_notification: Optional[Boolean] = None, protect_content: Optional[Boolean] = None, reply_to_message_id: Optional[Integer] = None, allow_sending_without_reply: Optional[Boolean] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, has_spoiler: Optional[Boolean] = None*) → *Message*

Use this method to send photos.

Source: https://core.telegram.org/bots/api#sendphoto

Parameters

- **chat_id** (typing.Union[base.Integer, base.String]) – Unique identifier for the target chat or username of the target channel
- **message_thread_id** (typing.Optional[base.Integer]) – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **photo** (typing.Union[base.InputFile, base.String]) – Photo to send
- **caption** (typing.Optional[base.String]) – Photo caption (may also be used when resending photos by file_id), 0-1024 characters
- **parse_mode** (typing.Optional[base.String]) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.
- **caption_entities** (typing.Optional[typing.List[types.MessageEntity]]) – List of special entities that appear in message text, which can be specified instead of parse_mode
- **disable_notification** (typing.Optional[base.Boolean]) – Sends the message silently. Users will receive a notification with no sound
- **protect_content** (typing.Optional[base.Boolean]) – Protects the contents of sent messages from forwarding and saving
- **reply_to_message_id** (typing.Optional[base.Integer]) – If the message is a reply, ID of the original message
- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **has_spoiler** (typing.Optional[base.Boolean]) – Pass True if the photo needs to be covered with a spoiler animation

Returns
On success, the sent Message is returned

Return type
types.Message

async send_audio(*chat_id: Union[Integer, String], audio: Union[InputFile, String], caption: Optional[String] = None, parse_mode: Optional[String] = None, caption_entities: Optional[List[*MessageEntity*]] = None, duration: Optional[Integer] = None, performer: Optional[String] = None, title: Optional[String] = None, thumb: Optional[Union[InputFile, String]] = None, message_thread_id: Optional[Integer] = None, disable_notification: Optional[Boolean] = None, protect_content: Optional[Boolean] = None, reply_to_message_id: Optional[Integer] = None, allow_sending_without_reply: Optional[Boolean] = None, reply_markup: Optional[Union[*InlineKeyboardMarkup*, *ReplyKeyboardMarkup*, *ReplyKeyboardRemove*, *ForceReply*]] = None*) → *Message*

Use this method to send audio files, if you want Telegram clients to display them in the music player. Your audio must be in the .mp3 format.

For sending voice messages, use the sendVoice method instead.

Source: https://core.telegram.org/bots/api#sendaudio

Parameters

- **chat_id** (typing.Union[base.Integer, base.String]) – Unique identifier for the target chat or username of the target channel
- **message_thread_id** (typing.Optional[base.Integer]) – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **audio** (typing.Union[base.InputFile, base.String]) – Audio file to send
- **caption** (typing.Optional[base.String]) – Audio caption, 0-1024 characters
- **parse_mode** (typing.Optional[base.String]) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.
- **caption_entities** (typing.Optional[typing.List[types.MessageEntity]]) – List of special entities that appear in message text, which can be specified instead of parse_mode
- **duration** (typing.Optional[base.Integer]) – Duration of the audio in seconds
- **performer** (typing.Optional[base.String]) – Performer
- **title** (typing.Optional[base.String]) – Track name
- **thumb** (typing.Union[base.InputFile, base.String, None]) – Thumbnail of the file sent
- **disable_notification** (typing.Optional[base.Boolean]) – Sends the message silently. Users will receive a notification with no sound
- **protect_content** (typing.Optional[base.Boolean]) – Protects the contents of sent messages from forwarding and saving
- **reply_to_message_id** (typing.Optional[base.Integer]) – If the message is a reply, ID of the original message
- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user

Returns

On success, the sent Message is returned

Return type

types.Message

async send_document(*chat_id: Union[Integer, String], document: Union[InputFile, String], thumb: Optional[Union[InputFile, String]] = None, caption: Optional[String] = None, parse_mode: Optional[String] = None, caption_entities: Optional[List[*MessageEntity*]] = None, disable_content_type_detection: Optional[Boolean] = None, message_thread_id: Optional[Integer] = None, disable_notification: Optional[Boolean] = None, protect_content: Optional[Boolean] = None, reply_to_message_id: Optional[Integer] = None, allow_sending_without_reply: Optional[Boolean] = None, reply_markup: Optional[Union[*InlineKeyboardMarkup*, *ReplyKeyboardMarkup*, *ReplyKeyboardRemove*, *ForceReply*]] = None*) → *Message*

Use this method to send general files. On success, the sent Message is returned. Bots can currently send files of any type of up to 50 MB in size, this limit may be changed in the future.

Source: https://core.telegram.org/bots/api#senddocument

Parameters

- **chat_id** (typing.Union[base.Integer, base.String]) – Unique identifier for the target chat or username of the target channel
- **message_thread_id** (typing.Optional[base.Integer]) – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **document** (typing.Union[base.InputFile, base.String]) – File to send
- **thumb** (typing.Union[base.InputFile, base.String, None]) – Thumbnail of the file sent
- **caption** (typing.Optional[base.String]) – Document caption (may also be used when resending documents by file_id), 0-1024 characters
- **disable_content_type_detection** (typing.Optional[base.Boolean]) – Disables automatic server-side content type detection for files uploaded using multipart/form-data
- **parse_mode** (typing.Optional[base.String]) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.
- **caption_entities** (typing.Optional[typing.List[types.MessageEntity]]) – List of special entities that appear in message text, which can be specified instead of parse_mode
- **disable_notification** (typing.Optional[base.Boolean]) – Sends the message silently. Users will receive a notification with no sound
- **protect_content** (typing.Optional[base.Boolean]) – Protects the contents of sent messages from forwarding and saving
- **reply_to_message_id** (typing.Optional[base.Integer]) – If the message is a reply, ID of the original message
- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply], None]) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user

Returns

On success, the sent Message is returned

Return type

types.Message

async send_video(*chat_id: Union[Integer, String], video: Union[InputFile, String], duration: Optional[Integer] = None, width: Optional[Integer] = None, height: Optional[Integer] = None, thumb: Optional[Union[InputFile, String]] = None, caption: Optional[String] = None, parse_mode: Optional[String] = None, caption_entities: Optional[List[MessageEntity]] = None, supports_streaming: Optional[Boolean] = None, message_thread_id: Optional[Integer] = None, disable_notification: Optional[Boolean] = None, protect_content: Optional[Boolean] = None, reply_to_message_id: Optional[Integer] = None, allow_sending_without_reply: Optional[Boolean] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, has_spoiler: Optional[Boolean] = None*) → *Message*

Use this method to send video files, Telegram clients support mp4 videos (other formats may be sent as Document).

Source: https://core.telegram.org/bots/api#sendvideo

Parameters

- **chat_id** (typing.Union[base.Integer, base.String]) – Unique identifier for the target chat or username of the target channel
- **message_thread_id** (typing.Optional[base.Integer]) – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **video** (typing.Union[base.InputFile, base.String]) – Video to send
- **duration** (typing.Optional[base.Integer]) – Duration of sent video in seconds
- **width** (typing.Optional[base.Integer]) – Video width
- **height** (typing.Optional[base.Integer]) – Video height
- **thumb** (typing.Union[base.InputFile, base.String, None]) – Thumbnail of the file sent
- **caption** (typing.Optional[base.String]) – Video caption (may also be used when resending videos by file_id), 0-1024 characters
- **parse_mode** (typing.Optional[base.String]) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.
- **caption_entities** (typing.Optional[typing.List[types.MessageEntity]]) – List of special entities that appear in message text, which can be specified instead of parse_mode
- **supports_streaming** (typing.Optional[base.Boolean]) – Pass True, if the uploaded video is suitable for streaming
- **disable_notification** (typing.Optional[base.Boolean]) – Sends the message silently. Users will receive a notification with no sound
- **protect_content** (typing.Optional[base.Boolean]) – Protects the contents of sent messages from forwarding and saving
- **reply_to_message_id** (typing.Optional[base.Integer]) – If the message is a reply, ID of the original message
- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply,

None]) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user

- **has_spoiler** (typing.Optional[base.Boolean]) – Pass True if the video needs to be covered with a spoiler animation

Returns
On success, the sent Message is returned

Return type
types.Message

async send_animation(*chat_id: Union[Integer, String], animation: Union[InputFile, String], duration: Optional[Integer] = None, width: Optional[Integer] = None, height: Optional[Integer] = None, thumb: Optional[Union[InputFile, String]] = None, caption: Optional[String] = None, parse_mode: Optional[String] = None, caption_entities: Optional[List[MessageEntity]] = None, message_thread_id: Optional[Integer] = None, disable_notification: Optional[Boolean] = None, protect_content: Optional[Boolean] = None, reply_to_message_id: Optional[Integer] = None, allow_sending_without_reply: Optional[Boolean] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, has_spoiler: Optional[Boolean] = None*) → *Message*

Use this method to send animation files (GIF or H.264/MPEG-4 AVC video without sound).

On success, the sent Message is returned. Bots can currently send animation files of up to 50 MB in size, this limit may be changed in the future.

Source https://core.telegram.org/bots/api#sendanimation

Parameters

- **chat_id** (typing.Union[base.Integer, base.String]) – Unique identifier for the target chat or username of the target channel (in the format @channelusername)
- **message_thread_id** (typing.Optional[base.Integer]) – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **animation** (typing.Union[base.InputFile, base.String]) – Animation to send. Pass a file_id as String to send an animation that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get an animation from the Internet, or upload a new animation using multipart/form-data
- **duration** (typing.Optional[base.Integer]) – Duration of sent animation in seconds
- **width** (typing.Optional[base.Integer]) – Animation width
- **height** (typing.Optional[base.Integer]) – Animation height
- **thumb** (typing.Union[typing.Union[base.InputFile, base.String], None]) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320.
- **caption** (typing.Optional[base.String]) – Animation caption (may also be used when resending animation by file_id), 0-1024 characters
- **parse_mode** (typing.Optional[base.String]) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption

- **caption_entities** (typing.Optional[typing.List[types.MessageEntity]]) – List of special entities that appear in message text, which can be specified instead of parse_mode
- **disable_notification** (typing.Optional[base.Boolean]) – Sends the message silently. Users will receive a notification with no sound
- **protect_content** (typing.Optional[base.Boolean]) – Protects the contents of sent messages from forwarding and saving
- **reply_to_message_id** (typing.Optional[base.Integer]) – If the message is a reply, ID of the original message
- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (typing.Union[typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply], None]) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **has_spoiler** (typing.Optional[base.Boolean]) – Pass True if the animation needs to be covered with a spoiler animation

Returns
On success, the sent Message is returned

Return type
types.Message

async send_voice(*chat_id: Union[Integer, String], voice: Union[InputFile, String], caption: Optional[String] = None, parse_mode: Optional[String] = None, caption_entities: Optional[List[MessageEntity]] = None, duration: Optional[Integer] = None, message_thread_id: Optional[Integer] = None, disable_notification: Optional[Boolean] = None, protect_content: Optional[Boolean] = None, reply_to_message_id: Optional[Integer] = None, allow_sending_without_reply: Optional[Boolean] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None*) → *Message*

Use this method to send audio files, if you want Telegram clients to display the file as a playable voice message.

For this to work, your audio must be in an .ogg file encoded with OPUS (other formats may be sent as Audio or Document).

Source: https://core.telegram.org/bots/api#sendvoice

Parameters

- **chat_id** (typing.Union[base.Integer, base.String]) – Unique identifier for the target chat or username of the target channel
- **message_thread_id** (typing.Optional[base.Integer]) – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **voice** (typing.Union[base.InputFile, base.String]) – Audio file to send
- **caption** (typing.Optional[base.String]) – Voice message caption, 0-1024 characters

- **parse_mode** (typing.Optional[base.String]) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.
- **caption_entities** (typing.Optional[typing.List[types.MessageEntity]]) – List of special entities that appear in message text, which can be specified instead of parse_mode
- **duration** (typing.Optional[base.Integer]) – Duration of the voice message in seconds
- **disable_notification** (typing.Optional[base.Boolean]) – Sends the message silently. Users will receive a notification with no sound
- **protect_content** (typing.Optional[base.Boolean]) – Protects the contents of sent messages from forwarding and saving
- **reply_to_message_id** (typing.Optional[base.Integer]) – If the message is a reply, ID of the original message
- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user

Returns
On success, the sent Message is returned

Return type
types.Message

async send_video_note(*chat_id: Union[Integer, String], video_note: Union[InputFile, String], duration: Optional[Integer] = None, length: Optional[Integer] = None, thumb: Optional[Union[InputFile, String]] = None, message_thread_id: Optional[Integer] = None, disable_notification: Optional[Boolean] = None, protect_content: Optional[Boolean] = None, reply_to_message_id: Optional[Integer] = None, allow_sending_without_reply: Optional[Boolean] = None, reply_markup: Optional[Union[*InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply*]] = None*) → *Message*

As of v.4.0, Telegram clients support rounded square mp4 videos of up to 1 minute long. Use this method to send video messages.

Source: https://core.telegram.org/bots/api#sendvideonote

Parameters

- **chat_id** (typing.Union[base.Integer, base.String]) – Unique identifier for the target chat or username of the target channel
- **message_thread_id** (typing.Optional[base.Integer]) – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **video_note** (typing.Union[base.InputFile, base.String]) – Video note to send
- **duration** (typing.Optional[base.Integer]) – Duration of sent video in seconds
- **length** (typing.Optional[base.Integer]) – Video width and height

- **thumb** (typing.Union[base.InputFile, base.String, None]) – Thumbnail of the file sent
- **disable_notification** (typing.Optional[base.Boolean]) – Sends the message silently. Users will receive a notification with no sound
- **protect_content** (typing.Optional[base.Boolean]) – Protects the contents of sent messages from forwarding and saving
- **reply_to_message_id** (typing.Optional[base.Integer]) – If the message is a reply, ID of the original message
- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user

Returns
On success, the sent Message is returned

Return type
types.Message

async send_media_group(*chat_id: Union[Integer, String]*, *media: Union[*MediaGroup*, List]*, *message_thread_id: Optional[Integer] = None*, *disable_notification: Optional[Boolean] = None*, *protect_content: Optional[Boolean] = None*, *reply_to_message_id: Optional[Integer] = None*, *allow_sending_without_reply: Optional[Boolean] = None*) → List[*Message*]

Use this method to send a group of photos, videos, documents or audios as an album. Documents and audio files can be only group in an album with messages of the same type. On success, an array of Messages that were sent is returned.

Source: https://core.telegram.org/bots/api#sendmediagroup

Parameters

- **chat_id** (typing.Union[base.Integer, base.String]) – Unique identifier for the target chat or username of the target channel (in the format @channelusername)
- **message_thread_id** (typing.Optional[base.Integer]) – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **media** (typing.Union[types.MediaGroup, typing.List]) – A JSON-serialized array describing messages to be sent, must include 2-10 items
- **disable_notification** (typing.Optional[base.Boolean]) – Sends messages silently. Users will receive a notification with no sound.
- **protect_content** (typing.Optional[base.Boolean]) – Protects the contents of sent messages from forwarding and saving
- **reply_to_message_id** (typing.Optional[base.Integer]) – If the messages are a reply, ID of the original message
- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found

Returns
On success, an array of the sent Messages is returned

Return type
List[types.Message]

async send_location(*chat_id: Union[Integer, String]*, *latitude: Float*, *longitude: Float*, *horizontal_accuracy: Optional[Float] = None*, *live_period: Optional[Integer] = None*, *heading: Optional[Integer] = None*, *proximity_alert_radius: Optional[Integer] = None*, *message_thread_id: Optional[Integer] = None*, *disable_notification: Optional[Boolean] = None*, *protect_content: Optional[Boolean] = None*, *reply_to_message_id: Optional[Integer] = None*, *allow_sending_without_reply: Optional[Boolean] = None*, *reply_markup: Optional[Union[*InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply*]] = None*) → *Message*

Use this method to send point on the map.

Source: https://core.telegram.org/bots/api#sendlocation

Parameters

- **chat_id** (typing.Union[base.Integer, base.String]) – Unique identifier for the target chat or username of the target channel
- **message_thread_id** (typing.Optional[base.Integer]) – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **latitude** (base.Float) – Latitude of the location
- **longitude** (base.Float) – Longitude of the location
- **horizontal_accuracy** (typing.Optional[base.Float]) – The radius of uncertainty for the location, measured in meters; 0-1500
- **live_period** (typing.Optional[base.Integer]) – Period in seconds for which the location will be updated
- **heading** (typing.Optional[base.Integer]) – For live locations, a direction in which the user is moving, in degrees. Must be between 1 and 360 if specified.
- **proximity_alert_radius** (typing.Optional[base.Integer]) – For live locations, a maximum distance for proximity alerts about approaching another chat member, in meters. Must be between 1 and 100000 if specified.
- **disable_notification** (typing.Optional[base.Boolean]) – Sends the message silently. Users will receive a notification with no sound
- **protect_content** (typing.Optional[base.Boolean]) – Protects the contents of sent messages from forwarding and saving
- **reply_to_message_id** (typing.Optional[base.Integer]) – If the message is a reply, ID of the original message
- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user

Returns
On success, the sent Message is returned

Return type
`types.Message`

async edit_message_live_location(*latitude: Float, longitude: Float, chat_id: Optional[Union[Integer, String]] = None, message_id: Optional[Integer] = None, inline_message_id: Optional[String] = None, horizontal_accuracy: Optional[Float] = None, heading: Optional[Integer] = None, proximity_alert_radius: Optional[Integer] = None, reply_markup: Optional[InlineKeyboardMarkup] = None*) → *Message*

Use this method to edit live location messages sent by the bot or via the bot (for inline bots). A location can be edited until its live_period expires or editing is explicitly disabled by a call to stopMessageLiveLocation.

Source: https://core.telegram.org/bots/api#editmessagelivelocation

Parameters

- **chat_id** (`typing.Union[base.Integer, base.String, None]`) – Required if inline_message_id is not specified
- **message_id** (`typing.Optional[base.Integer]`) – Required if inline_message_id is not specified. Identifier of the sent message
- **inline_message_id** (`typing.Optional[base.String]`) – Required if chat_id and message_id are not specified. Identifier of the inline message
- **latitude** (`base.Float`) – Latitude of new location
- **longitude** (`base.Float`) – Longitude of new location
- **horizontal_accuracy** (`typing.Optional[base.Float]`) – The radius of uncertainty for the location, measured in meters; 0-1500
- **heading** (`typing.Optional[base.Integer]`) – Direction in which the user is moving, in degrees. Must be between 1 and 360 if specified.
- **proximity_alert_radius** (`typing.Optional[base.Integer]`) – For live locations, a maximum distance for proximity alerts about approaching another chat member, in meters. Must be between 1 and 100000 if specified.
- **reply_markup** (`typing.Optional[types.InlineKeyboardMarkup]`) – A JSON-serialized object for a new inline keyboard

Returns
On success, if the edited message was sent by the bot, the edited Message is returned, otherwise True is returned.

Return type
`typing.Union[types.Message, base.Boolean]`

async stop_message_live_location(*chat_id: Optional[Union[Integer, String]] = None, message_id: Optional[Integer] = None, inline_message_id: Optional[String] = None, reply_markup: Optional[InlineKeyboardMarkup] = None*) → *Message*

Use this method to stop updating a live location message sent by the bot or via the bot (for inline bots) before live_period expires.

Source: https://core.telegram.org/bots/api#stopmessagelivelocation

Parameters

- **chat_id** (`typing.Union[base.Integer, base.String, None]`) – Required if inline_message_id is not specified

- **message_id** (typing.Optional[base.Integer]) – Required if inline_message_id is not specified. Identifier of the sent message
- **inline_message_id** (typing.Optional[base.String]) – Required if chat_id and message_id are not specified. Identifier of the inline message
- **reply_markup** (typing.Optional[types.InlineKeyboardMarkup]) – A JSON-serialized object for a new inline keyboard

Returns
On success, if the message was sent by the bot, the sent Message is returned, otherwise True is returned.

Return type
typing.Union[types.Message, base.Boolean]

async send_venue(*chat_id: Union[Integer, String]*, *latitude: Float*, *longitude: Float*, *title: String*, *address: String*, *foursquare_id: Optional[String] = None*, *foursquare_type: Optional[String] = None*, *google_place_id: Optional[String] = None*, *google_place_type: Optional[String] = None*, *message_thread_id: Optional[Integer] = None*, *disable_notification: Optional[Boolean] = None*, *protect_content: Optional[Boolean] = None*, *reply_to_message_id: Optional[Integer] = None*, *allow_sending_without_reply: Optional[Boolean] = None*, *reply_markup: Optional[Union[*InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply*]] = None*) → *Message*

Use this method to send information about a venue.

Source: https://core.telegram.org/bots/api#sendvenue

Parameters

- **chat_id** (typing.Union[base.Integer, base.String]) – Unique identifier for the target chat or username of the target channel (in the format @channelusername)
- **message_thread_id** (typing.Optional[base.Integer]) – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **latitude** (base.Float) – Latitude of the venue
- **longitude** (base.Float) – Longitude of the venue
- **title** (base.String) – Name of the venue
- **address** (base.String) – Address of the venue
- **foursquare_id** (typing.Optional[base.String]) – Foursquare identifier of the venue
- **foursquare_type** (typing.Optional[base.String]) – Foursquare type of the venue, if known
- **google_place_id** (typing.Optional[base.String]) – Google Places identifier of the venue
- **google_place_type** (typing.Optional[base.String]) – Google Places type of the venue. See supported types: https://developers.google.com/places/web-service/supported_types
- **disable_notification** (typing.Optional[base.Boolean]) – Sends the message silently. Users will receive a notification with no sound
- **protect_content** (typing.Optional[base.Boolean]) – Protects the contents of sent messages from forwarding and saving

- **reply_to_message_id** (typing.Optional[base.Integer]) – If the message is a reply, ID of the original message
- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user

Returns
On success, the sent Message is returned

Return type
types.Message

async send_contact(*chat_id: Union[Integer, String], phone_number: String, first_name: String, last_name: Optional[String] = None, vcard: Optional[String] = None, message_thread_id: Optional[Integer] = None, disable_notification: Optional[Boolean] = None, protect_content: Optional[Boolean] = None, reply_to_message_id: Optional[Integer] = None, allow_sending_without_reply: Optional[Boolean] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None*) → *Message*

Use this method to send phone contacts.

Source: https://core.telegram.org/bots/api#sendcontact

Parameters

- **chat_id** (typing.Union[base.Integer, base.String]) – Unique identifier for the target chat or username of the target channel
- **message_thread_id** (typing.Optional[base.Integer]) – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **phone_number** (base.String) – Contact's phone number
- **first_name** (base.String) – Contact's first name
- **last_name** (typing.Optional[base.String]) – Contact's last name
- **vcard** (typing.Optional[base.String]) – vcard
- **disable_notification** (typing.Optional[base.Boolean]) – Sends the message silently. Users will receive a notification with no sound
- **protect_content** (typing.Optional[base.Boolean]) – Protects the contents of sent messages from forwarding and saving
- **reply_to_message_id** (typing.Optional[base.Integer]) – If the message is a reply, ID of the original message
- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user

Returns
On success, the sent Message is returned

Return type
`types.Message`

async send_poll(*chat_id: Union[Integer, String], question: String, options: List[String], is_anonymous: Optional[Boolean] = None, type: Optional[String] = None, allows_multiple_answers: Optional[Boolean] = None, correct_option_id: Optional[Integer] = None, explanation: Optional[String] = None, explanation_parse_mode: Optional[String] = None, explanation_entities: Optional[List[MessageEntity]] = None, open_period: Optional[Integer] = None, close_date: Optional[Union[Integer, datetime, timedelta]] = None, is_closed: Optional[Boolean] = None, message_thread_id: Optional[Integer] = None, disable_notification: Optional[Boolean] = None, protect_content: Optional[Boolean] = None, reply_to_message_id: Optional[Integer] = None, allow_sending_without_reply: Optional[Boolean] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None*) → *Message*

Use this method to send a native poll. On success, the sent Message is returned.

Source: https://core.telegram.org/bots/api#sendpoll

Parameters

- **chat_id** (`typing.Union[base.Integer, base.String]`) – Unique identifier for the target chat or username of the target channel (in the format @channelusername)
- **message_thread_id** (`typing.Optional[base.Integer]`) – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **question** (`base.String`) – Poll question, 1-300 characters
- **options** (`typing.List[base.String]`) – A list of answer options, 2-10 strings 1-100 characters each
- **is_anonymous** (`typing.Optional[base.Boolean]`) – True, if the poll needs to be anonymous, defaults to True
- **type** (`typing.Optional[base.String]`) – Poll type, "quiz" or "regular", defaults to "regular"
- **allows_multiple_answers** (`typing.Optional[base.Boolean]`) – True, if the poll allows multiple answers, ignored for polls in quiz mode, defaults to False
- **correct_option_id** (`typing.Optional[base.Integer]`) – 0-based identifier of the correct answer option, required for polls in quiz mode
- **explanation** (`typing.Optional[base.String]`) – Text that is shown when a user chooses an incorrect answer or taps on the lamp icon in a quiz-style poll, 0-200 characters with at most 2 line feeds after entities parsing
- **explanation_parse_mode** (`typing.Optional[base.String]`) – Mode for parsing entities in the explanation. See formatting options for more details.
- **explanation_entities** (`typing.Optional[typing.List[types.MessageEntity]]`) – List of special entities that appear in message text, which can be specified instead of parse_mode
- **open_period** (`typing.Optional[base.Integer]`) – Amount of time in seconds the poll will be active after creation, 5-600. Can't be used together with close_date.

- **close_date** (typing.Union[base.Integer, datetime.datetime, datetime.timedelta, None]) – Point in time (Unix timestamp) when the poll will be automatically closed. Must be at least 5 and no more than 600 seconds in

  the future. Can't be used together with open_period.

- **is_closed** (typing.Optional[base.Boolean]) – Pass True, if the poll needs to be immediately closed
- **disable_notification** (typing.Optional[Boolean]) – Sends the message silently. Users will receive a notification with no sound.
- **protect_content** (typing.Optional[base.Boolean]) – Protects the contents of sent messages from forwarding and saving
- **reply_to_message_id** (typing.Optional[Integer]) – If the message is a reply, ID of the original message
- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user

Returns
On success, the sent Message is returned

Return type
types.Message

async send_dice(*chat_id: Union[Integer, String], message_thread_id: Optional[Integer] = None, disable_notification: Optional[Boolean] = None, protect_content: Optional[Boolean] = None, emoji: Optional[String] = None, reply_to_message_id: Optional[Integer] = None, allow_sending_without_reply: Optional[Boolean] = None, reply_markup: Optional[Union[*InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply*]] = None*) → *Message*

Use this method to send an animated emoji that will display a random value. On success, the sent Message is returned.

Source: https://core.telegram.org/bots/api#senddice

Parameters

- **chat_id** (typing.Union[base.Integer, base.String]) – Unique identifier for the target chat or username of the target channel (in the format @channelusername)
- **message_thread_id** (typing.Optional[base.Integer]) – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **emoji** (typing.Optional[base.String]) – Emoji on which the dice throw animation is based. Currently, must be one of "", "", "", "", or "". Dice can have values 1-6 for "" and "", values 1-5 for "" and "", and values 1-64 for "". Defaults to ""
- **disable_notification** (typing.Optional[base.Boolean]) – Sends the message silently. Users will receive a notification with no sound
- **protect_content** (typing.Optional[base.Boolean]) – Protects the contents of sent messages from forwarding and saving

- **reply_to_message_id** (typing.Optional[base.Integer]) – If the message is a reply, ID of the original message
- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user

Returns
On success, the sent Message is returned

Return type
types.Message

async send_chat_action(*chat_id: Union[Integer, String]*, *action: String*, *message_thread_id: Optional[Integer] = None*) → Boolean

Use this method when you need to tell the user that something is happening on the bot's side. The status is set for 5 seconds or less (when a message arrives from your bot, Telegram clients clear its typing status). Returns True on success.

Example: The ImageBot needs some time to process a request and upload the image. Instead of sending a text message along the lines of "Retrieving image, please wait…", the bot may use sendChatAction with action = upload_photo. The user will see a "sending photo" status for the bot.

We only recommend using this method when a response from the bot will take a noticeable amount of time to arrive.

Source: https://core.telegram.org/bots/api#sendchataction

Parameters

- **chat_id** (typing.Union[base.Integer, base.String]) – Unique identifier for the target chat or username of the target channel (in the format @channelusername)
- **action** (base.String) – Type of action to broadcast. Choose one, depending on what the user is about to receive: *typing* for text messages, *upload_photo* for photos, *record_video* or *upload_video* for videos, *record_voice* or *upload_voice* for voice notes, *upload_document* for general files, *find_location* for location data, *record_video_note* or *upload_video_note* for video notes.
- **message_thread_id** (typing.Optional[base.Integer]) – Unique identifier for the target message thread; supergroups only

Returns
Returns True on success

Return type
base.Boolean

async get_user_profile_photos(*user_id: Integer*, *offset: Optional[Integer] = None*, *limit: Optional[Integer] = None*) → *UserProfilePhotos*

Use this method to get a list of profile pictures for a user. Returns a UserProfilePhotos object.

Source: https://core.telegram.org/bots/api#getuserprofilephotos

Parameters

- **user_id** (base.Integer) – Unique identifier of the target user

- **offset** (typing.Optional[base.Integer]) – Sequential number of the first photo to be returned. By default, all photos are returned
- **limit** (typing.Optional[base.Integer]) – Limits the number of photos to be retrieved. Values between 1—100 are accepted. Defaults to 100

Returns
Returns a UserProfilePhotos object

Return type
types.UserProfilePhotos

async get_file(*file_id: String*) → *File*

Use this method to get basic info about a file and prepare it for downloading. For the moment, bots can download files of up to 20MB in size.

Note: This function may not preserve the original file name and MIME type. You should save the file's MIME type and name (if available) when the File object is received.

Source: https://core.telegram.org/bots/api#getfile

Parameters
file_id (base.String) – File identifier to get info about

Returns
On success, a File object is returned

Return type
types.File

async ban_chat_member(*chat_id: Union[Integer, String]*, *user_id: Integer*, *until_date: Optional[Union[Integer, datetime, timedelta]] = None*, *revoke_messages: Optional[Boolean] = None*) → Boolean

Use this method to ban a user in a group, a supergroup or a channel. In the case of supergroups and channels, the user will not be able to return to the chat on their own using invite links, etc., unless unbanned first. The bot must be an administrator in the chat for this to work and must have the appropriate admin rights. Returns True on success.

Source: https://core.telegram.org/bots/api#banchatmember

Parameters

- **chat_id** (typing.Union[base.Integer, base.String]) – Unique identifier for the target group or username of the target supergroup or channel (in the format @channelusername)
- **user_id** (base.Integer) – Unique identifier of the target user
- **until_date** (typing.Union[base.Integer, datetime.datetime, datetime.timedelta, None]) – Date when the user will be unbanned, unix time. If user is banned for more than 366 days or less than 30 seconds from the current time they are considered to be banned forever. Applied for supergroups and channels only.
- **revoke_messages** (typing.Optional[base.Boolean]) – Pass True to delete all messages from the chat for the user that is being removed. If False, the user will be able to see messages in the group that were sent before the user was removed. Always True for supergroups and channels.

Returns
Returns True on success

Return type
`base.Boolean`

async kick_chat_member(*chat_id: Union[Integer, String]*, *user_id: Integer*, *until_date: Optional[Union[Integer, datetime, timedelta]] = None*, *revoke_messages: Optional[Boolean] = None*) → Boolean

Renamed to ban_chat_member.

async unban_chat_member(*chat_id: Union[Integer, String]*, *user_id: Integer*, *only_if_banned: Optional[Boolean] = None*) → Boolean

Use this method to unban a previously kicked user in a supergroup or channel. The user will not return to the group or channel automatically, but will be able to join via link, etc. The bot must be an administrator for this to work. By default, this method guarantees that after the call the user is not a member of the chat, but will be able to join it. So if the user is a member of the chat they will also be removed from the chat. If you don't want this, use the parameter only_if_banned. Returns True on success.

Source: https://core.telegram.org/bots/api#unbanchatmember

Parameters

- **chat_id** (`typing.Union[base.Integer, base.String]`) – Unique identifier for the target group or username of the target supergroup or channel (in the format @username)
- **user_id** (`base.Integer`) – Unique identifier of the target user
- **only_if_banned** (`typing.Optional[base.Boolean]`) – Do nothing if the user is not banned

Returns
Returns True on success

Return type
`base.Boolean`

async restrict_chat_member(*chat_id: Union[Integer, String]*, *user_id: Integer*, *permissions: Optional[ChatPermissions]*, *use_independent_chat_permissions: Optional[Boolean] = None*, *until_date: Optional[Union[Integer, datetime, timedelta]] = None*, *can_send_messages: Optional[Boolean] = None*, *can_send_media_messages: Optional[Boolean] = None*, *can_send_other_messages: Optional[Boolean] = None*, *can_add_web_page_previews: Optional[Boolean] = None*) → Boolean

Use this method to restrict a user in a supergroup. The bot must be an administrator in the supergroup for this to work and must have the appropriate admin rights. Pass True for all boolean parameters to lift restrictions from a user.

Source: https://core.telegram.org/bots/api#restrictchatmember

Parameters

- **chat_id** (`typing.Union[base.Integer, base.String]`) – Unique identifier for the target chat or username of the target supergroup
- **user_id** (`base.Integer`) – Unique identifier of the target user
- **permissions** (`ChatPermissions`) – New user permissions
- **use_independent_chat_permissions** (`typing.Optional[base.Boolean]`) – Pass True if chat permissions are set independently. Otherwise, the can_send_other_messages and can_add_web_page_previews permissions will imply the can_send_messages, can_send_audios, can_send_documents, can_send_photos, can_send_videos,

can_send_video_notes, and can_send_voice_notes permissions; the can_send_polls permission will imply the can_send_messages permission.

- **until_date** (typing.Optional[base.Integer]) – Date when restrictions will be lifted for the user, unix time
- **can_send_messages** (typing.Optional[base.Boolean]) – Pass True, if the user can send text messages, contacts, locations and venues
- **can_send_media_messages** (typing.Optional[base.Boolean]) – Pass True, if the user can send audios, documents, photos, videos, video notes and voice notes, implies can_send_messages
- **can_send_other_messages** (typing.Optional[base.Boolean]) – Pass True, if the user can send animations, games, stickers and use inline bots, implies can_send_media_messages
- **can_add_web_page_previews** (typing.Optional[base.Boolean]) – Pass True, if the user may add web page previews to their messages, implies can_send_media_messages

Returns
Returns True on success

Return type
base.Boolean

async promote_chat_member(*chat_id: Union[Integer, String], user_id: Integer, is_anonymous: Optional[Boolean] = None, can_manage_chat: Optional[Boolean] = None, can_change_info: Optional[Boolean] = None, can_post_messages: Optional[Boolean] = None, can_edit_messages: Optional[Boolean] = None, can_delete_messages: Optional[Boolean] = None, can_manage_voice_chats: Optional[Boolean] = None, can_invite_users: Optional[Boolean] = None, can_restrict_members: Optional[Boolean] = None, can_pin_messages: Optional[Boolean] = None, can_promote_members: Optional[Boolean] = None, can_manage_video_chats: Optional[Boolean] = None, can_manage_topics: Optional[Boolean] = None*) → Boolean

Use this method to promote or demote a user in a supergroup or a channel. The bot must be an administrator in the chat for this to work and must have the appropriate admin rights. Pass False for all boolean parameters to demote a user.

Source: https://core.telegram.org/bots/api#promotechatmember

Parameters

- **chat_id** (typing.Union[base.Integer, base.String]) – Unique identifier for the target chat or username of the target channel
- **user_id** (base.Integer) – Unique identifier of the target user
- **is_anonymous** (typing.Optional[base.Boolean]) – Pass True, if the administrator's presence in the chat is hidden
- **can_manage_chat** (typing.Optional[base.Boolean]) – Pass True, if the administrator can access the chat event log, chat statistics, message statistics in channels, see channel members, see anonymous administrators in supergroups and ignore slow mode. Implied by any other administrator privilege
- **can_change_info** (typing.Optional[base.Boolean]) – Pass True, if the administrator can change chat title, photo and other settings

- **can_post_messages** (typing.Optional[base.Boolean]) – Pass True, if the administrator can create channel posts, channels only
- **can_edit_messages** (typing.Optional[base.Boolean]) – Pass True, if the administrator can edit messages of other users, channels only
- **can_delete_messages** (typing.Optional[base.Boolean]) – Pass True, if the administrator can delete messages of other users
- **can_manage_voice_chats** (typing.Optional[base.Boolean]) – Pass True, if the administrator can manage voice chats, supergroups only
- **can_invite_users** (typing.Optional[base.Boolean]) – Pass True, if the administrator can invite new users to the chat
- **can_restrict_members** (typing.Optional[base.Boolean]) – Pass True, if the administrator can restrict, ban or unban chat members
- **can_pin_messages** (typing.Optional[base.Boolean]) – Pass True, if the administrator can pin messages, supergroups only
- **can_promote_members** (typing.Optional[base.Boolean]) – Pass True, if the administrator can add new administrators with a subset of his own privileges or demote administrators that he has promoted, directly or indirectly (promoted by administrators that were appointed by him)
- **can_manage_video_chats** – Pass True, if the administrator can manage video chats
- **can_manage_topics** (typing.Optional[base.Boolean]) – Pass True if the user is allowed to create, rename, close, and reopen forum topics, supergroups only

Returns
Returns True on success

Return type
base.Boolean

async set_chat_administrator_custom_title(*chat_id: Union[Integer, String]*, *user_id: Integer*, *custom_title: String*) → Boolean

Use this method to set a custom title for an administrator in a supergroup promoted by the bot.

Returns True on success.

Source: https://core.telegram.org/bots/api#setchatadministratorcustomtitle

Parameters

- **chat_id** – Unique identifier for the target chat or username of the target supergroup
- **user_id** – Unique identifier of the target user
- **custom_title** – New custom title for the administrator; 0-16 characters, emoji are not allowed

Returns
True on success.

async ban_chat_sender_chat(*chat_id: Union[Integer, String]*, *sender_chat_id: Integer*)

Ban a channel chat in a supergroup or a channel.

Until the chat is unbanned, the owner of the banned chat won't be able to send messages on behalf of any of their channels. The bot must be an administrator in the supergroup or channel for this to work and must have the appropriate administrator rights. Returns True on success.

Source: https://core.telegram.org/bots/api#banchatsenderchat

Parameters

- **chat_id** – Unique identifier for the target chat or username of the target channel (in the format @channelusername)
- **sender_chat_id** – Unique identifier of the target sender chat

async unban_chat_sender_chat(*chat_id: Union[Integer, String]*, *sender_chat_id: Integer*)

Unban a previously banned channel chat in a supergroup or channel.

The bot must be an administrator for this to work and must have the appropriate administrator rights. Returns True on success.

Source: https://core.telegram.org/bots/api#unbanchatsenderchat

Parameters

- **chat_id** – Unique identifier for the target chat or username of the target channel (in the format @channelusername)
- **sender_chat_id** – Unique identifier of the target sender chat

async set_chat_permissions(*chat_id: Union[Integer, String]*, *permissions: ChatPermissions*, *use_independent_chat_permissions: Optional[Boolean] = None*) → Boolean

Use this method to set default chat permissions for all members. The bot must be an administrator in the group or a supergroup for this to work and must have the can_restrict_members admin rights.

Returns True on success.

Parameters

- **chat_id** – Unique identifier for the target chat or username of the target supergroup
- **permissions** – New default chat permissions
- **use_independent_chat_permissions** (`typing.Optional[base.Boolean]`) – Pass True if chat permissions are set independently. Otherwise, the can_send_other_messages and can_add_web_page_previews permissions will imply the can_send_messages, can_send_audios, can_send_documents, can_send_photos, can_send_videos, can_send_video_notes, and can_send_voice_notes permissions; the can_send_polls permission will imply the can_send_messages permission.

Returns
True on success.

async export_chat_invite_link(*chat_id: Union[Integer, String]*) → String

Use this method to generate a new invite link for a chat; any previously generated link is revoked. The bot must be an administrator in the chat for this to work and must have the appropriate admin rights.

Source: https://core.telegram.org/bots/api#exportchatinvitelink

Parameters
chat_id (`typing.Union[base.Integer, base.String]`) – Unique identifier for the target chat or username of the target channel

Returns
Returns exported invite link as String on success

Return type
`base.String`

async create_chat_invite_link(*chat_id: Union[Integer, String], expire_date: Optional[Union[Integer, datetime, timedelta]] = None, member_limit: Optional[Integer] = None, name: Optional[String] = None, creates_join_request: Optional[Boolean] = None*) → ChatInviteLink

Use this method to create an additional invite link for a chat. The bot must be an administrator in the chat for this to work and must have the appropriate admin rights. The link can be revoked using the method revokeChatInviteLink.

Source: https://core.telegram.org/bots/api#createchatinvitelink

Parameters

- **chat_id** (typing.Union[base.Integer, base.String]) – Unique identifier for the target chat or username of the target channel (in the format @channelusername)
- **expire_date** (typing.Union[base.Integer, datetime.datetime, datetime.timedelta, None]) – Point in time when the link will expire
- **member_limit** (typing.Optional[base.Integer]) – Maximum number of users that can be members of the chat simultaneously after joining the chat via this invite link; 1-99999
- **name** (typing.Optional[base.String]) – Invite link name; 0-32 characters
- **creates_join_request** (typing.Optional[base.Boolean]) – True, if users joining the chat via the link need to be approved by chat administrators. If True, member_limit can't be specified

Returns
the new invite link as ChatInviteLink object.

Return type
types.ChatInviteLink

async edit_chat_invite_link(*chat_id: Union[Integer, String], invite_link: String, expire_date: Optional[Union[Integer, datetime, timedelta]] = None, member_limit: Optional[Integer] = None, name: Optional[String] = None, creates_join_request: Optional[Boolean] = None*) → ChatInviteLink

Use this method to edit a non-primary invite link created by the bot. The bot must be an administrator in the chat for this to work and must have the appropriate admin rights.

Source: https://core.telegram.org/bots/api#editchatinvitelink

Parameters

- **chat_id** (typing.Union[base.Integer, base.String]) – Unique identifier for the target chat or username of the target channel (in the format @channelusername)
- **invite_link** (base.String) – The invite link to edit
- **expire_date** (typing.Union[base.Integer, datetime.datetime, datetime.timedelta, None]) – Point in time (Unix timestamp) when the link will expire
- **member_limit** (typing.Optional[base.Integer]) – Maximum number of users that can be members of the chat simultaneously after joining the chat via this invite link; 1-99999
- **name** (typing.Optional[base.String]) – Invite link name; 0-32 characters
- **creates_join_request** (typing.Optional[base.Boolean]) – True, if users joining the chat via the link need to be approved by chat administrators. If True, member_limit can't be specified

Returns
edited invite link as a ChatInviteLink object.

async revoke_chat_invite_link(*chat_id: Union[Integer, String]*, *invite_link: String*) → ChatInviteLink

Use this method to revoke an invite link created by the bot. If the primary link is revoked, a new link is automatically generated. The bot must be an administrator in the chat for this to work and must have the appropriate admin rights.

Source: https://core.telegram.org/bots/api#revokechatinvitelink

Parameters

- **chat_id** – Unique identifier for the target chat or username of the target channel (in the format @channelusername)
- **invite_link** – The invite link to revoke

Returns
the revoked invite link as ChatInviteLink object

async approve_chat_join_request(*chat_id: Union[Integer, String]*, *user_id: Integer*) → Boolean

Use this method to approve a chat join request. The bot must be an administrator in the chat for this to work and must have the can_invite_users administrator right.

Returns True on success.

Source: https://core.telegram.org/bots/api#approvechatjoinrequest

Parameters

- **chat_id** (*Union[base.Integer, base.String]*) – Unique identifier for the target chat or username of the target channel (in the format @channelusername)
- **user_id** (*base.Integer*) – Unique identifier of the target user

Returns

async decline_chat_join_request(*chat_id: Union[Integer, String]*, *user_id: Integer*) → Boolean

Use this method to decline a chat join request. The bot must be an administrator in the chat for this to work and must have the can_invite_users administrator right. Returns True on success.

Returns True on success.

Source: https://core.telegram.org/bots/api#declinechatjoinrequest

Parameters

- **chat_id** (*Union[base.Integer, base.String]*) – Unique identifier for the target chat or username of the target channel (in the format @channelusername)
- **user_id** (*base.Integer*) – Unique identifier of the target user

Returns

async set_chat_photo(*chat_id: Union[Integer, String]*, *photo: InputFile*) → Boolean

Use this method to set a new profile photo for the chat. Photos can't be changed for private chats. The bot must be an administrator in the chat for this to work and must have the appropriate admin rights.

Note: In regular groups (non-supergroups), this method will only work if the 'All Members Are Admins' setting is off in the target group.

Source: https://core.telegram.org/bots/api#setchatphoto

Parameters

- **chat_id** (typing.Union[base.Integer, base.String]) – Unique identifier for the target chat or username of the target channel
- **photo** (base.InputFile) – New chat photo, uploaded using multipart/form-data

Returns
Returns True on success

Return type
base.Boolean

async delete_chat_photo(*chat_id: Union[Integer, String]*) → Boolean

Use this method to delete a chat photo. Photos can't be changed for private chats. The bot must be an administrator in the chat for this to work and must have the appropriate admin rights.

Note: In regular groups (non-supergroups), this method will only work if the 'All Members Are Admins' setting is off in the target group.

Source: https://core.telegram.org/bots/api#deletechatphoto

Parameters
chat_id (typing.Union[base.Integer, base.String]) – Unique identifier for the target chat or username of the target channel

Returns
Returns True on success

Return type
base.Boolean

async set_chat_title(*chat_id: Union[Integer, String]*, *title: String*) → Boolean

Use this method to change the title of a chat. Titles can't be changed for private chats. The bot must be an administrator in the chat for this to work and must have the appropriate admin rights.

Note: In regular groups (non-supergroups), this method will only work if the 'All Members Are Admins' setting is off in the target group.

Source: https://core.telegram.org/bots/api#setchattitle

Parameters

- **chat_id** (typing.Union[base.Integer, base.String]) – Unique identifier for the target chat or username of the target channel
- **title** (base.String) – New chat title, 1-255 characters

Returns
Returns True on success

Return type
base.Boolean

async set_chat_description(*chat_id: Union[Integer, String]*, *description: Optional[String] = None*) → Boolean

Use this method to change the description of a supergroup or a channel. The bot must be an administrator in the chat for this to work and must have the appropriate admin rights.

Source: https://core.telegram.org/bots/api#setchatdescription

Parameters

- **chat_id** (typing.Union[base.Integer, base.String]) – Unique identifier for the target chat or username of the target channel

- **description** (typing.Optional[base.String]) – New chat description, 0-255 characters

Returns
Returns True on success

Return type
base.Boolean

async pin_chat_message(*chat_id: Union[Integer, String]*, *message_id: Integer*, *disable_notification: Optional[Boolean] = None*) → Boolean

Use this method to add a message to the list of pinned messages in a chat. If the chat is not a private chat, the bot must be an administrator in the chat for this to work and must have the 'can_pin_messages' admin right in a supergroup or 'can_edit_messages' admin right in a channel. Returns True on success.

Source: https://core.telegram.org/bots/api#pinchatmessage

Parameters

- **chat_id** (typing.Union[base.Integer, base.String]) – Unique identifier for the target chat or username of the target channel (in the format @channelusername)
- **message_id** (base.Integer) – Identifier of a message to pin
- **disable_notification** (typing.Optional[base.Boolean]) – Pass True, if it is not necessary to send a notification to all group members about the new pinned message

Returns
Returns True on success

Return type
base.Boolean

async unpin_chat_message(*chat_id: Union[Integer, String]*, *message_id: Optional[Integer] = None*) → Boolean

Use this method to remove a message from the list of pinned messages in a chat. If the chat is not a private chat, the bot must be an administrator in the chat for this to work and must have the 'can_pin_messages' admin right in a supergroup or 'can_edit_messages' admin right in a channel. Returns True on success.

Source: https://core.telegram.org/bots/api#unpinchatmessage

Parameters

- **chat_id** (typing.Union[base.Integer, base.String]) – Unique identifier for the target chat or username of the target channel (in the format @channelusername)
- **message_id** (typing.Optional[base.Integer]) – Identifier of a message to unpin. If not specified, the most recent pinned message (by sending date) will be unpinned.

Returns
Returns True on success

Return type
base.Boolean

async unpin_all_chat_messages(*chat_id: Union[Integer, String]*) → Boolean

Use this method to clear the list of pinned messages in a chat. If the chat is not a private chat, the bot must be an administrator in the chat for this to work and must have the 'can_pin_messages' admin right in a supergroup or 'can_edit_messages' admin right in a channel. Returns True on success.

Source: https://core.telegram.org/bots/api#unpinallchatmessages

Parameters
chat_id (typing.Union[base.Integer, base.String]) – Unique identifier for the target chat or username of the target channel (in the format @channelusername)

Returns
Returns True on success

Return type
base.Boolean

async close_general_forum_topic(*chat_id: Union[Integer, String]*) → Boolean

Use this method to close an open 'General' topic in a forum supergroup chat. The bot must be an administrator in the chat for this to work and must have the *can_manage_topics* administrator rights.

Returns True on success.

Parameters
chat_id – Unique identifier for the target chat or username of the target supergroup (in the format @supergroupusername)

Returns
Returns True on success.

async edit_general_forum_topic(*chat_id: Union[Integer, String]*, *name: String*) → Boolean

Use this method to edit the name of the 'General' topic in a forum supergroup chat. The bot must be an administrator in the chat for this to work and must have *can_manage_topics* administrator rights.

Returns True on success.

Parameters

- **chat_id** – Unique identifier for the target chat or username of the target supergroup (in the format @supergroupusername)
- **name** – New topic name, 1-128 characters

Returns
Returns True on success.

async hide_general_forum_topic(*chat_id: Union[Integer, String]*) → Boolean

Use this method to hide the 'General' topic in a forum supergroup chat. The bot must be an administrator in the chat for this to work and must have the *can_manage_topics* administrator rights.

The topic will be automatically closed if it was open. Returns True on success.

Parameters
chat_id – Unique identifier for the target chat or username of the target supergroup (in the format @supergroupusername)

Returns
Returns True on success.

async reopen_general_forum_topic(*chat_id: Union[Integer, String]*) → Boolean

Use this method to reopen a closed 'General' topic in a forum supergroup chat. The bot must be an administrator in the chat for this to work and must have the *can_manage_topics* administrator rights. The topic will be automatically unhidden if it was hidden. Returns True on success.

Parameters
chat_id – Unique identifier for the target chat or username of the target supergroup (in the format @supergroupusername)

Returns
Returns True on success.

async unhide_general_forum_topic(*chat_id: Union[Integer, String]*) → Boolean

Use this method to unhide the 'General' topic in a forum supergroup chat. The bot must be an administrator in the chat for this to work and must have the *can_manage_topics* administrator rights.

Returns `True` on success.

Parameters
chat_id – Unique identifier for the target chat or username of the target supergroup (in the format `@supergroupusername`)

Returns
Returns `True` on success.

async leave_chat(*chat_id: Union[Integer, String]*) → Boolean

Use this method for your bot to leave a group, supergroup or channel.

Source: https://core.telegram.org/bots/api#leavechat

Parameters
chat_id(`typing.Union[base.Integer, base.String]`) – Unique identifier for the target chat or username of the target supergroup or channel

Returns
Returns True on success

Return type
`base.Boolean`

async get_chat(*chat_id: Union[Integer, String]*) → *Chat*

Use this method to get up to date information about the chat (current name of the user for one-on-one conversations, current username of a user, group or channel, etc.).

Source: https://core.telegram.org/bots/api#getchat

Parameters
chat_id(`typing.Union[base.Integer, base.String]`) – Unique identifier for the target chat or username of the target supergroup or channel

Returns
Returns a Chat object on success

Return type
`types.Chat`

async get_chat_administrators(*chat_id: Union[Integer, String]*) → List[Union[ChatMemberOwner, ChatMemberAdministrator]]

Use this method to get a list of administrators in a chat.

Source: https://core.telegram.org/bots/api#getchatadministrators

Parameters
chat_id(`typing.Union[base.Integer, base.String]`) – Unique identifier for the target chat or username of the target supergroup or channel

Returns
On success, returns an Array of ChatMember objects that contains information about all chat administrators except other bots. If the chat is a group or a supergroup and no administrators were appointed, only the creator will be returned.

Return type
`typing.List[types.ChatMember]`

async get_chat_member_count(*chat_id: Union[Integer, String]*) → Integer

Use this method to get the number of members in a chat.

Source: https://core.telegram.org/bots/api#getchatmembercount

Parameters

chat_id (typing.Union[base.Integer, base.String]) – Unique identifier for the target chat or username of the target supergroup or channel

Returns

Returns Int on success

Return type

base.Integer

async get_chat_members_count(*chat_id: Union[Integer, String]*) → Integer

Renamed to get_chat_member_count.

async get_chat_member(*chat_id: Union[Integer, String]*, *user_id: Integer*) → *ChatMember*

Use this method to get information about a member of a chat.

Source: https://core.telegram.org/bots/api#getchatmember

Parameters

- **chat_id** (typing.Union[base.Integer, base.String]) – Unique identifier for the target chat or username of the target supergroup or channel
- **user_id** (base.Integer) – Unique identifier of the target user

Returns

Returns a ChatMember object on success

Return type

types.ChatMember

async set_chat_sticker_set(*chat_id: Union[Integer, String]*, *sticker_set_name: String*) → Boolean

Use this method to set a new group sticker set for a supergroup. The bot must be an administrator in the chat for this to work and must have the appropriate admin rights.

Use the field can_set_sticker_set optionally returned in getChat requests to check if the bot can use this method.

Source: https://core.telegram.org/bots/api#setchatstickerset

Parameters

- **chat_id** (typing.Union[base.Integer, base.String]) – Unique identifier for the target chat or username of the target supergroup
- **sticker_set_name** (base.String) – Name of the sticker set to be set as the group sticker set

Returns

Returns True on success

Return type

base.Boolean

async delete_chat_sticker_set(*chat_id: Union[Integer, String]*) → Boolean

Use this method to delete a group sticker set from a supergroup. The bot must be an administrator in the chat for this to work and must have the appropriate admin rights.

Use the field can_set_sticker_set optionally returned in getChat requests to check if the bot can use this method.

Source: https://core.telegram.org/bots/api#deletechatstickerset

Parameters

chat_id(typing.Union[base.Integer, base.String]) – Unique identifier for the target chat or username of the target supergroup

Returns

Returns True on success

Return type

base.Boolean

async get_forum_topic_icon_stickers() → List[*Sticker*]

Use this method to get custom emoji stickers, which can be used as a forum topic icon by any user. Requires no parameters.

Returns an Array of Sticker objects.

Source: https://core.telegram.org/bots/api#getforumtopiciconstickers

Returns

Returns an Array of Sticker objects.

async create_forum_topic(*chat_id: Union[int, str]*, *name: String*, *icon_color: Optional[Integer] = None*, *icon_custom_emoji_id: Optional[String] = None*) → ForumTopic

Use this method to create a topic in a forum supergroup chat. The bot must be an administrator in the chat for this to work and must have the can_manage_topics administrator rights.

Returns information about the created topic as a ForumTopic object.

Source: https://core.telegram.org/bots/api#createforumtopic

Parameters

- **chat_id** – Unique identifier for the target chat or username of the target supergroup (in the format @supergroupusername)
- **name** – Topic name, 1-128 characters
- **icon_color** – Color of the topic icon in RGB format. Currently, must be one of 0x6FB9F0, 0xFFD67E, 0xCB86DB, 0x8EEE98, 0xFF93B2, or 0xFB6F5F
- **icon_custom_emoji_id** – Unique identifier of the custom emoji shown as the topic icon. Use getForumTopicIconStickers to get all allowed custom emoji identifiers.

Returns

Returns information about the created topic as a ForumTopic object.

async edit_forum_topic(*chat_id: Union[int, str]*, *name: Optional[String] = None*, *message_thread_id: Optional[Integer] = None*, *icon_custom_emoji_id: Optional[String] = None*) → Boolean

Use this method to edit name and icon of a topic in a forum supergroup chat.

The bot must be an administrator in the chat for this to work and must have can_manage_topics administrator rights, unless it is the creator of the topic.

Returns True on success.

Source: https://core.telegram.org/bots/api#editforumtopic

Parameters

- **chat_id** – Unique identifier for the target chat or username of the target supergroup (in the format @supergroupusername)
- **name** – Unique identifier for the target message thread of the forum topic
- **message_thread_id** – New topic name, 1-128 characters
- **icon_custom_emoji_id** – New unique identifier of the custom emoji shown as the topic icon. Use getForumTopicIconStickers to get all allowed custom emoji identifiers

Returns
Returns True on success.

async close_forum_topic(*chat_id: Union[int, str]*, *message_thread_id: Optional[Integer] = None*) → Boolean

Use this method to close an open topic in a forum supergroup chat. The bot must be an administrator in the chat for this to work and must have the can_manage_topics administrator rights, unless it is the creator of the topic.

Returns True on success.

Parameters

- **chat_id** – Unique identifier for the target chat or username of the target supergroup (in the format @supergroupusername)
- **message_thread_id** – Unique identifier for the target message thread of the forum topic

Returns
Returns True on success.

async reopen_forum_topic(*chat_id: Union[int, str]*, *message_thread_id: Optional[Integer] = None*) → Boolean

Use this method to reopen a closed topic in a forum supergroup chat. The bot must be an administrator in the chat for this to work and must have the can_manage_topics administrator rights, unless it is the creator of the topic.

Returns True on success.

Source: https://core.telegram.org/bots/api#reopenforumtopic

Parameters

- **chat_id** – Unique identifier for the target chat or username of the target supergroup (in the format @supergroupusername)
- **message_thread_id** – Unique identifier for the target message thread of the forum topic

Returns
Returns True on success.

async delete_forum_topic(*chat_id: Union[int, str]*, *message_thread_id: Optional[Integer] = None*) → Boolean

Use this method to delete a forum topic along with all its messages in a forum supergroup chat. The bot must be an administrator in the chat for this to work and must have the can_delete_messages administrator rights.

Returns True on success.

Source: https://core.telegram.org/bots/api#deleteforumtopic

Parameters

- **chat_id** – Unique identifier for the target chat or username of the target supergroup (in the format @supergroupusername)
- **message_thread_id** – Unique identifier for the target message thread of the forum topic

Returns
Returns True on success.

async unpin_all_forum_topic_messages(*chat_id: Union[int, str]*, *message_thread_id: Optional[Integer] = None*) → Boolean

Use this method to clear the list of pinned messages in a forum topic. The bot must be an administrator in the chat for this to work and must have the can_pin_messages administrator right in the supergroup.

Returns True on success.

Source: https://core.telegram.org/bots/api#unpinallforumtopicmessages

Parameters

- **chat_id** – Unique identifier for the target chat or username of the target supergroup (in the format @supergroupusername)
- **message_thread_id** – Unique identifier for the target message thread of the forum topic

Returns
Returns True on success.

async answer_callback_query(*callback_query_id: String*, *text: Optional[String] = None*, *show_alert: Optional[Boolean] = None*, *url: Optional[String] = None*, *cache_time: Optional[Integer] = None*) → Boolean

Use this method to send answers to callback queries sent from inline keyboards. The answer will be displayed to the user as a notification at the top of the chat screen or as an alert.

Alternatively, the user can be redirected to the specified Game URL. For this option to work, you must first create a game for your bot via @Botfather and accept the terms. Otherwise, you may use links like t.me/your_bot?start=XXXX that open your bot with a parameter.

Source: https://core.telegram.org/bots/api#answercallbackquery

Parameters

- **callback_query_id** (`base.String`) – Unique identifier for the query to be answered
- **text** (`typing.Optional[base.String]`) – Text of the notification. If not specified, nothing will be shown to the user, 0-1024 characters
- **show_alert** (`typing.Optional[base.Boolean]`) – If true, an alert will be shown by the client instead of a notification at the top of the chat screen. Defaults to false.
- **url** (`typing.Optional[base.String]`) – URL that will be opened by the user's client
- **cache_time** (`typing.Optional[base.Integer]`) – The maximum amount of time in seconds that the result of the callback query may be cached client-side.

Returns
On success, True is returned

Return type
`base.Boolean`

async set_my_commands(*commands: List[BotCommand]*, *scope: Optional[BotCommandScope] = None*, *language_code: Optional[String] = None*) → Boolean

Use this method to change the list of the bot's commands.

Source: https://core.telegram.org/bots/api#setmycommands

Parameters

- **commands** – A JSON-serialized list of bot commands to be set as the list of the bot's commands. At most 100 commands can be specified.
- **scope** – A JSON-serialized object, describing scope of users for which the commands are relevant. Defaults to BotCommandScopeDefault.
- **language_code** – A two-letter ISO 639-1 language code. If empty, commands will be applied to all users from the given scope, for whose language there are no dedicated commands

Returns
Returns True on success.

Return type
`base.Boolean`

async delete_my_commands(*scope: Optional[BotCommandScope] = None*, *language_code: Optional[String] = None*) → Boolean

Use this method to delete the list of the bot's commands for the given scope and user language. After deletion, higher level commands will be shown to affected users.

Source: https://core.telegram.org/bots/api#deletemycommands

Parameters

- **scope** – A JSON-serialized object, describing scope of users for which the commands are relevant. Defaults to BotCommandScopeDefault.
- **language_code** – A two-letter ISO 639-1 language code. If empty, commands will be applied to all users from the given scope, for whose language there are no dedicated commands

Returns
Returns True on success.

Return type
`base.Boolean`

async get_my_commands(*scope: Optional[BotCommandScope] = None*, *language_code: Optional[String] = None*) → List[BotCommand]

Use this method to get the current list of the bot's commands for the given scope and user language. Returns Array of BotCommand on success. If commands aren't set, an empty list is returned.

Source: https://core.telegram.org/bots/api#getmycommands

Parameters

- **scope** – A JSON-serialized object, describing scope of users for which the commands are relevant. Defaults to BotCommandScopeDefault.
- **language_code** – A two-letter ISO 639-1 language code. If empty, commands will be applied to all users from the given scope, for whose language there are no dedicated commands

Returns
Returns Array of BotCommand on success or empty list.

Return type
`typing.List[types.BotCommand]`

async set_chat_menu_button(*chat_id: Optional[Integer] = None*, *menu_button: Optional[MenuButton] = None*) → bool

Use this method to change bot's menu button in a private chat, or the default menu button.

Returns True on success.

Source https://core.telegram.org/bots/api#setchatmenubutton

Parameters

- **chat_id** – Unique identifier for the target private chat. If not specified, default bot's menu button will be changed
- **menu_button** – A JSON-serialized object for the new bot's menu button. Defaults to MenuButtonDefault

Returns
Returns True on success.

async get_chat_menu_button(*chat_id: Optional[Integer] = None*) → Union[types.MenuButtonCommands, types.MenuButtonDefault, types.MenuButtonWebApp]

Use this method to get the current value of the bot's menu button in a private chat, or the default menu button.

Returns MenuButton on success.

Source https://core.telegram.org/bots/api#getchatmenu

Parameters
chat_id – Unique identifier for the target private chat. If not specified, default bot's menu button will be returned

Returns
Returns MenuButton on success.

async set_my_default_administrator_rights(*rights: Optional[ChatAdministratorRights] = None*, *for_channels: Optional[Boolean] = None*) → Boolean

Use this method to change default administrator rights of the bot for adding it as an administrator to groups or channels. Returns True on success.

Source: https://core.telegram.org/bots/api#setmydefaultadministratorrights

Parameters

- **rights** – A JSON-serialized object, describing new default administrator rights. If not specified, the default administrator rights will be cleared.
- **for_channels** – Pass True to change default administrator rights of the bot in channels. Otherwise, default administrator rights of the bot for groups and supergroups will be changed.

Returns
Returns True on success.

async get_my_default_administrator_rights(*for_channels: Optional[Boolean] = None*) → ChatAdministratorRights

Use this method to get the current default administrator rights of the bot. Returns ChatAdministratorRights on success.

Source: https://core.telegram.org/bots/api#getmydefaultadministratorrights

Parameters

for_channels – Pass True to get default administrator rights of the bot in channels. Otherwise, default administrator rights of the bot for groups and supergroups will be returned.

Returns

async edit_message_text(*text: String, chat_id: Optional[Union[Integer, String]] = None, message_id: Optional[Integer] = None, inline_message_id: Optional[String] = None, parse_mode: Optional[String] = None, entities: Optional[List[*MessageEntity*]] = None, disable_web_page_preview: Optional[Boolean] = None, reply_markup: Optional[*InlineKeyboardMarkup*] = None*) → *Message*

Use this method to edit text and game messages sent by the bot or via the bot (for inline bots).

Source: https://core.telegram.org/bots/api#editmessagetext

Parameters

- **chat_id** (typing.Union[base.Integer, base.String, None]) – Required if inline_message_id is not specified Unique identifier for the target chat or username of the target channel
- **message_id** (typing.Optional[base.Integer]) – Required if inline_message_id is not specified. Identifier of the sent message
- **inline_message_id** (typing.Optional[base.String]) – Required if chat_id and message_id are not specified. Identifier of the inline message
- **text** (base.String) – New text of the message
- **parse_mode** (typing.Optional[base.String]) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.
- **entities** (typing.Optional[typing.List[types.MessageEntity]]) – List of special entities that appear in message text, which can be specified instead of parse_mode
- **disable_web_page_preview** (typing.Optional[base.Boolean]) – Disables link previews for links in this message
- **reply_markup** (typing.Optional[types.InlineKeyboardMarkup]) – A JSON-serialized object for an inline keyboard

Returns

On success, if edited message is sent by the bot, the edited Message is returned, otherwise True is returned.

Return type

typing.Union[types.Message, base.Boolean]

async edit_message_caption(*chat_id: Optional[Union[Integer, String]] = None, message_id: Optional[Integer] = None, inline_message_id: Optional[String] = None, caption: Optional[String] = None, parse_mode: Optional[String] = None, caption_entities: Optional[List[*MessageEntity*]] = None, reply_markup: Optional[*InlineKeyboardMarkup*] = None*) → *Message*

Use this method to edit captions of messages sent by the bot or via the bot (for inline bots).

Source: https://core.telegram.org/bots/api#editmessagecaption

Parameters

- **chat_id** (typing.Union[base.Integer, base.String, None]) – Required if inline_message_id is not specified Unique identifier for the target chat or username of the target channel
- **message_id** (typing.Optional[base.Integer]) – Required if inline_message_id is not specified. Identifier of the sent message
- **inline_message_id** (typing.Optional[base.String]) – Required if chat_id and message_id are not specified. Identifier of the inline message
- **caption** (typing.Optional[base.String]) – New caption of the message
- **parse_mode** (typing.Optional[base.String]) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.
- **caption_entities** (typing.Optional[typing.List[types.MessageEntity]]) – List of special entities that appear in message text, which can be specified instead of parse_mode
- **reply_markup** (typing.Optional[types.InlineKeyboardMarkup]) – A JSON-serialized object for an inline keyboard

Returns

On success, if edited message is sent by the bot, the edited Message is returned, otherwise True is returned.

Return type

typing.Union[types.Message, base.Boolean]

async edit_message_media(*media: InputMedia, chat_id: Optional[Union[Integer, String]] = None, message_id: Optional[Integer] = None, inline_message_id: Optional[String] = None, reply_markup: Optional[InlineKeyboardMarkup] = None*) → Union[*Message*, Boolean]

Use this method to edit audio, document, photo, or video messages. If a message is a part of a message album, then it can be edited only to a photo or a video. Otherwise, message type can be changed arbitrarily. When inline message is edited, new file can't be uploaded. Use previously uploaded file via its file_id or specify a URL.

On success, if the edited message was sent by the bot, the edited Message is returned, otherwise True is returned.

Source https://core.telegram.org/bots/api#editmessagemedia

Parameters

- **chat_id** (typing.Union[typing.Union[base.Integer, base.String], None]) – Required if inline_message_id is not specified
- **message_id** (typing.Optional[base.Integer]) – Required if inline_message_id is not specified. Identifier of the sent message
- **inline_message_id** (typing.Optional[base.String]) – Required if chat_id and message_id are not specified. Identifier of the inline message
- **media** (types.InputMedia) – A JSON-serialized object for a new media content of the message
- **reply_markup** (typing.Optional[types.InlineKeyboardMarkup]) – A JSON-serialized object for a new inline keyboard

Returns
On success, if the edited message was sent by the bot, the edited Message is returned, otherwise True is returned

Return type
`typing.Union[types.Message, base.Boolean]`

async edit_message_reply_markup(*chat_id: Optional[Union[Integer, String]] = None*, *message_id: Optional[Integer] = None*, *inline_message_id: Optional[String] = None*, *reply_markup: Optional[InlineKeyboardMarkup] = None*) → *Message*

Use this method to edit only the reply markup of messages sent by the bot or via the bot (for inline bots).

Source: https://core.telegram.org/bots/api#editmessagereplymarkup

Parameters

- **chat_id** (`typing.Union[base.Integer, base.String, None]`) – Required if inline_message_id is not specified Unique identifier for the target chat or username of the target channel
- **message_id** (`typing.Optional[base.Integer]`) – Required if inline_message_id is not specified. Identifier of the sent message
- **inline_message_id** (`typing.Optional[base.String]`) – Required if chat_id and message_id are not specified. Identifier of the inline message
- **reply_markup** (`typing.Optional[types.InlineKeyboardMarkup]`) – A JSON-serialized object for an inline keyboard

Returns
On success, if edited message is sent by the bot, the edited Message is returned, otherwise True is returned.

Return type
`typing.Union[types.Message, base.Boolean]`

async stop_poll(*chat_id: Union[String, Integer]*, *message_id: Integer*, *reply_markup: Optional[InlineKeyboardMarkup] = None*) → Poll

Use this method to stop a poll which was sent by the bot. On success, the stopped Poll with the final results is returned.

Parameters

- **chat_id** (`typing.Union[base.String, base.Integer]`) – Unique identifier for the target chat or username of the target channel
- **message_id** (`base.Integer`) – Identifier of the original message with the poll
- **reply_markup** (`typing.Optional[types.InlineKeyboardMarkup]`) – A JSON-serialized object for a new message inline keyboard.

Returns
On success, the stopped Poll with the final results is returned.

Return type
`types.Poll`

async delete_message(*chat_id: Union[Integer, String]*, *message_id: Integer*) → Boolean

Use this method to delete a message, including service messages, with the following limitations: - A message can only be deleted if it was sent less than 48 hours ago. - Bots can delete outgoing messages in private chats, groups, and supergroups. - Bots can delete incoming messages in private chats. - Bots

granted can_post_messages permissions can delete outgoing messages in channels. - If the bot is an administrator of a group, it can delete any message there. - If the bot has can_delete_messages permission in a supergroup or a channel, it can delete any message there.

Source: https://core.telegram.org/bots/api#deletemessage

Parameters

- **chat_id** (typing.Union[base.Integer, base.String]) – Unique identifier for the target chat or username of the target channel
- **message_id** (base.Integer) – Identifier of the message to delete

Returns
Returns True on success

Return type
base.Boolean

async send_sticker(*chat_id: Union[Integer, String], sticker: Union[InputFile, String], message_thread_id: Optional[Integer] = None, disable_notification: Optional[Boolean] = None, protect_content: Optional[Boolean] = None, reply_to_message_id: Optional[Integer] = None, allow_sending_without_reply: Optional[Boolean] = None, reply_markup: Optional[Union[*InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply*]] = None*) → *Message*

Use this method to send .webp stickers.

Source: https://core.telegram.org/bots/api#sendsticker

Parameters

- **chat_id** (typing.Union[base.Integer, base.String]) – Unique identifier for the target chat or username of the target channel
- **message_thread_id** (typing.Optional[base.Integer]) – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **sticker** (typing.Union[base.InputFile, base.String]) – Sticker to send
- **disable_notification** (typing.Optional[base.Boolean]) – Sends the message silently. Users will receive a notification with no sound
- **protect_content** (typing.Optional[base.Boolean]) – Protects the contents of sent messages from forwarding and saving
- **reply_to_message_id** (typing.Optional[base.Integer]) – If the message is a reply, ID of the original message
- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user

Returns
On success, the sent Message is returned

Return type
types.Message

async get_sticker_set(*name: String*) → *StickerSet*

Use this method to get a sticker set.

Source: https://core.telegram.org/bots/api#getstickerset

Parameters

name (base.String) – Name of the sticker set

Returns

On success, a StickerSet object is returned

Return type

types.StickerSet

async upload_sticker_file(*user_id: Integer*, *png_sticker: InputFile*) → *File*

Use this method to upload a .png file with a sticker for later use in createNewStickerSet and addStickerToSet methods (can be used multiple times).

Source: https://core.telegram.org/bots/api#uploadstickerfile

Parameters

- **user_id** (base.Integer) – User identifier of sticker file owner
- **png_sticker** (base.InputFile) – Png image with the sticker, must be up to 512 kilobytes in size, dimensions must not exceed 512px, and either width or height must be exactly 512px.

Returns

Returns the uploaded File on success

Return type

types.File

async get_custom_emoji_stickers(*custom_emoji_ids: List[String]*) → List[*Sticker*]

Use this method to get information about custom emoji stickers by their identifiers.

Source: https://core.telegram.org/bots/api#getcustomemojistickers

Parameters

custom_emoji_ids (typing.List[base.String]) – List of custom emoji identifiers. At most 200 custom emoji identifiers can be specified.

Returns

Returns an Array of Sticker objects.

Return type

typing.List[types.Sticker]

async create_new_sticker_set(*user_id: Integer*, *name: String*, *title: String*, *emojis: String*, *png_sticker: Optional[Union[InputFile, String]] = None*, *tgs_sticker: Optional[InputFile] = None*, *webm_sticker: Optional[InputFile] = None*, *contains_masks: Optional[Boolean] = None*, *sticker_type: Optional[String] = None*, *mask_position: Optional[*MaskPosition*] = None*) → Boolean

Use this method to create a new sticker set owned by a user. The bot will be able to edit the sticker set thus created. You must use exactly one of the fields png_sticker or tgs_sticker.

Source: https://core.telegram.org/bots/api#createnewstickerset

Parameters

- **user_id** (base.Integer) – User identifier of created sticker set owner

- **name** (`base.String`) – Short name of sticker set, to be used in t.me/addstickers/ URLs (e.g., animals). Can contain only english letters, digits and underscores. Must begin with a letter, can't contain consecutive underscores and must end in "_by_<bot username>". <bot_username> is case insensitive. 1-64 characters.
- **title** (`base.String`) – Sticker set title, 1-64 characters
- **png_sticker** (`typing.Union[base.InputFile, base.String]`) – PNG image with the sticker, must be up to 512 kilobytes in size, dimensions must not exceed 512px, and either width or height must be exactly 512px. Pass a file_id as a String to send a file that already exists on the Telegram servers, pass an HTTP URL as a String for Telegram to get a file from the Internet, or upload a new one using multipart/form-data. More info on https://core.telegram.org/bots/api#sending-files
- **tgs_sticker** (`base.InputFile`) – TGS animation with the sticker, uploaded using multipart/form-data. See https://core.telegram.org/animated_stickers#technical-requirements for technical requirements
- **webm_sticker** (`base.String`) – WEBM video with the sticker, uploaded using multipart/form-data. See https://core.telegram.org/stickers#video-sticker-requirements for technical requirements
- **sticker_type** (`base.InputFile`) – Type of stickers in the set, pass "regular" or "mask". Custom emoji sticker sets can't be created via the Bot API at the moment. By default, a regular sticker set is created.
- **emojis** (`base.String`) – One or more emoji corresponding to the sticker
- **contains_masks** (`typing.Optional[base.Boolean]`) – Pass True, if a set of mask stickers should be created
- **mask_position** (`typing.Optional[types.MaskPosition]`) – A JSON-serialized object for position where the mask should be placed on faces

Returns
Returns True on success

Return type
`base.Boolean`

async add_sticker_to_set(*user_id: Integer*, *name: String*, *emojis: String*, *png_sticker: Optional[Union[InputFile, String]] = None*, *tgs_sticker: Optional[InputFile] = None*, *webm_sticker: Optional[InputFile] = None*, *mask_position: Optional[MaskPosition] = None*) → Boolean

Use this method to add a new sticker to a set created by the bot. You must use exactly one of the fields png_sticker or tgs_sticker. Animated stickers can be added to animated sticker sets and only to them. Animated sticker sets can have up to 50 stickers. Static sticker sets can have up to 120 stickers.

Source: https://core.telegram.org/bots/api#addstickertoset

Parameters

- **user_id** (`base.Integer`) – User identifier of sticker set owner
- **name** (`base.String`) – Sticker set name
- **png_sticker** (`typing.Union[base.InputFile, base.String]`) – PNG image with the sticker, must be up to 512 kilobytes in size, dimensions must not exceed 512px, and either width or height must be exactly 512px. Pass a file_id as a String to send a file that already exists on the Telegram servers, pass an HTTP URL as a String for Telegram to get a file from the Internet, or upload a new one using multipart/form-data. More info on https://core.telegram.org/bots/api#sending-files

- **tgs_sticker** (base.InputFile) – TGS animation with the sticker, uploaded using multipart/form-data. See https://core.telegram.org/animated_stickers#technical-requirements for technical requirements
- **webm_sticker** (base.InputFile) – WEBM video with the sticker, uploaded using multipart/form-data. See https://core.telegram.org/stickers#video-sticker-requirements for technical requirements
- **emojis** (base.String) – One or more emoji corresponding to the sticker
- **mask_position** (typing.Optional[types.MaskPosition]) – A JSON-serialized object for position where the mask should be placed on faces

Returns
Returns True on success

Return type
base.Boolean

async set_sticker_position_in_set(*sticker: String*, *position: Integer*) → Boolean

Use this method to move a sticker in a set created by the bot to a specific position.

Source: https://core.telegram.org/bots/api#setstickerpositioninset

Parameters

- **sticker** (base.String) – File identifier of the sticker
- **position** (base.Integer) – New sticker position in the set, zero-based

Returns
Returns True on success

Return type
base.Boolean

async delete_sticker_from_set(*sticker: String*) → Boolean

Use this method to delete a sticker from a set created by the bot.

Source: https://core.telegram.org/bots/api#deletestickerfromset

Parameters
sticker (base.String) – File identifier of the sticker

Returns
Returns True on success

Return type
base.Boolean

async set_sticker_set_thumb(*name: String*, *user_id: Integer*, *thumb: Optional[Union[InputFile, String]] = None*) → Boolean

Use this method to set the thumbnail of a sticker set. Animated thumbnails can be set for animated sticker sets only.

Source: https://core.telegram.org/bots/api#setstickersetthumb

Parameters

- **name** (base.String) – Sticker set name
- **user_id** (base.Integer) – User identifier of the sticker set owner

- **thumb** (typing.Union[base.InputFile, base.String]) – A PNG image with the thumbnail, must be up to 128 kilobytes in size and have width and height exactly 100px, or a TGS animation with the thumbnail up to 32 kilobytes in size; see https://core.telegram.org/stickers#animated-sticker-requirements for animated sticker technical requirements, or a WEBM video with the thumbnail up to 32 kilobytes in size; see https://core.telegram.org/stickers#video-sticker-requirements for video sticker technical requirements. Pass a file_id as a String to send a file that already exists on the Telegram servers, pass an HTTP URL as a String for Telegram to get a file from the Internet, or upload a new one using multipart/form-data. More info on https://core.telegram.org/bots/api#sending-files. Animated sticker set thumbnail can't be uploaded via HTTP URL.

Returns
Returns True on success

Return type
base.Boolean

async answer_inline_query(*inline_query_id: String*, *results: List[InlineQueryResult]*, *cache_time: Optional[Integer] = None*, *is_personal: Optional[Boolean] = None*, *next_offset: Optional[String] = None*, *switch_pm_text: Optional[String] = None*, *switch_pm_parameter: Optional[String] = None*) → Boolean

Use this method to send answers to an inline query. No more than 50 results per query are allowed.

Source: https://core.telegram.org/bots/api#answerinlinequery

Parameters

- **inline_query_id** (base.String) – Unique identifier for the answered query
- **results** (typing.List[types.InlineQueryResult]) – A JSON-serialized array of results for the inline query
- **cache_time** (typing.Optional[base.Integer]) – The maximum amount of time in seconds that the result of the inline query may be cached on the server. Defaults to 300.
- **is_personal** (typing.Optional[base.Boolean]) – Pass True, if results may be cached on the server side only for the user that sent the query. By default, results may be returned to any user who sends the same query
- **next_offset** (typing.Optional[base.String]) – Pass the offset that a client should send in the next query with the same text to receive more results. Pass an empty string if there are no more results or if you don't support pagination. Offset length can't exceed 64 bytes.
- **switch_pm_text** (typing.Optional[base.String]) – If passed, clients will display a button with specified text that switches the user to a private chat with the bot and sends the bot a start message with the parameter switch_pm_parameter
- **switch_pm_parameter** (typing.Optional[base.String]) – Deep-linking parameter for the /start message sent to the bot when user presses the switch button. 1-64 characters, only A-Z, a-z, 0-9, _ and - are allowed.

Returns
On success, True is returned

Return type
base.Boolean

async answer_web_app_query(*web_app_query_id: String*, *result: InlineQueryResult*) → SentWebAppMessage

Use this method to set result of interaction with web app and send corresponding message on behalf of the user to the chat from which the query originated. On success, SentWebAppMessage is returned.

Source https://core.telegram.org/bots/api#answerwebappquery

Parameters

- **web_app_query_id** – Unique identifier for the answered query
- **result** – A JSON-serialized object with a description of the message to send

Returns

On success, SentWebAppMessage is returned.

async send_invoice(*chat_id: Union[Integer, String], title: String, description: String, payload: String, provider_token: String, currency: String, prices: List[LabeledPrice], max_tip_amount: Optional[Integer] = None, suggested_tip_amounts: Optional[List[Integer]] = None, start_parameter: Optional[String] = None, provider_data: Optional[Dict] = None, photo_url: Optional[String] = None, photo_size: Optional[Integer] = None, photo_width: Optional[Integer] = None, photo_height: Optional[Integer] = None, need_name: Optional[Boolean] = None, need_phone_number: Optional[Boolean] = None, need_email: Optional[Boolean] = None, need_shipping_address: Optional[Boolean] = None, send_phone_number_to_provider: Optional[Boolean] = None, send_email_to_provider: Optional[Boolean] = None, is_flexible: Optional[Boolean] = None, message_thread_id: Optional[Integer] = None, disable_notification: Optional[Boolean] = None, protect_content: Optional[Boolean] = None, reply_to_message_id: Optional[Integer] = None, allow_sending_without_reply: Optional[Boolean] = None, reply_markup: Optional[InlineKeyboardMarkup] = None*) → *Message*

Use this method to send invoices.

Source: https://core.telegram.org/bots/api#sendinvoice

Parameters

- **chat_id** (typing.Union[base.Integer, base.String]) – Unique identifier for the target chat or username of the target channel (in the format @channelusername)
- **message_thread_id** (typing.Optional[base.Integer]) – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **title** (base.String) – Product name, 1-32 characters
- **description** (base.String) – Product description, 1-255 characters
- **payload** (base.String) – Bot-defined invoice payload, 1-128 bytes This will not be displayed to the user, use for your internal processes.
- **provider_token** (base.String) – Payments provider token, obtained via Botfather
- **currency** (base.String) – Three-letter ISO 4217 currency code, see more on currencies
- **prices** (typing.List[types.LabeledPrice]) – Price breakdown, a list of components (e.g. product price, tax, discount, delivery cost, delivery tax, bonus, etc.)
- **max_tip_amount** (typing.Optional[base.Integer]) – The maximum accepted amount for tips in the smallest units of the currency (integer, not float/double). For example, for a maximum tip of US$ 1.45 pass max_tip_amount = 145. See the exp parameter in currencies.json, it shows the number of digits past the decimal point for each currency (2 for the majority of currencies). Defaults to 0

- **suggested_tip_amounts** (typing.Optional[typing.List[base.Integer]]) – A JSON-serialized array of suggested amounts of tips in the smallest units of the currency (integer, not float/double). At most 4 suggested tip amounts can be specified. The suggested tip amounts must be positive, passed in a strictly increased order and must not exceed max_tip_amount.
- **start_parameter** (typing.Optional[base.String]) – Unique deep-linking parameter. If left empty, forwarded copies of the sent message will have a Pay button, allowing multiple users to pay directly from the forwarded message, using the same invoice. If non-empty, forwarded copies of the sent message will have a URL button with a deep link to the bot (instead of a Pay button), with the value used as the start parameter
- **provider_data** (typing.Optional[typing.Dict]) – JSON-encoded data about the invoice, which will be shared with the payment provider
- **photo_url** (typing.Optional[base.String]) – URL of the product photo for the invoice
- **photo_size** (typing.Optional[base.Integer]) – Photo size
- **photo_width** (typing.Optional[base.Integer]) – Photo width
- **photo_height** (typing.Optional[base.Integer]) – Photo height
- **need_name** (typing.Optional[base.Boolean]) – Pass True, if you require the user's full name to complete the order
- **need_phone_number** (typing.Optional[base.Boolean]) – Pass True, if you require the user's phone number to complete the order
- **need_email** (typing.Optional[base.Boolean]) – Pass True, if you require the user's email to complete the order
- **need_shipping_address** (typing.Optional[base.Boolean]) – Pass True, if you require the user's shipping address to complete the order
- **send_phone_number_to_provider** (typing.Optional[base.Boolean]) – Pass True, if user's phone number should be sent to provider
- **send_email_to_provider** (typing.Optional[base.Boolean]) – Pass True, if user's email address should be sent to provider
- **is_flexible** (typing.Optional[base.Boolean]) – Pass True, if the final price depends on the shipping method
- **disable_notification** (typing.Optional[base.Boolean]) – Sends the message silently. Users will receive a notification with no sound
- **protect_content** (typing.Optional[base.Boolean]) – Protects the contents of sent messages from forwarding and saving
- **reply_to_message_id** (typing.Optional[base.Integer]) – If the message is a reply, ID of the original message
- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (typing.Optional[types.InlineKeyboardMarkup]) – A JSON-serialized object for an inline keyboard If empty, one 'Pay total price' button will be shown. If not empty, the first button must be a Pay button.

Returns

On success, the sent Message is returned

Return type
`types.Message`

async create_invoice_link(*title: String, description: String, payload: String, provider_token: String, currency: String, prices: List[LabeledPrice], max_tip_amount: Optional[int] = None, suggested_tip_amounts: Optional[List[int]] = None, provider_data: Optional[String] = None, photo_url: Optional[str] = None, photo_size: Optional[int] = None, photo_width: Optional[int] = None, photo_height: Optional[int] = None, need_name: Optional[bool] = None, need_phone_number: Optional[bool] = None, need_email: Optional[bool] = None, need_shipping_address: Optional[bool] = None, send_phone_number_to_provider: Optional[bool] = None, send_email_to_provider: Optional[bool] = None, is_flexible: Optional[bool] = None*) → str

Use this method to create a link for an invoice. On success, the created link is returned.

Source: https://core.telegram.org/bots/api#createinvoicelink

Parameters

- **title** – Product name, 1-32 characters
- **description** – Product description, 1-255 characters
- **payload** – Bot-defined invoice payload, 1-128 bytes. This will not be displayed to the user, use for your internal processes.
- **provider_token** – Payment provider token, obtained via BotFather
- **currency** – Three-letter ISO 4217 currency code, see more on currencies
- **prices** – Price breakdown, a JSON-serialized list of components (e.g. product price, tax, discount, delivery cost, delivery tax, bonus, etc.)
- **max_tip_amount** – The maximum accepted amount for tips in the smallest units of the currency (integer, not float/double). For example, for a maximum tip of US$ 1.45 pass max_tip_amount = 145. See the exp parameter in currencies.json, it shows the number of digits past the decimal point for each currency (2 for the majority of currencies). Defaults to 0
- **suggested_tip_amounts** – A JSON-serialized array of suggested amounts of tips in the smallest units of the currency (integer, not float/double). At most 4 suggested tip amounts can be specified. The suggested tip amounts must be positive, passed in a strictly increased order and must not exceed max_tip_amount.
- **provider_data** – JSON-serialized data about the invoice, which will be shared with the payment provider. A detailed description of required fields should be provided by the payment provider.
- **photo_url** – URL of the product photo for the invoice. Can be a photo of the goods or a marketing image for a service.
- **photo_size** – Photo size in bytes
- **photo_width** – Photo width
- **photo_height** – Photo height
- **need_name** – Pass True, if you require the user's full name to complete the order
- **need_phone_number** – Pass True, if you require the user's phone number to complete the order

- **need_email** – Pass True, if you require the user's email address to complete the order
- **need_shipping_address** – Pass True, if you require the user's shipping address to complete the order
- **send_phone_number_to_provider** – Pass True, if the user's phone number should be sent to the provider
- **send_email_to_provider** – Pass True, if the user's email address should be sent to the provider
- **is_flexible** – Pass True, if the final price depends on the shipping method

Returns

async answer_shipping_query(*shipping_query_id: String*, *ok: Boolean*, *shipping_options: Optional[List[ShippingOption]] = None*, *error_message: Optional[String] = None*) → Boolean

If you sent an invoice requesting a shipping address and the parameter is_flexible was specified, the Bot API will send an Update with a shipping_query field to the bot.

Source: https://core.telegram.org/bots/api#answershippingquery

Parameters

- **shipping_query_id** (`base.String`) – Unique identifier for the query to be answered
- **ok** (`base.Boolean`) – Specify True if delivery to the specified address is possible and False if there are any problems (for example, if delivery to the specified address is not possible)
- **shipping_options** (`typing.Union[typing.List[types.ShippingOption], None]`) – Required if ok is True. A JSON-serialized array of available shipping options
- **error_message** (`typing.Optional[base.String]`) – Required if ok is False Error message in human readable form that explains why it is impossible to complete the order (e.g. "Sorry, delivery to your desired address is unavailable'). Telegram will display this message to the user.

Returns
On success, True is returned

Return type
`base.Boolean`

async answer_pre_checkout_query(*pre_checkout_query_id: String*, *ok: Boolean*, *error_message: Optional[String] = None*) → Boolean

Once the user has confirmed their payment and shipping details, the Bot API sends the final confirmation in the form of an Update with the field pre_checkout_query. Use this method to respond to such pre-checkout queries.

Source: https://core.telegram.org/bots/api#answerprecheckoutquery

Parameters

- **pre_checkout_query_id** (`base.String`) – Unique identifier for the query to be answered
- **ok** (`base.Boolean`) – Specify True if everything is alright (goods are available, etc.) and the bot is ready to proceed with the order. Use False if there are any problems.
- **error_message** (`typing.Optional[base.String]`) – Required if ok is False Error message in human readable form that explains the reason for failure to proceed with the checkout (e.g. "Sorry, somebody just bought the last of our amazing black T-shirts while

you were busy filling out your payment details. Please choose a different color or garment!"). Telegram will display this message to the user.

Returns
On success, True is returned

Return type
`base.Boolean`

async set_passport_data_errors(*user_id: Integer*, *errors: List[PassportElementError]*) → Boolean

Informs a user that some of the Telegram Passport elements they provided contains errors. The user will not be able to re-submit their Passport to you until the errors are fixed (the contents of the field for which you returned the error must change). Returns True on success.

Use this if the data submitted by the user doesn't satisfy the standards your service requires for any reason. For example, if a birthday date seems invalid, a submitted document is blurry, a scan shows evidence of tampering, etc. Supply some details in the error message to make sure the user knows how to correct the issues.

Source https://core.telegram.org/bots/api#setpassportdataerrors

Parameters

- **user_id** (`base.Integer`) – User identifier
- **errors** (`typing.List[types.PassportElementError]`) – A JSON-serialized array describing the errors

Returns
Returns True on success

Return type
`base.Boolean`

async send_game(*chat_id: Integer*, *game_short_name: String*, *disable_notification: Optional[Boolean] = None*, *protect_content: Optional[Boolean] = None*, *reply_to_message_id: Optional[Integer] = None*, *allow_sending_without_reply: Optional[Boolean] = None*, *reply_markup: Optional[InlineKeyboardMarkup] = None*) → *Message*

Use this method to send a game.

Source: https://core.telegram.org/bots/api#sendgame

Parameters

- **chat_id** (`base.Integer`) – Unique identifier for the target chat
- **game_short_name** (`base.String`) – Short name of the game, serves as the unique identifier for the game. Set up your games via Botfather.
- **disable_notification** (`typing.Optional[base.Boolean]`) – Sends the message silently. Users will receive a notification with no sound
- **protect_content** (`typing.Optional[base.Boolean]`) – Protects the contents of sent messages from forwarding and saving
- **reply_to_message_id** (`typing.Optional[base.Integer]`) – If the message is a reply, ID of the original message
- **allow_sending_without_reply** (`typing.Optional[base.Boolean]`) – Pass True, if the message should be sent even if the specified replied-to message is not found

- **reply_markup** (typing.Optional[types.InlineKeyboardMarkup]) – A JSON-serialized object for an inline keyboard If empty, one 'Play game_title' button will be shown. If not empty, the first button must launch the game.

Returns
On success, the sent Message is returned

Return type
types.Message

async set_game_score(*user_id: Integer, score: Integer, force: Optional[Boolean] = None, disable_edit_message: Optional[Boolean] = None, chat_id: Optional[Integer] = None, message_id: Optional[Integer] = None, inline_message_id: Optional[String] = None*) → *Message*

Use this method to set the score of the specified user in a game.

Source: https://core.telegram.org/bots/api#setgamescore

Parameters

- **user_id** (base.Integer) – User identifier
- **score** (base.Integer) – New score, must be non-negative
- **force** (typing.Optional[base.Boolean]) – Pass True, if the high score is allowed to decrease This can be useful when fixing mistakes or banning cheaters
- **disable_edit_message** (typing.Optional[base.Boolean]) – Pass True, if the game message should not be automatically edited to include the current scoreboard
- **chat_id** (typing.Optional[base.Integer]) – Required if inline_message_id is not specified. Unique identifier for the target chat
- **message_id** (typing.Optional[base.Integer]) – Required if inline_message_id is not specified. Identifier of the sent message
- **inline_message_id** (typing.Optional[base.String]) – Required if chat_id and message_id are not specified. Identifier of the inline message

Returns
On success, if the message was sent by the bot, returns the edited Message, otherwise returns True Returns an error, if the new score is not greater than the user's current score in the chat and force is False.

Return type
typing.Union[types.Message, base.Boolean]

async get_game_high_scores(*user_id: Integer, chat_id: Optional[Integer] = None, message_id: Optional[Integer] = None, inline_message_id: Optional[String] = None*) → List[*GameHighScore*]

Use this method to get data for high score tables.

This method will currently return scores for the target user, plus two of his closest neighbors on each side. Will also return the top three users if the user and his neighbors are not among them. Please note that this behavior is subject to change.

Source: https://core.telegram.org/bots/api#getgamehighscores

Parameters

- **user_id** (base.Integer) – Target user id

- **chat_id** (typing.Optional[base.Integer]) – Required if inline_message_id is not specified. Unique identifier for the target chat
- **message_id** (typing.Optional[base.Integer]) – Required if inline_message_id is not specified. Identifier of the sent message
- **inline_message_id** (typing.Optional[base.String]) – Required if chat_id and message_id are not specified. Identifier of the inline message

Returns
Will return the score of the specified user and several of his neighbors in a game On success, returns an Array of GameHighScore objects. This method will currently return scores for the target user, plus two of his closest neighbors on each side. Will also return the top three users if the user and his neighbors are not among them.

Return type
typing.List[types.GameHighScore]

API Helpers

class aiogram.bot.api.**TelegramAPIServer**(*base: str*, *file: str*)

Bases: object

Base config for API Endpoints

api_url(*token: str*, *method: str*) → str

Generate URL for API methods

Parameters

- **token** – Bot token
- **method** – API method name (case insensitive)

Returns
URL

file_url(*token: str*, *path: str*) → str

Generate URL for downloading files

Parameters

- **token** – Bot token
- **path** – file path

Returns
URL

aiogram.bot.api.**check_token**(*token: str*) → bool

Validate BOT token

Parameters
token –

Returns

aiogram.bot.api.**check_result**(*method_name: str*, *content_type: str*, *status_code: int*, *body: str*)

Checks whether *result* is a valid API response. A result is considered invalid if: - The server returned an HTTP response code other than 200 - The content of the result is invalid JSON. - The method call was unsuccessful (The JSON 'ok' field equals False)

Parameters

- **method_name** – The name of the method called
- **status_code** – status code
- **content_type** – content type of result
- **body** – result body

Returns
The result parsed to a JSON dictionary

Raises
ApiException – if one of the above listed cases is applicable

`aiogram.bot.api.guess_filename(obj)`

Get file name from object

Parameters
obj –

Returns

`aiogram.bot.api.compose_data(params=None, files=None)`

Prepare request data

Parameters

- **params** –
- **files** –

Returns

class `aiogram.bot.api.Methods`

Bases: `Helper`

Helper for Telegram API Methods listed on https://core.telegram.org/bots/api

4.4.2 Telegram data types

Bases

Base TelegramObject

MetaTelegramObject

class `aiogram.types.base.MetaTelegramObject(name: str, bases: Tuple[Type], namespace: Dict[str, Any], **kwargs: Any)`

Bases: `type`

Metaclass for telegram objects

TelegramObject

class aiogram.types.base.**TelegramObject**(*conf: Optional[Dict[str, Any]] = None*, ***kwargs: Any*)

Bases: ContextInstanceMixin

Abstract class for telegram objects

Deserialize object

Parameters

- **conf** –
- **kwargs** –

property props: Dict[str, *BaseField*]

Get props

Returns
dict with props

property props_aliases: Dict[str, str]

Get aliases for props

Returns

property values: Dict[str, Any]

Get values

Returns

classmethod to_object(*data: Dict[str, Any]*, *conf: Optional[Dict[str, Any]] = None*) → T

Deserialize object

Parameters

- **data** –
- **conf** –

Returns

to_python() → Dict[str, Any]

Get object as JSON serializable

Returns

clean() → None

Remove empty values

as_json() → str

Get object as JSON string

Returns
JSON

Return type
str

iter_keys() → Generator[Any, None, None]

Iterate over keys

Returns

iter_values() → Generator[Any, None, None]

Iterate over values

Returns

Fields

BaseField

class aiogram.types.fields.**BaseField**(*, *base=None*, *default=None*, *alias=None*, *on_change=None*)

Bases: object

Base field (prop)

Init prop

Parameters

- **base** – class for child element
- **default** – default value
- **alias** – alias name (for e.g. field 'from' has to be named 'from_user' as 'from' is a builtin Python keyword
- **on_change** – callback will be called when value is changed

get_value(*instance*)

Get value for the current object instance

Parameters
instance –

Returns

set_value(*instance*, *value*, *parent=None*)

Set prop value

Parameters

- **instance** –
- **value** –
- **parent** –

Returns

abstract serialize(*value*)

Serialize value to python

Parameters
value –

Returns

abstract deserialize(*value*, *parent=None*)

Deserialize python object value to TelegramObject value

export(*instance*)

Alias for *serialize* but for current Object instance

Parameters

instance –

Returns

Field

class aiogram.types.fields.**Field**(*, *base=None*, *default=None*, *alias=None*, *on_change=None*)

Bases: *BaseField*

Simple field

Init prop

Parameters

- **base** – class for child element
- **default** – default value
- **alias** – alias name (for e.g. field 'from' has to be named 'from_user' as 'from' is a builtin Python keyword
- **on_change** – callback will be called when value is changed

serialize(*value*)

Serialize value to python

Parameters

value –

Returns

deserialize(*value*, *parent=None*)

Deserialize python object value to TelegramObject value

ListField

class aiogram.types.fields.**ListField**(**args*, ***kwargs*)

Bases: *Field*

The field contains a list of objects

Init prop

Parameters

- **base** – class for child element
- **default** – default value
- **alias** – alias name (for e.g. field 'from' has to be named 'from_user' as 'from' is a builtin Python keyword
- **on_change** – callback will be called when value is changed

serialize(*value*)

Serialize value to python

Parameters

value –

Returns

deserialize(*value*, *parent=None*)

Deserialize python object value to TelegramObject value

ListOfLists

class aiogram.types.fields.**ListOfLists**(*, *base=None*, *default=None*, *alias=None*, *on_change=None*)

Bases: *Field*

Init prop

Parameters

- **base** – class for child element
- **default** – default value
- **alias** – alias name (for e.g. field 'from' has to be named 'from_user' as 'from' is a builtin Python keyword
- **on_change** – callback will be called when value is changed

serialize(*value*)

Serialize value to python

Parameters

value –

Returns

deserialize(*value*, *parent=None*)

Deserialize python object value to TelegramObject value

DateTimeField

class aiogram.types.fields.**DateTimeField**(*, *base=None*, *default=None*, *alias=None*, *on_change=None*)

Bases: *Field*

In this field stored datetime

in: unixtime out: datetime

Init prop

Parameters

- **base** – class for child element
- **default** – default value
- **alias** – alias name (for e.g. field 'from' has to be named 'from_user' as 'from' is a builtin Python keyword
- **on_change** – callback will be called when value is changed

serialize(*value: datetime*)

Serialize value to python

Parameters
value –

Returns

deserialize(*value*, *parent=None*)

Deserialize python object value to TelegramObject value

TextField

class aiogram.types.fields.**TextField**(*, *prefix=None*, *suffix=None*, *default=None*, *alias=None*)

Bases: *Field*

Init prop

Parameters

- **base** – class for child element
- **default** – default value
- **alias** – alias name (for e.g. field 'from' has to be named 'from_user' as 'from' is a builtin Python keyword
- **on_change** – callback will be called when value is changed

serialize(*value*)

Serialize value to python

Parameters
value –

Returns

deserialize(*value*, *parent=None*)

Deserialize python object value to TelegramObject value

Mixins

Downloadable

class aiogram.types.mixins.**Downloadable**

Bases: object

Mixin for files

async download(*destination=None*, *timeout=30*, *chunk_size=65536*, *seek=True*, *make_dirs=True*, *, *destination_dir: Optional[Union[str, Path]] = None*, *destination_file: Optional[Union[str, Path, IOBase]] = None*)

Download file

At most one of these parameters can be used: :param destination_dir:, :param destination_file:

Parameters

- **destination** – deprecated, use :param destination_dir: or :param destination_file: instead
- **timeout** – Integer
- **chunk_size** – Integer
- **seek** – Boolean - go to start of file when downloading is finished.
- **make_dirs** – Make dirs if not exist
- **destination_dir** – directory for saving files
- **destination_file** – path to the file or instance of `io.IOBase`. For e. g. `io.BytesIO`

Returns
destination

async get_file()

Get file information

Returns
`aiogram.types.File`

async get_url()

Get file url.

Attention!! This method has security vulnerabilities for the reason that result contains bot's *access token* in open form. Use at your own risk!

Returns
url

Types

StickerSet

class `aiogram.types.sticker_set.`**StickerSet**(*conf: Optional[Dict[str, Any]] = None*, ***kwargs: Any*)

Bases: *TelegramObject*

This object represents a sticker set.

https://core.telegram.org/bots/api#stickerset

Deserialize object

Parameters

- **conf** –
- **kwargs** –

EncryptedCredentials

class aiogram.types.encrypted_credentials.**EncryptedCredentials**(*conf: Optional[Dict[str, Any]] = None*, ***kwargs: Any*)

Bases: *TelegramObject*

Contains data required for decrypting and authenticating EncryptedPassportElement. See the Telegram Passport Documentation for a complete description of the data decryption and authentication processes.

https://core.telegram.org/bots/api#encryptedcredentials

Deserialize object

Parameters

- **conf** –
- **kwargs** –

CallbackQuery

class aiogram.types.callback_query.**CallbackQuery**(*conf: Optional[Dict[str, Any]] = None*, ***kwargs: Any*)

Bases: *TelegramObject*

This object represents an incoming callback query from a callback button in an inline keyboard.

If the button that originated the query was attached to a message sent by the bot, the field message will be present.

If the button was attached to a message sent via the bot (in inline mode), the field inline_message_id will be present.

Exactly one of the fields data or game_short_name will be present.

https://core.telegram.org/bots/api#callbackquery

Deserialize object

Parameters

- **conf** –
- **kwargs** –

async answer(*text: Optional[String] = None*, *show_alert: Optional[Boolean] = None*, *url: Optional[String] = None*, *cache_time: Optional[Integer] = None*)

Use this method to send answers to callback queries sent from inline keyboards. The answer will be displayed to the user as a notification at the top of the chat screen or as an alert.

Alternatively, the user can be redirected to the specified Game URL. For this option to work, you must first create a game for your bot via @Botfather and accept the terms. Otherwise, you may use links like t.me/your_bot?start=XXXX that open your bot with a parameter.

Source: https://core.telegram.org/bots/api#answercallbackquery

Parameters

- **text** (typing.Optional[base.String]) – Text of the notification. If not specified, nothing will be shown to the user, 0-200 characters
- **show_alert** (typing.Optional[base.Boolean]) – If true, an alert will be shown by the client instead of a notification at the top of the chat screen. Defaults to false.

- **url** (typing.Optional[base.String]) – URL that will be opened by the user's client.
- **cache_time** (typing.Optional[base.Integer]) – The maximum amount of time in seconds that the result of the callback query may be cached client-side.

Returns
On success, True is returned.

Return type
base.Boolean

SuccessfulPayment

class aiogram.types.successful_payment.**SuccessfulPayment**(*conf: Optional[Dict[str, Any]] = None*, ***kwargs: Any*)

Bases: *TelegramObject*

This object contains basic information about a successful payment.

https://core.telegram.org/bots/api#successfulpayment

Deserialize object

Parameters

- **conf** –
- **kwargs** –

MessageEntity

class aiogram.types.message_entity.**MessageEntity**(*type: String*, *offset: Integer*, *length: Integer*, *url: Optional[String] = None*, *user: Optional[User] = None*, *language: Optional[String] = None*, *custom_emoji_id: Optional[String] = None*, ***kwargs*)

Bases: *TelegramObject*

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

https://core.telegram.org/bots/api#messageentity

Deserialize object

Parameters

- **conf** –
- **kwargs** –

get_text(*text*)

Get value of entity

Parameters
text – full text

Returns
part of text

parse(*text*, *as_html=True*)

Get entity value with markup

Parameters

- **text** – original text
- **as_html** – as html?

Returns

entity text with markup

MessageEntityType

class aiogram.types.message_entity.**MessageEntityType**

Bases: Helper

List of entity types

Key
MENTION

Key
HASHTAG

Key
CASHTAG

Key
BOT_COMMAND

Key
URL

Key
EMAIL

Key
PHONE_NUMBER

Key
BOLD

Key
ITALIC

Key
UNDERLINE

Key
STRIKETHROUGH

Key
SPOILER

Key
CODE

Key
PRE

Key
TEXT_LINK

Key
TEXT_MENTION

Key
CUSTOM_EMOJI

ShippingQuery

class aiogram.types.shipping_query.**ShippingQuery**(*conf: Optional[Dict[str, Any]] = None*, ***kwargs: Any*)

Bases: *TelegramObject*

This object contains information about an incoming shipping query.

https://core.telegram.org/bots/api#shippingquery

Deserialize object

Parameters

- **conf** –
- **kwargs** –

PassportData

class aiogram.types.passport_data.**PassportData**(*conf: Optional[Dict[str, Any]] = None*, ***kwargs: Any*)

Bases: *TelegramObject*

Contains information about Telegram Passport data shared with the bot by the user.

https://core.telegram.org/bots/api#passportdata

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InlineKeyboardMarkup

class aiogram.types.inline_keyboard.**InlineKeyboardMarkup**(*row_width=3*, *inline_keyboard=None*, ***kwargs*)

Bases: *TelegramObject*

This object represents an inline keyboard that appears right next to the message it belongs to.

Note: This will only work in Telegram versions released after 9 April, 2016. Older clients will display unsupported message.

https://core.telegram.org/bots/api#inlinekeyboardmarkup

Deserialize object

Parameters

- **conf** –

- **kwargs –

add(**args*)

Add buttons

Parameters
args –

Returns
self

Return type
types.InlineKeyboardMarkup

row(**buttons*)

Add row

Parameters
buttons –

Returns
self

Return type
types.InlineKeyboardMarkup

insert(*button*)

Insert button to last row

Parameters
button –

Returns
self

Return type
types.InlineKeyboardMarkup

InlineKeyboardButton

class aiogram.types.inline_keyboard.**InlineKeyboardButton**(*text: String, url: Optional[String] = None, login_url: Optional[LoginUrl] = None, callback_data: Optional[String] = None, switch_inline_query: Optional[String] = None, switch_inline_query_current_chat: Optional[String] = None, callback_game: Optional[CallbackGame] = None, pay: Optional[Boolean] = None, web_app: Optional[WebAppInfo] = None, **kwargs*)

Bases: *TelegramObject*

This object represents one button of an inline keyboard. You must use exactly one of the optional fields.

https://core.telegram.org/bots/api#inlinekeyboardbutton

Deserialize object

Parameters

- **conf** –
- **kwargs** –

User

class aiogram.types.user.**User**(*conf: Optional[Dict[str, Any]] = None*, ***kwargs: Any*)

Bases: *TelegramObject*

This object represents a Telegram user or bot.

https://core.telegram.org/bots/api#user

Deserialize object

Parameters

- **conf** –
- **kwargs** –

property full_name

You can get full name of user.

Returns
str

property mention

You can get user's username to mention him Full name will be returned if user has no username

Returns
str

property locale: Optional[Locale]

Get user's locale

Returns
babel.core.Locale

Video

class aiogram.types.video.**Video**(*conf: Optional[Dict[str, Any]] = None*, ***kwargs: Any*)

Bases: *TelegramObject*, *Downloadable*

This object represents a video file.

https://core.telegram.org/bots/api#video

Deserialize object

Parameters

- **conf** –
- **kwargs** –

EncryptedPassportElement

class aiogram.types.encrypted_passport_element.**EncryptedPassportElement**(*conf: Optional[Dict[str, Any]] = None, **kwargs: Any*)

Bases: *TelegramObject*

Contains information about documents or other Telegram Passport elements shared with the bot by the user.

https://core.telegram.org/bots/api#encryptedpassportelement

Deserialize object

Parameters

- **conf** –
- **kwargs** –

Game

class aiogram.types.game.**Game**(*conf: Optional[Dict[str, Any]] = None, **kwargs: Any*)

Bases: *TelegramObject*

This object represents a game.

Use BotFather to create and edit games, their short names will act as unique identifiers.

https://core.telegram.org/bots/api#game

Deserialize object

Parameters

- **conf** –
- **kwargs** –

File

class aiogram.types.file.**File**(*conf: Optional[Dict[str, Any]] = None, **kwargs: Any*)

Bases: *TelegramObject*, *Downloadable*

This object represents a file ready to be downloaded.

The file can be downloaded via the link https://api.telegram.org/file/bot<token>/<file_path>.

It is guaranteed that the link will be valid for at least 1 hour. When the link expires, a new one can be requested by calling getFile.

Maximum file size to download is 20 MB

https://core.telegram.org/bots/api#file

Deserialize object

Parameters

- **conf** –
- **kwargs** –

LabeledPrice

class aiogram.types.labeled_price.**LabeledPrice**(*label: String*, *amount: Integer*)

Bases: *TelegramObject*

This object represents a portion of the price for goods or services.

https://core.telegram.org/bots/api#labeledprice

Deserialize object

Parameters

- **conf** –
- **kwargs** –

CallbackGame

class aiogram.types.callback_game.**CallbackGame**(*conf: Optional[Dict[str, Any]] = None*, ***kwargs: Any*)

Bases: *TelegramObject*

A placeholder, currently holds no information. Use BotFather to set up your game.

https://core.telegram.org/bots/api#callbackgame

Deserialize object

Parameters

- **conf** –
- **kwargs** –

ReplyKeyboardMarkup

class aiogram.types.reply_keyboard.**ReplyKeyboardMarkup**(*keyboard: Optional[List[List[KeyboardButton]]] = None*, *resize_keyboard: Optional[Boolean] = None*, *one_time_keyboard: Optional[Boolean] = None*, *input_field_placeholder: Optional[String] = None*, *selective: Optional[Boolean] = None*, *row_width: Integer = 3*, *is_persistent: Optional[Boolean] = None*, *conf=None*)

Bases: *TelegramObject*

This object represents a custom keyboard with reply options (see https://core.telegram.org/bots#keyboards to bots for details and examples).

https://core.telegram.org/bots/api#replykeyboardmarkup

Deserialize object

Parameters

- **conf** –
- **kwargs** –

add(**args*)

Add buttons

Parameters
args –

Returns
self

Return type
types.ReplyKeyboardMarkup

row(**args*)

Add row

Parameters
args –

Returns
self

Return type
types.ReplyKeyboardMarkup

insert(*button*)

Insert button to last row

Parameters
button –

Returns
self

Return type
types.ReplyKeyboardMarkup

KeyboardButton

class aiogram.types.reply_keyboard.**KeyboardButton**(*text: String, request_user: Optional[KeyboardButtonRequestUser] = None, request_chat: Optional[KeyboardButtonRequestChat] = None, request_contact: Optional[Boolean] = None, request_location: Optional[Boolean] = None, request_poll: Optional[KeyboardButtonPollType] = None, web_app: Optional[WebAppInfo] = None, **kwargs*)

Bases: *TelegramObject*

This object represents one button of the reply keyboard. For simple text buttons String can be used instead of this object to specify text of the button. Optional fields request_contact, request_location, and request_poll are mutually exclusive. Note: request_contact and request_location options will only work in Telegram versions released after 9 April, 2016. Older clients will ignore them. Note: request_poll option will only work in Telegram versions released after 23 January, 2020. Older clients will receive unsupported message.

https://core.telegram.org/bots/api#keyboardbutton

Deserialize object

Parameters

- **conf** –
- **kwargs** –

ReplyKeyboardRemove

class aiogram.types.reply_keyboard.**ReplyKeyboardRemove**(*selective: Optional[Boolean] = None*)

Bases: *TelegramObject*

Upon receiving a message with this object, Telegram clients will remove the current custom keyboard and display the default letter-keyboard. By default, custom keyboards are displayed until a new keyboard is sent by a bot. An exception is made for one-time keyboards that are hidden immediately after the user presses a button (see ReplyKeyboardMarkup).

https://core.telegram.org/bots/api#replykeyboardremove

Deserialize object

Parameters

- **conf** –
- **kwargs** –

Chat

class aiogram.types.chat.**Chat**(*conf: Optional[Dict[str, Any]] = None*, ***kwargs: Any*)

Bases: *TelegramObject*

This object represents a chat.

https://core.telegram.org/bots/api#chat

Deserialize object

Parameters

- **conf** –
- **kwargs** –

property mention: Optional[String]

Get mention if a Chat has a username, or get full name if this is a Private Chat, otherwise None is returned

property shifted_id: int

Get shifted id of chat, e.g. for private links

For example: -1001122334455 -> 1122334455

async get_url() → String

Use this method to get chat link. Private chat returns user link. Other chat types return either username link (if they are public) or invite link (if they are private). :return: link :rtype: `base.String`

async update_chat()

Use this method to update Chat data

Returns
None

async set_photo(*photo: InputFile*) → Boolean

Use this method to set a new profile photo for the chat. Photos can't be changed for private chats. The bot must be an administrator in the chat for this to work and must have the appropriate admin rights.

Note: In regular groups (non-supergroups), this method will only work if the 'All Members Are Admins' setting is off in the target group.

Source: https://core.telegram.org/bots/api#setchatphoto

Parameters
photo (`base.InputFile`) – New chat photo, uploaded using multipart/form-data

Returns
Returns True on success.

Return type
`base.Boolean`

async delete_photo() → Boolean

Use this method to delete a chat photo. Photos can't be changed for private chats. The bot must be an administrator in the chat for this to work and must have the appropriate admin rights.

Note: In regular groups (non-supergroups), this method will only work if the 'All Members Are Admins' setting is off in the target group.

Source: https://core.telegram.org/bots/api#deletechatphoto

Returns
Returns True on success.

Return type
`base.Boolean`

async set_title(*title: String*) → Boolean

Use this method to change the title of a chat. Titles can't be changed for private chats. The bot must be an administrator in the chat for this to work and must have the appropriate admin rights.

Note: In regular groups (non-supergroups), this method will only work if the 'All Members Are Admins' setting is off in the target group.

Source: https://core.telegram.org/bots/api#setchattitle

Parameters
title (`base.String`) – New chat title, 1-255 characters

Returns
Returns True on success.

Return type
`base.Boolean`

async set_description(*description: String*) → Boolean

Use this method to change the description of a supergroup or a channel. The bot must be an administrator in the chat for this to work and must have the appropriate admin rights.

Source: https://core.telegram.org/bots/api#setchatdescription

Parameters
description (`typing.Optional[base.String]`) – New chat description, 0-255 characters

Returns
Returns True on success.

Return type
`base.Boolean`

async kick(*user_id: Integer*, *until_date: Optional[Union[Integer, datetime, timedelta]] = None*, *revoke_messages: Optional[Boolean] = None*) → Boolean

Use this method to kick a user from a group, a supergroup or a channel. In the case of supergroups and channels, the user will not be able to return to the chat on their own using invite links, etc., unless unbanned first.

The bot must be an administrator in the chat for this to work and must have the appropriate admin rights.

Source: https://core.telegram.org/bots/api#kickchatmember

Parameters

- **user_id** (`base.Integer`) – Unique identifier of the target user
- **until_date** (`typing.Union[base.Integer, datetime.datetime, datetime.timedelta, None]`) – Date when the user will be unbanned. If user is banned for more than 366 days or less than 30 seconds from the current time they are considered to be banned forever. Applied for supergroups and channels only.
- **revoke_messages** (`typing.Optional[base.Boolean]`) – Pass True to delete all messages from the chat for the user that is being removed. If False, the user will be able to see messages in the group that were sent before the user was removed. Always True for supergroups and channels.

Returns
Returns True on success

Return type
`base.Boolean`

async unban(*user_id: Integer*, *only_if_banned: Optional[Boolean] = None*) → Boolean

Use this method to unban a previously kicked user in a supergroup or channel. The user will not return to the group or channel automatically, but will be able to join via link, etc. The bot must be an administrator for this to work. By default, this method guarantees that after the call the user is not a member of the chat, but will be able to join it. So if the user is a member of the chat they will also be removed from the chat. If you don't want this, use the parameter only_if_banned. Returns True on success.

Source: https://core.telegram.org/bots/api#unbanchatmember

Parameters

- **user_id** (`base.Integer`) – Unique identifier of the target user
- **only_if_banned** (`typing.Optional[base.Boolean]`) – Do nothing if the user is not banned

Returns
Returns True on success.

Return type
`base.Boolean`

async restrict(*user_id: Integer*, *permissions: Optional[ChatPermissions] = None*, *until_date: Optional[Union[Integer, datetime, timedelta]] = None*, *can_send_messages: Optional[Boolean] = None*, *can_send_media_messages: Optional[Boolean] = None*, *can_send_other_messages: Optional[Boolean] = None*, *can_add_web_page_previews: Optional[Boolean] = None*) → Boolean

Use this method to restrict a user in a supergroup. The bot must be an administrator in the supergroup

for this to work and must have the appropriate admin rights. Pass True for all boolean parameters to lift restrictions from a user.

Source: https://core.telegram.org/bots/api#restrictchatmember

Parameters

- **user_id** (`base.Integer`) – Unique identifier of the target user
- **permissions** (`ChatPermissions`) – New user permissions
- **until_date** (`typing.Optional[base.Integer]`) – Date when restrictions will be lifted for the user, unix time.
- **can_send_messages** (`typing.Optional[base.Boolean]`) – Pass True, if the user can send text messages, contacts, locations and venues
- **can_send_media_messages** (`typing.Optional[base.Boolean]`) – Pass True, if the user can send audios, documents, photos, videos, video notes and voice notes, implies can_send_messages
- **can_send_other_messages** (`typing.Optional[base.Boolean]`) – Pass True, if the user can send animations, games, stickers and use inline bots, implies can_send_media_messages
- **can_add_web_page_previews** (`typing.Optional[base.Boolean]`) – Pass True, if the user may add web page previews to their messages, implies can_send_media_messages

Returns
Returns True on success.

Return type
`base.Boolean`

async promote(*user_id: Integer*, *is_anonymous: Optional[Boolean] = None*, *can_manage_chat: Optional[Boolean] = None*, *can_change_info: Optional[Boolean] = None*, *can_post_messages: Optional[Boolean] = None*, *can_edit_messages: Optional[Boolean] = None*, *can_delete_messages: Optional[Boolean] = None*, *can_manage_voice_chats: Optional[Boolean] = None*, *can_invite_users: Optional[Boolean] = None*, *can_restrict_members: Optional[Boolean] = None*, *can_pin_messages: Optional[Boolean] = None*, *can_promote_members: Optional[Boolean] = None*, *can_manage_video_chats: Optional[Boolean] = None*, *can_manage_topics: Optional[Boolean] = None*) → Boolean

Use this method to promote or demote a user in a supergroup or a channel. The bot must be an administrator in the chat for this to work and must have the appropriate admin rights. Pass False for all boolean parameters to demote a user.

Source: https://core.telegram.org/bots/api#promotechatmember

Parameters

- **user_id** (`base.Integer`) – Unique identifier of the target user
- **is_anonymous** (`typing.Optional[base.Boolean]`) – Pass True, if the administrator's presence in the chat is hidden
- **can_change_info** (`typing.Optional[base.Boolean]`) – Pass True, if the administrator can change chat title, photo and other settings
- **can_post_messages** (`typing.Optional[base.Boolean]`) – Pass True, if the administrator can create channel posts, channels only
- **can_edit_messages** (`typing.Optional[base.Boolean]`) – Pass True, if the administrator can edit messages of other users, channels only

- **can_delete_messages** (typing.Optional[base.Boolean]) – Pass True, if the administrator can delete messages of other users
- **can_invite_users** (typing.Optional[base.Boolean]) – Pass True, if the administrator can invite new users to the chat
- **can_restrict_members** (typing.Optional[base.Boolean]) – Pass True, if the administrator can restrict, ban or unban chat members
- **can_pin_messages** (typing.Optional[base.Boolean]) – Pass True, if the administrator can pin messages, supergroups only
- **can_promote_members** (typing.Optional[base.Boolean]) – Pass True, if the administrator can add new administrators with a subset of his own privileges or demote administrators that he has promoted, directly or indirectly (promoted by administrators that were appointed by him)

Returns
Returns True on success.

Return type
base.Boolean

async set_permissions(*permissions: ChatPermissions*) → Boolean

Use this method to set default chat permissions for all members. The bot must be an administrator in the group or a supergroup for this to work and must have the can_restrict_members admin rights.

Returns True on success.

Parameters
permissions – New default chat permissions

Returns
True on success.

async set_administrator_custom_title(*user_id: Integer*, *custom_title: String*) → Boolean

Use this method to set a custom title for an administrator in a supergroup promoted by the bot.

Returns True on success.

Source: https://core.telegram.org/bots/api#setchatadministratorcustomtitle

Parameters

- **user_id** – Unique identifier of the target user
- **custom_title** – New custom title for the administrator; 0-16 characters, emoji are not allowed

Returns
True on success.

async pin_message(*message_id: Integer*, *disable_notification: Optional[Boolean] = False*) → Boolean

Use this method to add a message to the list of pinned messages in a chat. If the chat is not a private chat, the bot must be an administrator in the chat for this to work and must have the 'can_pin_messages' admin right in a supergroup or 'can_edit_messages' admin right in a channel. Returns True on success.

Source: https://core.telegram.org/bots/api#pinchatmessage

Parameters

- **message_id** (base.Integer) – Identifier of a message to pin

- **disable_notification** (typing.Optional[base.Boolean]) – Pass True, if it is not necessary to send a notification to all group members about the new pinned message

Returns
Returns True on success

Return type
base.Boolean

async unpin_message(*message_id: Optional[Integer] = None*) → Boolean

Use this method to remove a message from the list of pinned messages in a chat. If the chat is not a private chat, the bot must be an administrator in the chat for this to work and must have the 'can_pin_messages' admin right in a supergroup or 'can_edit_messages' admin right in a channel. Returns True on success.

Source: https://core.telegram.org/bots/api#unpinchatmessage

Parameters
message_id (typing.Optional[base.Integer]) – Identifier of a message to unpin. If not specified, the most recent pinned message (by sending date) will be unpinned.

Returns
Returns True on success

Return type
base.Boolean

async unpin_all_messages()

Use this method to clear the list of pinned messages in a chat. If the chat is not a private chat, the bot must be an administrator in the chat for this to work and must have the 'can_pin_messages' admin right in a supergroup or 'can_edit_messages' admin right in a channel. Returns True on success.

Source: https://core.telegram.org/bots/api#unpinallchatmessages

Returns
Returns True on success

Return type
base.Boolean

async leave() → Boolean

Use this method for your bot to leave a group, supergroup or channel.

Source: https://core.telegram.org/bots/api#leavechat

Returns
Returns True on success.

Return type
base.Boolean

async get_administrators() → List[Union[ChatMemberOwner, ChatMemberAdministrator]]

Use this method to get a list of administrators in a chat.

Source: https://core.telegram.org/bots/api#getchatadministrators

Returns
On success, returns an Array of ChatMember objects that contains information about all chat administrators except other bots. If the chat is a group or a supergroup and no administrators were appointed, only the creator will be returned.

Return type
typing.List[typing.Union[types.ChatMemberOwner, types.ChatMemberAdministrator]]

async get_member_count() → Integer

Use this method to get the number of members in a chat.

Source: https://core.telegram.org/bots/api#getchatmembercount

Returns
Returns Int on success.

Return type
`base.Integer`

async get_members_count() → Integer

Renamed to get_member_count.

async get_member(*user_id: Integer*) → *ChatMember*

Use this method to get information about a member of a chat.

Source: https://core.telegram.org/bots/api#getchatmember

Parameters
user_id (`base.Integer`) – Unique identifier of the target user

Returns
Returns a ChatMember object on success.

Return type
`types.ChatMember`

async set_sticker_set(*sticker_set_name: String*) → Boolean

Use this method to set a new group sticker set for a supergroup. The bot must be an administrator in the chat for this to work and must have the appropriate admin rights.

Use the field can_set_sticker_set optionally returned in getChat requests to check if the bot can use this method.

Source: https://core.telegram.org/bots/api#setchatstickerset

Parameters
sticker_set_name (`base.String`) – Name of the sticker set to be set as the group sticker set

Returns
Returns True on success

Return type
`base.Boolean`

async delete_sticker_set() → Boolean

Use this method to delete a group sticker set from a supergroup. The bot must be an administrator in the chat for this to work and must have the appropriate admin rights.

Use the field can_set_sticker_set optionally returned in getChat requests to check if the bot can use this method.

Source: https://core.telegram.org/bots/api#deletechatstickerset

Returns
Returns True on success

Return type
`base.Boolean`

async do(*action: String*, *message_thread_id: Optional[Integer] = None*) → Boolean

Use this method when you need to tell the user that something is happening on the bot's side. The status is set for 5 seconds or less (when a message arrives from your bot, Telegram clients clear its typing status).

We only recommend using this method when a response from the bot will take a noticeable amount of time to arrive.

Source: https://core.telegram.org/bots/api#sendchataction

Parameters

- **action** (`base.String`) – Type of action to broadcast.
- **message_thread_id** (`typing.Optional[base.Integer]`) – Unique identifier for the target message thread; supergroups only

Returns
Returns True on success.

Return type
`base.Boolean`

async export_invite_link() → String

Use this method to export an invite link to a supergroup or a channel. The bot must be an administrator in the chat for this to work and must have the appropriate admin rights.

Source: https://core.telegram.org/bots/api#exportchatinvitelink

Returns
Returns exported invite link as String on success.

Return type
`base.String`

async create_invite_link(*expire_date: Optional[Union[Integer, datetime, timedelta]] = None*, *member_limit: Optional[Integer] = None*, *name: Optional[String] = None*, *creates_join_request: Optional[Boolean] = None*) → ChatInviteLink

Shortcut for createChatInviteLink method.

async edit_invite_link(*invite_link: String*, *expire_date: Optional[Union[Integer, datetime, timedelta]] = None*, *member_limit: Optional[Integer] = None*, *name: Optional[String] = None*, *creates_join_request: Optional[Boolean] = None*) → ChatInviteLink

Shortcut for editChatInviteLink method.

async revoke_invite_link(*invite_link: String*) → ChatInviteLink

Shortcut for revokeChatInviteLink method.

async delete_message(*message_id: Integer*) → Boolean

Shortcut for deleteMessage method.

async ban_sender_chat(*sender_chat_id: Integer*)

Shortcut for banChatSenderChat method.

async unban_sender_chat(*sender_chat_id: Integer*)

Shortcut for unbanChatSenderChat method.

ChatType

class aiogram.types.chat.**ChatType**

Bases: Helper

List of chat types

Key
SENDER

Key
PRIVATE

Key
GROUP

Key
SUPER_GROUP

Key
SUPERGROUP

Key
CHANNEL

classmethod is_private(*obj*) → bool

Check chat is private

Parameters
obj –

Returns

classmethod is_group(*obj*) → bool

Check chat is group

Parameters
obj –

Returns

classmethod is_super_group(*obj*) → bool

Check chat is super-group

Parameters
obj –

Returns

classmethod is_group_or_super_group(*obj*) → bool

Check chat is group or super-group

Parameters
obj –

Returns

classmethod is_channel(*obj*) → bool

Check chat is channel

Parameters
obj –

Returns

ChatActions

class aiogram.types.chat.**ChatActions**

Bases: Helper

List of chat actions

Key
TYPING

Key
UPLOAD_PHOTO

Key
RECORD_VIDEO

Key
UPLOAD_VIDEO

Key
RECORD_AUDIO

Key
UPLOAD_AUDIO

Key
UPLOAD_DOCUMENT

Key
FIND_LOCATION

Key
RECORD_VIDEO_NOTE

Key
UPLOAD_VIDEO_NOTE

classmethod calc_timeout(*text*, *timeout=0.8*)

Calculate timeout for text

Parameters

- **text** –
- **timeout** –

Returns

async classmethod typing(*sleep=None*)

Do typing

Parameters
sleep – sleep timeout

Returns

async classmethod upload_photo(*sleep=None*)

Do upload_photo

Parameters
sleep – sleep timeout

Returns

async classmethod record_video(*sleep=None*)

Do record video

Parameters
sleep – sleep timeout

Returns

async classmethod upload_video(*sleep=None*)

Do upload video

Parameters
sleep – sleep timeout

Returns

async classmethod record_audio(*sleep=None*)

Do record audio

Parameters
sleep – sleep timeout

Returns

async classmethod upload_audio(*sleep=None*)

Do upload audio

Parameters
sleep – sleep timeout

Returns

async classmethod record_voice(*sleep=None*)

Do record voice

Parameters
sleep – sleep timeout

Returns

async classmethod upload_voice(*sleep=None*)

Do upload voice

Parameters
sleep – sleep timeout

Returns

async classmethod upload_document(*sleep=None*)

Do upload document

Parameters
sleep – sleep timeout

Returns

async classmethod find_location(*sleep=None*)

Do find location

Parameters
sleep – sleep timeout

Returns

async classmethod record_video_note(*sleep=None*)

Do record video note

Parameters

sleep – sleep timeout

Returns

async classmethod upload_video_note(*sleep=None*)

Do upload video note

Parameters

sleep – sleep timeout

Returns

async classmethod choose_sticker(*sleep=None*)

Do choose sticker

Parameters

sleep – sleep timeout

Returns

Document

class aiogram.types.document.**Document**(*conf: Optional[Dict[str, Any]] = None*, ***kwargs: Any*)

Bases: *TelegramObject*, *Downloadable*

This object represents a general file (as opposed to photos, voice messages and audio files).

https://core.telegram.org/bots/api#document

Deserialize object

Parameters

- **conf** –
- **kwargs** –

Audio

class aiogram.types.audio.**Audio**(*conf: Optional[Dict[str, Any]] = None*, ***kwargs: Any*)

Bases: *TelegramObject*, *Downloadable*

This object represents an audio file to be treated as music by the Telegram clients.

https://core.telegram.org/bots/api#audio

Deserialize object

Parameters

- **conf** –
- **kwargs** –

ForceReply

class aiogram.types.force_reply.**ForceReply**(*conf: Optional[Dict[str, Any]] = None*, ***kwargs: Any*)

Bases: *TelegramObject*

Upon receiving a message with this object, Telegram clients will display a reply interface to the user (act as if the user has selected the bot's message and tapped 'Reply'). This can be extremely useful if you want to create user-friendly step-by-step interfaces without having to sacrifice privacy mode.

https://core.telegram.org/bots/api#forcereply

Deserialize object

Parameters

- **conf** –
- **kwargs** –

classmethod create(*input_field_placeholder: Optional[String] = None*, *selective: Optional[Boolean] = None*) → *ForceReply*

Create new force reply

Parameters

- **selective** –
- **input_field_placeholder** –

Returns

PassportElementError

class aiogram.types.passport_element_error.**PassportElementError**(*conf: Optional[Dict[str, Any]] = None*, ***kwargs: Any*)

Bases: *TelegramObject*

This object represents an error in the Telegram Passport element which was submitted that should be resolved by the user.

https://core.telegram.org/bots/api#passportelementerror

Deserialize object

Parameters

- **conf** –
- **kwargs** –

PassportElementErrorDataField

class aiogram.types.passport_element_error.**PassportElementErrorDataField**(*source: String*, *type: String*, *field_name: String*, *data_hash: String*, *message: String*)

Bases: *PassportElementError*

Represents an issue in one of the data fields that was provided by the user. The error is considered resolved when the field's value changes.

https://core.telegram.org/bots/api#passportelementerrordatafield

Deserialize object

Parameters

- **conf** –
- **kwargs** –

PassportElementErrorFile

class aiogram.types.passport_element_error.**PassportElementErrorFile**(*source: String*, *type: String*, *file_hash: String*, *message: String*)

Bases: *PassportElementError*

Represents an issue with a document scan. The error is considered resolved when the file with the document scan changes.

https://core.telegram.org/bots/api#passportelementerrorfile

Deserialize object

Parameters

- **conf** –
- **kwargs** –

PassportElementErrorFiles

class aiogram.types.passport_element_error.**PassportElementErrorFiles**(*source: String*, *type: String*, *file_hashes: List[String]*, *message: String*)

Bases: *PassportElementError*

Represents an issue with a list of scans. The error is considered resolved when the list of files containing the scans changes.

https://core.telegram.org/bots/api#passportelementerrorfiles

Deserialize object

Parameters

- **conf** –
- **kwargs** –

PassportElementErrorFrontSide

class aiogram.types.passport_element_error.**PassportElementErrorFrontSide**(*source: String*, *type: String*, *file_hash: String*, *message: String*)

Bases: *PassportElementError*

Represents an issue with the front side of a document. The error is considered resolved when the file with the front side of the document changes.

https://core.telegram.org/bots/api#passportelementerrorfrontside

Deserialize object

Parameters

- **conf** –
- **kwargs** –

PassportElementErrorReverseSide

class aiogram.types.passport_element_error.**PassportElementErrorReverseSide**(*source: String*, *type: String*, *file_hash: String*, *message: String*)

Bases: *PassportElementError*

Represents an issue with the reverse side of a document. The error is considered resolved when the file with reverse side of the document changes.

https://core.telegram.org/bots/api#passportelementerrorreverseside

Deserialize object

Parameters

- **conf** –
- **kwargs** –

PassportElementErrorSelfie

class aiogram.types.passport_element_error.**PassportElementErrorSelfie**(*source: String*, *type: String*, *file_hash: String*, *message: String*)

Bases: *PassportElementError*

Represents an issue with the selfie with a document. The error is considered resolved when the file with the selfie changes.

https://core.telegram.org/bots/api#passportelementerrorselfie

Deserialize object

Parameters

- **conf** –
- **kwargs** –

ShippingAddress

class aiogram.types.shipping_address.**ShippingAddress**(*conf: Optional[Dict[str, Any]] = None*, ***kwargs: Any*)

Bases: *TelegramObject*

This object represents a shipping address.

https://core.telegram.org/bots/api#shippingaddress

Deserialize object

Parameters

- **conf** –
- **kwargs** –

ResponseParameters

class aiogram.types.response_parameters.**ResponseParameters**(*conf: Optional[Dict[str, Any]] = None*, ***kwargs: Any*)

Bases: *TelegramObject*

Contains information about why a request was unsuccessful.

https://core.telegram.org/bots/api#responseparameters

Deserialize object

Parameters

- **conf** –
- **kwargs** –

OrderInfo

class aiogram.types.order_info.**OrderInfo**(*conf: Optional[Dict[str, Any]] = None*, ***kwargs: Any*)

Bases: *TelegramObject*

This object represents information about an order.

https://core.telegram.org/bots/api#orderinfo

Deserialize object

Parameters

- **conf** –
- **kwargs** –

GameHighScore

class aiogram.types.game_high_score.**GameHighScore**(*conf: Optional[Dict[str, Any]] = None*, ***kwargs: Any*)

Bases: *TelegramObject*

This object represents one row of the high scores table for a game. And that's about all we've got for now. If you've got any questions, please check out our Bot FAQ

https://core.telegram.org/bots/api#gamehighscore

Deserialize object

Parameters

- **conf** –
- **kwargs** –

Sticker

class aiogram.types.sticker.**Sticker**(*conf: Optional[Dict[str, Any]] = None*, ***kwargs: Any*)

Bases: *TelegramObject*, *Downloadable*

This object represents a sticker.

https://core.telegram.org/bots/api#sticker

Deserialize object

Parameters

- **conf** –
- **kwargs** –

async set_position_in_set(*position: Integer*) → Boolean

Use this method to move a sticker in a set created by the bot to a specific position.

Source: https://core.telegram.org/bots/api#setstickerpositioninset

Parameters
position (base.Integer) – New sticker position in the set, zero-based

Returns
Returns True on success

Return type
base.Boolean

async delete_from_set() → Boolean

Use this method to delete a sticker from a set created by the bot.

Source: https://core.telegram.org/bots/api#deletestickerfromset

Returns
Returns True on success

Return type
base.Boolean

InlineQuery

class aiogram.types.inline_query.**InlineQuery**(*conf: Optional[Dict[str, Any]] = None*, ***kwargs: Any*)

Bases: *TelegramObject*

This object represents an incoming inline query.

When the user sends an empty query, your bot could return some default or trending results.

https://core.telegram.org/bots/api#inlinequery

Deserialize object

Parameters

- **conf** –
- **kwargs** –

async answer(*results: List[InlineQueryResult], cache_time: Optional[Integer] = None, is_personal: Optional[Boolean] = None, next_offset: Optional[String] = None, switch_pm_text: Optional[String] = None, switch_pm_parameter: Optional[String] = None*)

Use this method to send answers to an inline query. No more than 50 results per query are allowed.

Source: https://core.telegram.org/bots/api#answerinlinequery

Parameters

- **results** (typing.List[types.InlineQueryResult]) – A JSON-serialized array of results for the inline query
- **cache_time** (typing.Optional[base.Integer]) – The maximum amount of time in seconds that the result of the inline query may be cached on the server. Defaults to 300.
- **is_personal** (typing.Optional[base.Boolean]) – Pass True, if results may be cached on the server side only for the user that sent the query. By default, results may be returned to any user who sends the same query
- **next_offset** (typing.Optional[base.String]) – Pass the offset that a client should send in the next query with the same text to receive more results. Pass an empty string if there are no more results or if you don‘t support pagination. Offset length can't exceed 64 bytes.
- **switch_pm_text** (typing.Optional[base.String]) – If passed, clients will display a button with specified text that switches the user to a private chat with the bot and sends the bot a start message with the parameter switch_pm_parameter
- **switch_pm_parameter** (typing.Optional[base.String]) – Deep-linking parameter for the /start message sent to the bot when user presses the switch button. 1-64 characters, only A-Z, a-z, 0-9, _ and - are allowed.

Returns

On success, True is returned

Return type

base.Boolean

Location

class aiogram.types.location.**Location**(*conf: Optional[Dict[str, Any]] = None*, ***kwargs: Any*)

Bases: *TelegramObject*

This object represents a point on the map.

https://core.telegram.org/bots/api#location

Deserialize object

Parameters

- **conf** –
- **kwargs** –

Animation

class aiogram.types.animation.**Animation**(*conf: Optional[Dict[str, Any]] = None*, ***kwargs: Any*)

Bases: *TelegramObject*, *Downloadable*

You can provide an animation for your game so that it looks stylish in chats (check out Lumberjack for an example). This object represents an animation file to be displayed in the message containing a game.

https://core.telegram.org/bots/api#animation

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InputMedia

class aiogram.types.input_media.**InputMedia**(**args*, ***kwargs*)

Bases: *TelegramObject*

This object represents the content of a media message to be sent. It should be one of

- InputMediaAnimation
- InputMediaDocument
- InputMediaAudio
- InputMediaPhoto
- InputMediaVideo

That is only base class.

https://core.telegram.org/bots/api#inputmedia

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InputMediaAnimation

class aiogram.types.input_media.**InputMediaAnimation**(*media: InputFile, thumb: Optional[Union[InputFile, String]] = None, caption: Optional[String] = None, width: Optional[Integer] = None, height: Optional[Integer] = None, duration: Optional[Integer] = None, parse_mode: Optional[String] = None, caption_entities: Optional[List[MessageEntity]] = None, has_spoiler: Optional[Boolean] = None, **kwargs*)

Bases: *InputMedia*

Represents an animation file (GIF or H.264/MPEG-4 AVC video without sound) to be sent.

https://core.telegram.org/bots/api#inputmediaanimation

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InputMediaDocument

class aiogram.types.input_media.**InputMediaDocument**(*media: InputFile, thumb: Optional[Union[InputFile, String]] = None, caption: Optional[String] = None, parse_mode: Optional[String] = None, caption_entities: Optional[List[MessageEntity]] = None, disable_content_type_detection: Optional[Boolean] = None, **kwargs*)

Bases: *InputMedia*

Represents a general file to be sent.

https://core.telegram.org/bots/api#inputmediadocument

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InputMediaAudio

class aiogram.types.input_media.**InputMediaAudio**(*media: InputFile, thumb: Optional[Union[InputFile, String]] = None, caption: Optional[String] = None, duration: Optional[Integer] = None, performer: Optional[String] = None, title: Optional[String] = None, parse_mode: Optional[String] = None, caption_entities: Optional[List[MessageEntity]] = None, **kwargs*)

Bases: *InputMedia*

Represents an audio file to be treated as music to be sent.

https://core.telegram.org/bots/api#inputmediaaudio

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InputMediaPhoto

class aiogram.types.input_media.**InputMediaPhoto**(*media: InputFile, caption: Optional[String] = None, parse_mode: Optional[String] = None, caption_entities: Optional[List[MessageEntity]] = None, has_spoiler: Optional[Boolean] = None, **kwargs*)

Bases: *InputMedia*

Represents a photo to be sent.

https://core.telegram.org/bots/api#inputmediaphoto

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InputMediaVideo

class aiogram.types.input_media.**InputMediaVideo**(*media: InputFile, thumb: Optional[Union[InputFile, String]] = None, caption: Optional[String] = None, width: Optional[Integer] = None, height: Optional[Integer] = None, duration: Optional[Integer] = None, parse_mode: Optional[String] = None, caption_entities: Optional[List[MessageEntity]] = None, supports_streaming: Optional[Boolean] = None, has_spoiler: Optional[Boolean] = None, **kwargs*)

Bases: *InputMedia*

Represents a video to be sent.

https://core.telegram.org/bots/api#inputmediavideo

Deserialize object

Parameters

- **conf** –
- **kwargs** –

MediaGroup

class aiogram.types.input_media.**MediaGroup**(*medias: Optional[List[Union[InputMedia, Dict]]] = None*)

Bases: *TelegramObject*

Helper for sending media group

Deserialize object

Parameters

- **conf** –
- **kwargs** –

attach_many(**medias: Union[InputMedia, Dict]*)

Attach list of media

Parameters
medias –

attach(*media: Union[InputMedia, Dict]*)

Attach media

Parameters
media –

attach_audio(*audio: Union[InputMediaAudio, InputFile], thumb: Optional[Union[InputFile, String]] = None, caption: Optional[String] = None, duration: Optional[Integer] = None, performer: Optional[String] = None, title: Optional[String] = None, parse_mode: Optional[String] = None, caption_entities: Optional[List[MessageEntity]] = None*)

Attach audio

Parameters

- **audio** –
- **thumb** –
- **caption** –
- **duration** –
- **performer** –
- **title** –
- **parse_mode** –
- **caption_entities** –

attach_document(*document: Union[InputMediaDocument, InputFile], thumb: Optional[Union[InputFile, String]] = None, caption: Optional[String] = None, parse_mode: Optional[String] = None, caption_entities: Optional[List[MessageEntity]] = None, disable_content_type_detection: Optional[Boolean] = None*)

Attach document

Parameters

- **document** –
- **caption** –
- **thumb** –
- **parse_mode** –
- **caption_entities** –
- **disable_content_type_detection** –

attach_photo(*photo: Union[InputMediaPhoto, InputFile], caption: Optional[String] = None, parse_mode: Optional[String] = None, caption_entities: Optional[List[MessageEntity]] = None*)

Attach photo

Parameters

- **photo** –
- **caption** –
- **parse_mode** –
- **caption_entities** –

attach_video(*video: Union[InputMediaVideo, InputFile], thumb: Optional[Union[InputFile, String]] = None, caption: Optional[String] = None, width: Optional[Integer] = None, height: Optional[Integer] = None, duration: Optional[Integer] = None, parse_mode: Optional[String] = None, caption_entities: Optional[List[MessageEntity]] = None, supports_streaming: Optional[Boolean] = None*)

Attach video

Parameters

- **video** –
- **thumb** –
- **caption** –
- **width** –
- **height** –
- **duration** –
- **parse_mode** –
- **caption_entities** –
- **supports_streaming** –

to_python() → List

Get object as JSON serializable

Returns

InlineQueryResult

class aiogram.types.inline_query_result.**InlineQueryResult**(***kwargs*)

Bases: *TelegramObject*

This object represents one result of an inline query.

Telegram clients currently support results of the following 20 types

https://core.telegram.org/bots/api#inlinequeryresult

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InlineQueryResultArticle

class aiogram.types.inline_query_result.**InlineQueryResultArticle**(**, id: String, title: String, input_message_content: InputMessageContent, reply_markup: Optional[InlineKeyboardMarkup] = None, url: Optional[String] = None, hide_url: Optional[Boolean] = None, description: Optional[String] = None, thumb_url: Optional[String] = None, thumb_width: Optional[Integer] = None, thumb_height: Optional[Integer] = None*)

Bases: *InlineQueryResult*

Represents a link to an article or web page.

https://core.telegram.org/bots/api#inlinequeryresultarticle

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InlineQueryResultPhoto

class aiogram.types.inline_query_result.**InlineQueryResultPhoto**(*, *id: String*, *photo_url: String*, *thumb_url: String*, *photo_width: Optional[Integer] = None*, *photo_height: Optional[Integer] = None*, *title: Optional[String] = None*, *description: Optional[String] = None*, *caption: Optional[String] = None*, *parse_mode: Optional[String] = None*, *caption_entities: Optional[List[MessageEntity]] = None*, *reply_markup: Optional[InlineKeyboardMarkup] = None*, *input_message_content: Optional[InputMessageContent] = None*)

Bases: *InlineQueryResult*

Represents a link to a photo.

By default, this photo will be sent by the user with optional caption. Alternatively, you can use input_message_content to send a message with the specified content instead of the photo.

https://core.telegram.org/bots/api#inlinequeryresultphoto

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InlineQueryResultGif

class aiogram.types.inline_query_result.**InlineQueryResultGif**(*, *id: String*, *gif_url: String*, *gif_width: Optional[Integer] = None*, *gif_height: Optional[Integer] = None*, *gif_duration: Optional[Integer] = None*, *thumb_url: Optional[String] = None*, *title: Optional[String] = None*, *caption: Optional[String] = None*, *parse_mode: Optional[String] = None*, *reply_markup: Optional[InlineKeyboardMarkup] = None*, *caption_entities: Optional[List[MessageEntity]] = None*, *input_message_content: Optional[InputMessageContent] = None*)

Bases: *InlineQueryResult*

Represents a link to an animated GIF file.

By default, this animated GIF file will be sent by the user with optional caption. Alternatively, you can use input_message_content to send a message with the specified content instead of the animation.

https://core.telegram.org/bots/api#inlinequeryresultgif

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InlineQueryResultMpeg4Gif

class aiogram.types.inline_query_result.**InlineQueryResultMpeg4Gif**(*, *id: String*, *mpeg4_url: String*, *thumb_url: String*, *mpeg4_width: Optional[Integer] = None*, *mpeg4_height: Optional[Integer] = None*, *mpeg4_duration: Optional[Integer] = None*, *title: Optional[String] = None*, *caption: Optional[String] = None*, *parse_mode: Optional[String] = None*, *reply_markup: Optional[InlineKeyboardMarkup] = None*, *caption_entities: Optional[List[MessageEntity]] = None*, *input_message_content: Optional[InputMessageContent] = None*)

Bases: *InlineQueryResult*

Represents a link to a video animation (H.264/MPEG-4 AVC video without sound).

By default, this animated MPEG-4 file will be sent by the user with optional caption. Alternatively, you can use input_message_content to send a message with the specified content instead of the animation.

https://core.telegram.org/bots/api#inlinequeryresultmpeg4gif

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InlineQueryResultVideo

class aiogram.types.inline_query_result.**InlineQueryResultVideo**(*, *id: String*, *video_url: String*, *mime_type: String*, *thumb_url: String*, *title: String*, *caption: Optional[String] = None*, *parse_mode: Optional[String] = None*, *video_width: Optional[Integer] = None*, *video_height: Optional[Integer] = None*, *video_duration: Optional[Integer] = None*, *description: Optional[String] = None*, *reply_markup: Optional[InlineKeyboardMarkup] = None*, *caption_entities: Optional[List[MessageEntity]] = None*, *input_message_content: Optional[InputMessageContent] = None*)

Bases: *InlineQueryResult*

Represents a link to a page containing an embedded video player or a video file.

By default, this video file will be sent by the user with an optional caption. Alternatively, you can use input_message_content to send a message with the specified content instead of the video.

If an InlineQueryResultVideo message contains an embedded video (e.g., YouTube), you must replace its content using input_message_content.

https://core.telegram.org/bots/api#inlinequeryresultvideo

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InlineQueryResultAudio

class aiogram.types.inline_query_result.**InlineQueryResultAudio**(*, *id: String*, *audio_url: String*, *title: String*, *caption: Optional[String] = None*, *parse_mode: Optional[String] = None*, *performer: Optional[String] = None*, *audio_duration: Optional[Integer] = None*, *reply_markup: Optional[InlineKeyboardMarkup] = None*, *caption_entities: Optional[List[MessageEntity]] = None*, *input_message_content: Optional[InputMessageContent] = None*)

Bases: *InlineQueryResult*

Represents a link to an mp3 audio file. By default, this audio file will be sent by the user. Alternatively, you can use input_message_content to send a message with the specified content instead of the audio.

Note: This will only work in Telegram versions released after 9 April, 2016. Older clients will ignore them.

https://core.telegram.org/bots/api#inlinequeryresultaudio

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InlineQueryResultVoice

class aiogram.types.inline_query_result.**InlineQueryResultVoice**(*, *id: String*, *voice_url: String*, *title: String*, *caption: Optional[String] = None*, *parse_mode: Optional[String] = None*, *voice_duration: Optional[Integer] = None*, *reply_markup: Optional[InlineKeyboardMarkup] = None*, *caption_entities: Optional[List[MessageEntity]] = None*, *input_message_content: Optional[InputMessageContent] = None*)

Bases: *InlineQueryResult*

Represents a link to a voice recording in an .ogg container encoded with OPUS.

By default, this voice recording will be sent by the user. Alternatively, you can use input_message_content to send a message with the specified content instead of the the voice message.

Note: This will only work in Telegram versions released after 9 April, 2016. Older clients will ignore them.

https://core.telegram.org/bots/api#inlinequeryresultvoice

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InlineQueryResultDocument

class aiogram.types.inline_query_result.**InlineQueryResultDocument**(*, *id: String*, *title: String*, *caption: Optional[String] = None*, *parse_mode: Optional[String] = None*, *caption_entities: Optional[List[MessageEntity]] = None*, *document_url: Optional[String] = None*, *mime_type: Optional[String] = None*, *description: Optional[String] = None*, *reply_markup: Optional[InlineKeyboardMarkup] = None*, *input_message_content: Optional[InputMessageContent] = None*, *thumb_url: Optional[String] = None*, *thumb_width: Optional[Integer] = None*, *thumb_height: Optional[Integer] = None*)

Bases: *InlineQueryResult*

Represents a link to a file.

By default, this file will be sent by the user with an optional caption. Alternatively, you can use input_message_content to send a message with the specified content instead of the file. Currently, only .PDF and .ZIP files can be sent using this method.

Note: This will only work in Telegram versions released after 9 April, 2016. Older clients will ignore them.

https://core.telegram.org/bots/api#inlinequeryresultdocument

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InlineQueryResultLocation

class aiogram.types.inline_query_result.**InlineQueryResultLocation**(*, *id: String*, *latitude: Float*, *longitude: Float*, *title: String*, *horizontal_accuracy: Optional[Float] = None*, *live_period: Optional[Integer] = None*, *heading: Optional[Integer] = None*, *proximity_alert_radius: Optional[Integer] = None*, *reply_markup: Optional[InlineKeyboardMarkup] = None*, *input_message_content: Optional[InputMessageContent] = None*, *thumb_url: Optional[String] = None*, *thumb_width: Optional[Integer] = None*, *thumb_height: Optional[Integer] = None*)

Bases: *InlineQueryResult*

Represents a location on a map.

By default, the location will be sent by the user. Alternatively, you can use input_message_content to send a message with the specified content instead of the location.

https://core.telegram.org/bots/api#inlinequeryresultlocation

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InlineQueryResultVenue

class aiogram.types.inline_query_result.**InlineQueryResultVenue**(*, *id: String*, *latitude: Float*, *longitude: Float*, *title: String*, *address: String*, *foursquare_id: Optional[String] = None*, *foursquare_type: Optional[String] = None*, *google_place_id: Optional[String] = None*, *google_place_type: Optional[String] = None*, *reply_markup: Optional[InlineKeyboardMarkup] = None*, *input_message_content: Optional[InputMessageContent] = None*, *thumb_url: Optional[String] = None*, *thumb_width: Optional[Integer] = None*, *thumb_height: Optional[Integer] = None*)

Bases: *InlineQueryResult*

Represents a venue. By default, the venue will be sent by the user.

Alternatively, you can use input_message_content to send a message with the specified content instead of the venue.

Note: This will only work in Telegram versions released after 9 April, 2016. Older clients will ignore them.

https://core.telegram.org/bots/api#inlinequeryresultvenue

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InlineQueryResultContact

class aiogram.types.inline_query_result.**InlineQueryResultContact**(*, *id: String*, *phone_number: String*, *first_name: String*, *last_name: Optional[String] = None*, *reply_markup: Optional[InlineKeyboardMarkup] = None*, *input_message_content: Optional[InputMessageContent] = None*, *thumb_url: Optional[String] = None*, *thumb_width: Optional[Integer] = None*, *thumb_height: Optional[Integer] = None*, *foursquare_type: Optional[String] = None*)

Bases: *InlineQueryResult*

Represents a contact with a phone number.

By default, this contact will be sent by the user. Alternatively, you can use input_message_content to send a message with the specified content instead of the contact.

Note: This will only work in Telegram versions released after 9 April, 2016. Older clients will ignore them.

https://core.telegram.org/bots/api#inlinequeryresultcontact

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InlineQueryResultGame

class aiogram.types.inline_query_result.**InlineQueryResultGame**(*, *id: String*, *game_short_name: String*, *reply_markup: Optional[*InlineKeyboardMarkup*] = None*)

Bases: *InlineQueryResult*

Represents a Game.

Note: This will only work in Telegram versions released after October 1, 2016. Older clients will not display any inline results if a game result is among them.

https://core.telegram.org/bots/api#inlinequeryresultgame

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InlineQueryResultCachedPhoto

class aiogram.types.inline_query_result.**InlineQueryResultCachedPhoto**(*, *id: String*, *photo_file_id: String*, *title: Optional[String] = None*, *description: Optional[String] = None*, *caption: Optional[String] = None*, *parse_mode: Optional[String] = None*, *caption_entities: Optional[List[MessageEntity]] = None*, *reply_markup: Optional[InlineKeyboardMarkup] = None*, *input_message_content: Optional[InputMessageContent] = None*)

Bases: *InlineQueryResult*

Represents a link to a photo stored on the Telegram servers.

By default, this photo will be sent by the user with an optional caption. Alternatively, you can use input_message_content to send a message with the specified content instead of the photo.

https://core.telegram.org/bots/api#inlinequeryresultcachedphoto

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InlineQueryResultCachedGif

class aiogram.types.inline_query_result.**InlineQueryResultCachedGif**(*, *id: String*, *gif_file_id: String*, *title: Optional[String] = None*, *caption: Optional[String] = None*, *parse_mode: Optional[String] = None*, *caption_entities: Optional[List[MessageEntity]] = None*, *reply_markup: Optional[InlineKeyboardMarkup] = None*, *input_message_content: Optional[InputMessageContent] = None*)

Bases: *InlineQueryResult*

Represents a link to an animated GIF file stored on the Telegram servers.

By default, this animated GIF file will be sent by the user with an optional caption. Alternatively, you can use input_message_content to send a message with specified content instead of the animation.

https://core.telegram.org/bots/api#inlinequeryresultcachedgif

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InlineQueryResultCachedMpeg4Gif

class aiogram.types.inline_query_result.**InlineQueryResultCachedMpeg4Gif**(*, *id: String*, *mpeg4_file_id: String*, *title: Optional[String] = None*, *caption: Optional[String] = None*, *parse_mode: Optional[String] = None*, *caption_entities: Optional[List[MessageEntity]] = None*, *reply_markup: Optional[InlineKeyboardMarkup] = None*, *input_message_content: Optional[InputMessageContent] = None*)

Bases: *InlineQueryResult*

Represents a link to a video animation (H.264/MPEG-4 AVC video without sound) stored on the Telegram servers.

By default, this animated MPEG-4 file will be sent by the user with an optional caption. Alternatively, you can use input_message_content to send a message with the specified content instead of the animation.

https://core.telegram.org/bots/api#inlinequeryresultcachedmpeg4gif

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InlineQueryResultCachedSticker

class aiogram.types.inline_query_result.**InlineQueryResultCachedSticker**(*, *id: String*, *sticker_file_id: String*, *reply_markup: Optional[InlineKeyboardMarkup] = None*, *input_message_content: Optional[InputMessageContent] = None*)

Bases: *InlineQueryResult*

Represents a link to a sticker stored on the Telegram servers.

By default, this sticker will be sent by the user. Alternatively, you can use input_message_content to send a message with the specified content instead of the sticker.

Note: This will only work in Telegram versions released after 9 April, 2016. Older clients will ignore them.

https://core.telegram.org/bots/api#inlinequeryresultcachedsticker

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InlineQueryResultCachedDocument

class aiogram.types.inline_query_result.**InlineQueryResultCachedDocument**(*, *id: String*, *title: String*, *document_file_id: String*, *description: Optional[String] = None*, *caption: Optional[String] = None*, *parse_mode: Optional[String] = None*, *caption_entities: Optional[List[MessageEntity]] = None*, *reply_markup: Optional[InlineKeyboardMarkup] = None*, *input_message_content: Optional[InputMessageContent] = None*)

Bases: *InlineQueryResult*

Represents a link to a file stored on the Telegram servers. By default, this file will be sent by the user with an optional caption. Alternatively, you can use input_message_content to send a message with the specified content instead of the file.

Note: This will only work in Telegram versions released after 9 April, 2016. Older clients will ignore them.

https://core.telegram.org/bots/api#inlinequeryresultcacheddocument

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InlineQueryResultCachedVideo

class aiogram.types.inline_query_result.**InlineQueryResultCachedVideo**(*, *id: String*, *video_file_id: String*, *title: String*, *description: Optional[String] = None*, *caption: Optional[String] = None*, *parse_mode: Optional[String] = None*, *caption_entities: Optional[List[MessageEntity]] = None*, *reply_markup: Optional[InlineKeyboardMarkup] = None*, *input_message_content: Optional[InputMessageContent] = None*)

Bases: *InlineQueryResult*

Represents a link to a video file stored on the Telegram servers.

By default, this video file will be sent by the user with an optional caption. Alternatively, you can use input_message_content to send a message with the specified content instead of the video.

https://core.telegram.org/bots/api#inlinequeryresultcachedvideo

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InlineQueryResultCachedVoice

class aiogram.types.inline_query_result.**InlineQueryResultCachedVoice**(*, *id: String*, *voice_file_id: String*, *title: String*, *caption: Optional[String] = None*, *parse_mode: Optional[String] = None*, *caption_entities: Optional[List[MessageEntity]] = None*, *reply_markup: Optional[InlineKeyboardMarkup] = None*, *input_message_content: Optional[InputMessageContent] = None*)

Bases: *InlineQueryResult*

Represents a link to a voice message stored on the Telegram servers.

By default, this voice message will be sent by the user. Alternatively, you can use input_message_content to send a message with the specified content instead of the voice message.

Note: This will only work in Telegram versions released after 9 April, 2016. Older clients will ignore them.

https://core.telegram.org/bots/api#inlinequeryresultcachedvoice

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InlineQueryResultCachedAudio

class aiogram.types.inline_query_result.**InlineQueryResultCachedAudio**(*, *id: String*, *audio_file_id: String*, *caption: Optional[String] = None*, *parse_mode: Optional[String] = None*, *caption_entities: Optional[List[MessageEntity]] = None*, *reply_markup: Optional[InlineKeyboardMarkup] = None*, *input_message_content: Optional[InputMessageContent] = None*)

Bases: *InlineQueryResult*

Represents a link to an mp3 audio file stored on the Telegram servers.

By default, this audio file will be sent by the user. Alternatively, you can use input_message_content to send a message with the specified content instead of the audio.

Note: This will only work in Telegram versions released after 9 April, 2016. Older clients will ignore them.

https://core.telegram.org/bots/api#inlinequeryresultcachedaudio

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InputFile

class aiogram.types.input_file.**InputFile**(*path_or_bytesio: Union[str, IOBase, Path, _WebPipe]*, *filename=None*, *conf=None*)

Bases: *TelegramObject*

This object represents the contents of a file to be uploaded. Must be posted using multipart/form-data in the usual way that files are uploaded via the browser.

Also that is not typical TelegramObject!

https://core.telegram.org/bots/api#inputfile

Parameters

- **path_or_bytesio** –
- **filename** –
- **conf** –

get_filename() → str

Get file name

Returns
name

get_file()

Get file object

Returns

classmethod from_url(*url*, *filename=None*, *chunk_size=65536*)

Download file from URL

Manually is not required action. You can send urls instead!

Parameters

- **url** – target URL
- **filename** – optional. set custom file name
- **chunk_size** –

Returns
InputFile

save(*filename*, *chunk_size=65536*)

Write file to disk

Parameters

- **filename** –
- **chunk_size** –

to_python()

Get object as JSON serializable

Returns

classmethod to_object(*data*)

Deserialize object

Parameters

- **data** –
- **conf** –

Returns

PreCheckoutQuery

class aiogram.types.pre_checkout_query.**PreCheckoutQuery**(*conf: Optional[Dict[str, Any]] = None*, ***kwargs: Any*)

Bases: *TelegramObject*

This object contains information about an incoming pre-checkout query. Your bot can offer users HTML5 games to play solo or to compete against each other in groups and one-on-one chats.

Create games via @BotFather using the /newgame command.

Please note that this kind of power requires responsibility: you will need to accept the terms for each game that your bots will be offering.

https://core.telegram.org/bots/api#precheckoutquery

Deserialize object

Parameters

- **conf** –
- **kwargs** –

Voice

class aiogram.types.voice.**Voice**(*conf: Optional[Dict[str, Any]] = None*, ***kwargs: Any*)

Bases: *TelegramObject*, *Downloadable*

This object represents a voice note.

https://core.telegram.org/bots/api#voice

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InputMessageContent

class aiogram.types.input_message_content.**InputMessageContent**(*conf: Optional[Dict[str, Any]] = None*, ***kwargs: Any*)

Bases: *TelegramObject*

This object represents the content of a message to be sent as a result of an inline query.

Telegram clients currently support the following 4 types

https://core.telegram.org/bots/api#inputmessagecontent

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InputContactMessageContent

class aiogram.types.input_message_content.**InputContactMessageContent**(*phone_number: String*, *first_name: Optional[String] = None*, *last_name: Optional[String] = None*, *vcard: Optional[String] = None*)

Bases: *InputMessageContent*

Represents the content of a contact message to be sent as the result of an inline query.

Note: This will only work in Telegram versions released after 9 April, 2016. Older clients will ignore them.

https://core.telegram.org/bots/api#inputcontactmessagecontent

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InputLocationMessageContent

class aiogram.types.input_message_content.**InputLocationMessageContent**(*latitude: Float*, *longitude: Float*, *horizontal_accuracy: Optional[Float] = None*, *live_period: Optional[Integer] = None*, *heading: Optional[Integer] = None*, *proximity_alert_radius: Optional[Integer] = None*)

Bases: *InputMessageContent*

Represents the content of a location message to be sent as the result of an inline query.

https://core.telegram.org/bots/api#inputlocationmessagecontent

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InputTextMessageContent

class aiogram.types.input_message_content.**InputTextMessageContent**(*message_text: String*, *parse_mode: Optional[String] = None*, *entities: Optional[List[MessageEntity]] = None*, *disable_web_page_preview: Optional[Boolean] = None*)

Bases: *InputMessageContent*

Represents the content of a text message to be sent as the result of an inline query.

https://core.telegram.org/bots/api#inputtextmessagecontent

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InputVenueMessageContent

class aiogram.types.input_message_content.**InputVenueMessageContent**(*latitude: Float*, *longitude: Float*, *title: String*, *address: String*, *foursquare_id: Optional[String] = None*, *foursquare_type: Optional[String] = None*, *google_place_id: Optional[String] = None*, *google_place_type: Optional[String] = None*)

Bases: *InputMessageContent*

Represents the content of a venue message to be sent as the result of an inline query.

Note: This will only work in Telegram versions released after 9 April, 2016. Older clients will ignore them.

https://core.telegram.org/bots/api#inputvenuemessagecontent

Deserialize object

Parameters

- **conf** –
- **kwargs** –

Update

class aiogram.types.update.**Update**(*conf: Optional[Dict[str, Any]] = None*, ***kwargs: Any*)

Bases: *TelegramObject*

This object represents an incoming update. At most one of the optional parameters can be present in any given update.

https://core.telegram.org/bots/api#update

Deserialize object

Parameters

- **conf** –
- **kwargs** –

AllowedUpdates

class aiogram.types.update.**AllowedUpdates**

Bases: Helper

Helper for allowed_updates parameter in getUpdates and setWebhook methods.

You can use &, + or | operators for make combination of allowed updates.

Example:

```
>>> bot.get_updates(allowed_updates=AllowedUpdates.MESSAGE + AllowedUpdates.
↪EDITED_MESSAGE)
```

PhotoSize

class aiogram.types.photo_size.**PhotoSize**(*conf: Optional[Dict[str, Any]] = None*, ***kwargs: Any*)

Bases: *TelegramObject*, *Downloadable*

This object represents one size of a photo or a file / sticker thumbnail.

https://core.telegram.org/bots/api#photosize

Deserialize object

Parameters

- **conf** –
- **kwargs** –

Venue

class aiogram.types.venue.**Venue**(*conf: Optional[Dict[str, Any]] = None*, ***kwargs: Any*)

Bases: *TelegramObject*

This object represents a venue.

https://core.telegram.org/bots/api#venue

Deserialize object

Parameters

- **conf** –
- **kwargs** –

ChosenInlineResult

class aiogram.types.chosen_inline_result.**ChosenInlineResult**(*conf: Optional[Dict[str, Any]] = None*, ***kwargs: Any*)

Bases: *TelegramObject*

Represents a result of an inline query that was chosen by the user and sent to their chat partner.

Note: It is necessary to enable inline feedback via @Botfather in order to receive these objects in updates. Your bot can accept payments from Telegram users. Please see the introduction to payments for more details on the process and how to set up payments for your bot. Please note that users will need Telegram v.4.0 or higher to use payments (released on May 18, 2017).

https://core.telegram.org/bots/api#choseninlineresult

Deserialize object

Parameters

- **conf** –
- **kwargs** –

VideoNote

class aiogram.types.video_note.**VideoNote**(*conf: Optional[Dict[str, Any]] = None*, ***kwargs: Any*)

Bases: *TelegramObject*, *Downloadable*

This object represents a video message (available in Telegram apps as of v.4.0).

https://core.telegram.org/bots/api#videonote

Deserialize object

Parameters

- **conf** –
- **kwargs** –

WebhookInfo

class aiogram.types.webhook_info.**WebhookInfo**(*conf: Optional[Dict[str, Any]] = None*, ***kwargs: Any*)

Bases: *TelegramObject*

Contains information about the current status of a webhook.

https://core.telegram.org/bots/api#webhookinfo

Deserialize object

Parameters

- **conf** –
- **kwargs** –

PassportFile

class aiogram.types.passport_file.**PassportFile**(*conf: Optional[Dict[str, Any]] = None*, ***kwargs: Any*)

Bases: *TelegramObject*

This object represents a file uploaded to Telegram Passport. Currently all Telegram Passport files are in JPEG format when decrypted and don't exceed 10MB.

https://core.telegram.org/bots/api#passportfile

Deserialize object

Parameters

- **conf** –
- **kwargs** –

ChatMember

class aiogram.types.chat_member.**ChatMember**(*conf: Optional[Dict[str, Any]] = None*, ***kwargs: Any*)

Bases: *TelegramObject*

This object contains information about one member of a chat. Currently, the following 6 types of chat members are supported:

ChatMemberOwner ChatMemberAdministrator ChatMemberMember ChatMemberRestricted ChatMemberLeft ChatMemberBanned

https://core.telegram.org/bots/api#chatmember

Deserialize object

Parameters

- **conf** –
- **kwargs** –

classmethod to_object(*data: Dict[str, Any]*, *conf: Optional[Dict[str, Any]] = None*) → Union[ChatMemberOwner, ChatMemberAdministrator, ChatMemberMember, ChatMemberRestricted, ChatMemberLeft, ChatMemberBanned]

Deserialize object

Parameters

- **data** –
- **conf** –

Returns

ChatMemberStatus

class aiogram.types.chat_member.**ChatMemberStatus**

Bases: Helper

Chat member status

ShippingOption

class aiogram.types.shipping_option.**ShippingOption**(*id: String*, *title: String*, *prices: Optional[List[LabeledPrice]] = None*)

Bases: *TelegramObject*

This object represents one shipping option.

https://core.telegram.org/bots/api#shippingoption

Deserialize object

Parameters

- **conf** –
- **kwargs** –

add(*price: LabeledPrice*)

Add price

Parameters
price –

Returns

ChatPhoto

class aiogram.types.chat_photo.**ChatPhoto**(*conf: Optional[Dict[str, Any]] = None*, ***kwargs: Any*)

Bases: *TelegramObject*

This object represents a chat photo.

https://core.telegram.org/bots/api#chatphoto

Deserialize object

Parameters

- **conf** –
- **kwargs** –

async download_small(*destination=None*, *timeout=30*, *chunk_size=65536*, *seek=True*, *make_dirs=True*)

Download file

Parameters

- **destination** – filename or instance of `io.IOBase`. For e. g. `io.BytesIO`
- **timeout** – Integer
- **chunk_size** – Integer
- **seek** – Boolean - go to start of file when downloading is finished.
- **make_dirs** – Make dirs if not exist

Returns

destination

async download_big(*destination=None*, *timeout=30*, *chunk_size=65536*, *seek=True*, *make_dirs=True*)

Download file

Parameters

- **destination** – filename or instance of `io.IOBase`. For e. g. `io.BytesIO`
- **timeout** – Integer
- **chunk_size** – Integer
- **seek** – Boolean - go to start of file when downloading is finished.
- **make_dirs** – Make dirs if not exist

Returns

destination

Contact

class aiogram.types.contact.**Contact**(*conf: Optional[Dict[str, Any]] = None*, ***kwargs: Any*)

Bases: *TelegramObject*

This object represents a phone contact.

https://core.telegram.org/bots/api#contact

Deserialize object

Parameters

- **conf** –
- **kwargs** –

Message

class aiogram.types.message.**Message**(*conf: Optional[Dict[str, Any]] = None*, ***kwargs: Any*)

Bases: *TelegramObject*

This object represents a message.

https://core.telegram.org/bots/api#message

Deserialize object

Parameters

- **conf** –
- **kwargs** –

is_forward() → bool

Check that the message is forwarded. Only *forward_date* is required to be in forwarded message.

Returns
bool

is_command() → bool

Check message text is command

Returns
bool

get_full_command() → Optional[Tuple[str, str]]

Split command and args

Returns
tuple of (command, args)

get_command(*pure=False*) → Optional[str]

Get command from message

Returns

get_args() → Optional[str]

Get arguments

Returns

parse_entities(*as_html=True*) → str

Text or caption formatted as HTML or Markdown.

Returns
str

property from_id: int

User id if sent by user or chat/channel id if sent on behalf of a channel or chat

Returns
int

property md_text: str

Text or caption formatted as markdown.

Returns
str

property html_text: str

Text or caption formatted as HTML

Returns
str

property url: str

Get URL for the message

Returns
str

link(*text*, *as_html=True*) → str

Generate URL for using in text messages with HTML or MD parse mode

Parameters

- **text** – link label
- **as_html** – generate as HTML

Returns
str

async answer(*text: String*, *parse_mode: Optional[String] = None*, *entities: Optional[List[*MessageEntity*]] = None*, *disable_web_page_preview: Optional[Boolean] = None*, *disable_notification: Optional[Boolean] = None*, *protect_content: Optional[Boolean] = None*, *allow_sending_without_reply: Optional[Boolean] = None*, *reply_markup: Optional[Union[*InlineKeyboardMarkup*,* ReplyKeyboardMarkup*,* ReplyKeyboardRemove*,* ForceReply*]] = None*, *reply: Boolean = False*) → *Message*

Answer to this message

Parameters

- **text** (base.String) – Text of the message to be sent
- **parse_mode** (typing.Optional[base.String]) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.
- **entities** (typing.Optional[typing.List[MessageEntity]]) – List of special entities that appear in message text, which can be specified instead of parse_mode
- **disable_web_page_preview** (typing.Optional[base.Boolean]) – Disables link previews for links in this message
- **disable_notification** (typing.Optional[base.Boolean]) – Sends the message silently. Users will receive a notification with no sound
- **protect_content** (typing.Optional[base.Boolean]) – Protects the contents of sent messages from forwarding and saving
- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **reply** (base.Boolean) – fill 'reply_to_message_id'

Returns
On success, the sent Message is returned

Return type
`types.Message`

async answer_photo(*photo: Union[InputFile, String], caption: Optional[String] = None, parse_mode: Optional[String] = None, caption_entities: Optional[List[MessageEntity]] = None, disable_notification: Optional[Boolean] = None, protect_content: Optional[Boolean] = None, allow_sending_without_reply: Optional[Boolean] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, reply: Boolean = False*) → *Message*

Use this method to send photos.

Source: https://core.telegram.org/bots/api#sendphoto

Parameters

- **photo** (`typing.Union[base.InputFile, base.String]`) – Photo to send
- **caption** (`typing.Optional[base.String]`) – Photo caption (may also be used when resending photos by file_id), 0-1024 characters
- **parse_mode** (`typing.Optional[base.String]`) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.
- **caption_entities** (`typing.Optional[typing.List[MessageEntity]]`) – List of special entities that appear in message text, which can be specified instead of parse_mode
- **disable_notification** (`typing.Optional[base.Boolean]`) – Sends the message silently. Users will receive a notification with no sound
- **protect_content** (`typing.Optional[base.Boolean]`) – Protects the contents of sent messages from forwarding and saving
- **allow_sending_without_reply** (`typing.Optional[base.Boolean]`) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]`) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **reply** (`base.Boolean`) – fill 'reply_to_message_id'

Returns
On success, the sent Message is returned

Return type
`types.Message`

async answer_audio(*audio: Union[InputFile, String], caption: Optional[String] = None, parse_mode: Optional[String] = None, caption_entities: Optional[List[MessageEntity]] = None, duration: Optional[Integer] = None, performer: Optional[String] = None, title: Optional[String] = None, thumb: Optional[Union[InputFile, String]] = None, disable_notification: Optional[Boolean] = None, protect_content: Optional[Boolean] = None, allow_sending_without_reply: Optional[Boolean] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, reply: Boolean = False*) → *Message*

Use this method to send audio files, if you want Telegram clients to display them in the music player. Your audio must be in the .mp3 format.

For sending voice messages, use the sendVoice method instead.

Source: https://core.telegram.org/bots/api#sendaudio

Parameters

- **audio** (`typing.Union[base.InputFile, base.String]`) – Audio file to send.
- **caption** (`typing.Optional[base.String]`) – Audio caption, 0-1024 characters after entities parsing
- **parse_mode** (`typing.Optional[base.String]`) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.
- **caption_entities** (`typing.Optional[typing.List[MessageEntity]]`) – List of special entities that appear in message text, which can be specified instead of parse_mode
- **duration** (`typing.Optional[base.Integer]`) – Duration of the audio in seconds
- **performer** (`typing.Optional[base.String]`) – Performer
- **title** (`typing.Optional[base.String]`) – Track name
- **thumb** (`typing.Union[typing.Union[base.InputFile, base.String], None]`) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320.
- **disable_notification** (`typing.Optional[base.Boolean]`) – Sends the message silently. Users will receive a notification with no sound.
- **protect_content** (`typing.Optional[base.Boolean]`) – Protects the contents of sent messages from forwarding and saving
- **allow_sending_without_reply** (`typing.Optional[base.Boolean]`) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]`) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **reply** (`base.Boolean`) – fill 'reply_to_message_id'

Returns

On success, the sent Message is returned.

Return type

`types.Message`

async answer_animation(*animation: Union[InputFile, String], duration: Optional[Integer] = None, width: Optional[Integer] = None, height: Optional[Integer] = None, thumb: Optional[Union[InputFile, String]] = None, caption: Optional[String] = None, parse_mode: Optional[String] = None, caption_entities: Optional[List[*MessageEntity*]] = None, disable_notification: Optional[Boolean] = None, protect_content: Optional[Boolean] = None, allow_sending_without_reply: Optional[Boolean] = None, reply_markup: Optional[Union[*InlineKeyboardMarkup*, *ReplyKeyboardMarkup*, *ReplyKeyboardRemove*, *ForceReply*]] = None, reply: Boolean = False*) → *Message*

Use this method to send animation files (GIF or H.264/MPEG-4 AVC video without sound).

On success, the sent Message is returned. Bots can currently send animation files of up to 50 MB in size, this limit may be changed in the future.

Source https://core.telegram.org/bots/api#sendanimation

Parameters

- **animation** (`typing.Union[base.InputFile, base.String]`) – Animation to send. Pass a file_id as String to send an animation that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get an animation from the Internet, or upload a new animation using multipart/form-data
- **duration** (`typing.Optional[base.Integer]`) – Duration of sent animation in seconds
- **width** (`typing.Optional[base.Integer]`) – Animation width
- **height** (`typing.Optional[base.Integer]`) – Animation height
- **thumb** (`typing.Union[typing.Union[base.InputFile, base.String], None]`) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail‘s width and height should not exceed 320.
- **caption** (`typing.Optional[base.String]`) – Animation caption (may also be used when resending animation by file_id), 0-1024 characters
- **parse_mode** (`typing.Optional[base.String]`) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption
- **caption_entities** (`typing.Optional[typing.List[MessageEntity]]`) – List of special entities that appear in message text, which can be specified instead of parse_mode
- **disable_notification** (`typing.Optional[base.Boolean]`) – Sends the message silently. Users will receive a notification with no sound
- **protect_content** (`typing.Optional[base.Boolean]`) – Protects the contents of sent messages from forwarding and saving
- **allow_sending_without_reply** (`typing.Optional[base.Boolean]`) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (`typing.Union[typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply], None]`) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **reply** (`base.Boolean`) – fill ‘reply_to_message_id’

Returns
On success, the sent Message is returned

Return type
`types.Message`

async answer_document(*document: Union[InputFile, String], thumb: Optional[Union[InputFile, String]] = None, caption: Optional[String] = None, parse_mode: Optional[String] = None, caption_entities: Optional[List[MessageEntity]] = None, disable_content_type_detection: Optional[Boolean] = None, disable_notification: Optional[Boolean] = None, protect_content: Optional[Boolean] = None, allow_sending_without_reply: Optional[Boolean] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, reply: Boolean = False*) → *Message*

Use this method to send general files. On success, the sent Message is returned. Bots can currently send files of any type of up to 50 MB in size, this limit may be changed in the future.

Source: https://core.telegram.org/bots/api#senddocument

Parameters

- **document** (`typing.Union[base.InputFile, base.String]`) – File to send
- **thumb** (`typing.Union[base.InputFile, base.String, None]`) – Thumbnail of the file sent
- **caption** (`typing.Optional[base.String]`) – Document caption (may also be used when resending documents by file_id), 0-1024 characters
- **disable_content_type_detection** (`typing.Optional[base.Boolean]`) – Disables automatic server-side content type detection for files uploaded using multipart/form-data
- **parse_mode** (`typing.Optional[base.String]`) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.
- **caption_entities** (`typing.Optional[typing.List[MessageEntity]]`) – List of special entities that appear in message text, which can be specified instead of parse_mode
- **disable_notification** (`typing.Optional[base.Boolean]`) – Sends the message silently. Users will receive a notification with no sound
- **protect_content** (`typing.Optional[base.Boolean]`) – Protects the contents of sent messages from forwarding and saving
- **allow_sending_without_reply** (`typing.Optional[base.Boolean]`) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply], None]`) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **reply** (`typing.Optional[base.Boolean]`) – True if the message is a reply

Returns
On success, the sent Message is returned

Return type
`types.Message`

async answer_video(*video: Union[InputFile, String], duration: Optional[Integer] = None, width: Optional[Integer] = None, height: Optional[Integer] = None, thumb: Optional[Union[InputFile, String]] = None, caption: Optional[String] = None, parse_mode: Optional[String] = None, caption_entities: Optional[List[*MessageEntity*]] = None, supports_streaming: Optional[Boolean] = None, disable_notification: Optional[Boolean] = None, protect_content: Optional[Boolean] = None, allow_sending_without_reply: Optional[Boolean] = None, reply_markup: Optional[Union[*InlineKeyboardMarkup*, *ReplyKeyboardMarkup*, *ReplyKeyboardRemove*, *ForceReply*]] = None, reply: Boolean = False*) → *Message*

Use this method to send video files, Telegram clients support mp4 videos (other formats may be sent as Document).

Source: https://core.telegram.org/bots/api#sendvideo

Parameters

- **video** (`typing.Union[base.InputFile, base.String]`) – Video to send.
- **duration** (`typing.Optional[base.Integer]`) – Duration of sent video in seconds
- **width** (`typing.Optional[base.Integer]`) – Video width
- **height** (`typing.Optional[base.Integer]`) – Video height
- **thumb** (`typing.Union[base.InputFile, base.String, None]`) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320.
- **caption** (`typing.Optional[base.String]`) – Video caption (may also be used when resending videos by file_id), 0-1024 characters after entities parsing
- **parse_mode** (`typing.Optional[base.String]`) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption
- **caption_entities** (`typing.Optional[typing.List[MessageEntity]]`) – List of special entities that appear in message text, which can be specified instead of parse_mode
- **supports_streaming** (`typing.Optional[base.Boolean]`) – Pass True, if the uploaded video is suitable for streaming
- **disable_notification** (`typing.Optional[base.Boolean]`) – Sends the message silently. Users will receive a notification with no sound.
- **protect_content** (`typing.Optional[base.Boolean]`) – Protects the contents of sent messages from forwarding and saving
- **allow_sending_without_reply** (`typing.Optional[base.Boolean]`) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]`) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **reply** (`base.Boolean`) – fill 'reply_to_message_id'

Returns
On success, the sent Message is returned.

Return type
`types.Message`

async answer_voice(*voice: Union[InputFile, String], caption: Optional[String] = None, parse_mode: Optional[String] = None, caption_entities: Optional[List[MessageEntity]] = None, duration: Optional[Integer] = None, disable_notification: Optional[Boolean] = None, protect_content: Optional[Boolean] = None, allow_sending_without_reply: Optional[Boolean] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, reply: Boolean = False*) → *Message*

Use this method to send audio files, if you want Telegram clients to display the file as a playable voice message.

For this to work, your audio must be in an .ogg file encoded with OPUS (other formats may be sent as Audio or Document).

Source: https://core.telegram.org/bots/api#sendvoice

Parameters

- **voice** (`typing.Union[base.InputFile, base.String]`) – Audio file to send.
- **caption** (`typing.Optional[base.String]`) – Voice message caption, 0-1024 characters after entities parsing
- **parse_mode** (`typing.Optional[base.String]`) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption
- **caption_entities** (`typing.Optional[typing.List[MessageEntity]]`) – List of special entities that appear in message text, which can be specified instead of parse_mode
- **duration** (`typing.Optional[base.Integer]`) – Duration of the voice message in seconds
- **disable_notification** (`typing.Optional[base.Boolean]`) – Sends the message silently. Users will receive a notification with no sound.
- **protect_content** (`typing.Optional[base.Boolean]`) – Protects the contents of sent messages from forwarding and saving
- **allow_sending_without_reply** (`typing.Optional[base.Boolean]`) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]`) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **reply** (`base.Boolean`) – fill 'reply_to_message_id'

Returns
On success, the sent Message is returned.

Return type
`types.Message`

async answer_video_note(*video_note: Union[InputFile, String], duration: Optional[Integer] = None, length: Optional[Integer] = None, thumb: Optional[Union[InputFile, String]] = None, disable_notification: Optional[Boolean] = None, protect_content: Optional[Boolean] = None, allow_sending_without_reply: Optional[Boolean] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, reply: Boolean = False*) → *Message*

As of v.4.0, Telegram clients support rounded square mp4 videos of up to 1 minute long. Use this method to send video messages.

Source: https://core.telegram.org/bots/api#sendvideonote

Parameters

- **video_note** (typing.Union[base.InputFile, base.String]) – Video note to send.
- **duration** (typing.Optional[base.Integer]) – Duration of sent video in seconds
- **length** (typing.Optional[base.Integer]) – Video width and height
- **thumb** (typing.Union[typing.Union[base.InputFile, base.String], None]) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320.
- **disable_notification** (typing.Optional[base.Boolean]) – Sends the message silently. Users will receive a notification with no sound.
- **protect_content** (typing.Optional[base.Boolean]) – Protects the contents of sent messages from forwarding and saving
- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **reply** (base.Boolean) – fill 'reply_to_message_id'

Returns
On success, the sent Message is returned.

Return type
types.Message

async answer_media_group(*media: Union[MediaGroup, List], disable_notification: Optional[Boolean] = None, protect_content: Optional[Boolean] = None, allow_sending_without_reply: Optional[Boolean] = None, reply: Boolean = False*) → List[*Message*]

Use this method to send a group of photos, videos, documents or audios as an album. Documents and audio files can be only group in an album with messages of the same type. On success, an array of Messages that were sent is returned.

Source: https://core.telegram.org/bots/api#sendmediagroup

Parameters

- **media** (typing.Union[types.MediaGroup, typing.List]) – A JSON-serialized array describing photos and videos to be sent

- **disable_notification** (typing.Optional[base.Boolean]) – Sends the message silently. Users will receive a notification with no sound.
- **protect_content** (typing.Optional[base.Boolean]) – Protects the contents of sent messages from forwarding and saving
- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply** (base.Boolean) – fill 'reply_to_message_id'

Returns
On success, an array of the sent Messages is returned.

Return type
List[types.Message]

async answer_location(*latitude: Float, longitude: Float, live_period: Optional[Integer] = None, disable_notification: Optional[Boolean] = None, protect_content: Optional[Boolean] = None, allow_sending_without_reply: Optional[Boolean] = None, horizontal_accuracy: Optional[Float] = None, heading: Optional[Integer] = None, proximity_alert_radius: Optional[Integer] = None, reply_markup: Optional[Union[*InlineKeyboardMarkup*,* ReplyKeyboardMarkup*,* ReplyKeyboardRemove*,* ForceReply*]] = None, reply: Boolean = False*) → *Message*

Use this method to send point on the map.

Source: https://core.telegram.org/bots/api#sendlocation

Parameters

- **latitude** (base.Float) – Latitude of the location
- **longitude** (base.Float) – Longitude of the location
- **horizontal_accuracy** (typing.Optional[base.Float]) – The radius of uncertainty for the location, measured in meters; 0-1500
- **live_period** (typing.Optional[base.Integer]) – Period in seconds for which the location will be updated
- **heading** (typing.Optional[base.Integer]) – For live locations, a direction in which the user is moving, in degrees. Must be between 1 and 360 if specified.
- **proximity_alert_radius** (typing.Optional[base.Integer]) – For live locations, a maximum distance for proximity alerts about approaching another chat member, in meters. Must be between 1 and 100000 if specified.
- **disable_notification** (typing.Optional[base.Boolean]) – Sends the message silently. Users will receive a notification with no sound.
- **protect_content** (typing.Optional[base.Boolean]) – Protects the contents of sent messages from forwarding and saving
- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user

- **reply** (base.Boolean) – fill 'reply_to_message_id'

Returns
On success, the sent Message is returned.

Return type
types.Message

async answer_venue(*latitude: Float, longitude: Float, title: String, address: String, foursquare_id: Optional[String] = None, foursquare_type: Optional[String] = None, google_place_id: Optional[String] = None, google_place_type: Optional[String] = None, disable_notification: Optional[Boolean] = None, protect_content: Optional[Boolean] = None, allow_sending_without_reply: Optional[Boolean] = None, reply_markup: Optional[Union[*InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply*]] = None, reply: Boolean = False*) → *Message*

Use this method to send information about a venue.

Source: https://core.telegram.org/bots/api#sendvenue

Parameters

- **latitude** (base.Float) – Latitude of the venue
- **longitude** (base.Float) – Longitude of the venue
- **title** (base.String) – Name of the venue
- **address** (base.String) – Address of the venue
- **foursquare_id** (typing.Optional[base.String]) – Foursquare identifier of the venue
- **foursquare_type** (typing.Optional[base.String]) – Foursquare type of the venue, if known
- **google_place_id** (typing.Optional[base.String]) – Google Places identifier of the venue
- **google_place_type** (typing.Optional[base.String]) – Google Places type of the venue. See supported types: https://developers.google.com/places/web-service/supported_types
- **disable_notification** (typing.Optional[base.Boolean]) – Sends the message silently. Users will receive a notification with no sound
- **protect_content** (typing.Optional[base.Boolean]) – Protects the contents of sent messages from forwarding and saving
- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **reply** (base.Boolean) – fill 'reply_to_message_id'

Returns
On success, the sent Message is returned.

Return type
`types.Message`

async answer_contact(*phone_number: String, first_name: String, last_name: Optional[String] = None, disable_notification: Optional[Boolean] = None, protect_content: Optional[Boolean] = None, allow_sending_without_reply: Optional[Boolean] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, reply: Boolean = False*) → *Message*

Use this method to send phone contacts.

Source: https://core.telegram.org/bots/api#sendcontact

Parameters

- **phone_number** (`base.String`) – Contact's phone number
- **first_name** (`base.String`) – Contact's first name
- **last_name** (`typing.Optional[base.String]`) – Contact's last name
- **disable_notification** (`typing.Optional[base.Boolean]`) – Sends the message silently. Users will receive a notification with no sound.
- **protect_content** (`typing.Optional[base.Boolean]`) – Protects the contents of sent messages from forwarding and saving
- **allow_sending_without_reply** (`typing.Optional[base.Boolean]`) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]`) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **reply** (`base.Boolean`) – fill 'reply_to_message_id'

Returns
On success, the sent Message is returned.

Return type
`types.Message`

async answer_sticker(*sticker: Union[InputFile, String], disable_notification: Optional[Boolean] = None, protect_content: Optional[Boolean] = None, allow_sending_without_reply: Optional[Boolean] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, reply: Boolean = False*) → *Message*

Use this method to send .webp stickers.

Source: https://core.telegram.org/bots/api#sendsticker

Parameters

- **sticker** (`typing.Union[base.InputFile, base.String]`) – Sticker to send.
- **disable_notification** (`typing.Optional[base.Boolean]`) – Sends the message silently. Users will receive a notification with no sound.
- **protect_content** (`typing.Optional[base.Boolean]`) – Protects the contents of sent messages from forwarding and saving

- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **reply** (base.Boolean) – fill 'reply_to_message_id'

Returns
On success, the sent Message is returned.

Return type
types.Message

async answer_poll(*question: String, options: List[String], is_anonymous: Optional[Boolean] = None, type: Optional[String] = None, allows_multiple_answers: Optional[Boolean] = None, correct_option_id: Optional[Integer] = None, explanation: Optional[String] = None, explanation_parse_mode: Optional[String] = None, explanation_entities: Optional[List[MessageEntity]] = None, open_period: Optional[Integer] = None, close_date: Optional[Union[Integer, datetime, timedelta]] = None, is_closed: Optional[Boolean] = None, disable_notification: Optional[Boolean] = None, protect_content: Optional[Boolean] = None, allow_sending_without_reply: Optional[Boolean] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, reply: Boolean = False*) → *Message*

Use this method to send a native poll. On success, the sent Message is returned.

Source: https://core.telegram.org/bots/api#sendpoll

Parameters

- **question** (base.String) – Poll question, 1-255 characters
- **options** (typing.List[base.String]) – List of answer options, 2-10 strings 1-100 characters each
- **is_anonymous** (typing.Optional[base.Boolean]) – True, if the poll needs to be anonymous, defaults to True
- **type** (typing.Optional[base.String]) – Poll type, "quiz" or "regular", defaults to "regular"
- **allows_multiple_answers** (typing.Optional[base.Boolean]) – True, if the poll allows multiple answers, ignored for polls in quiz mode, defaults to False
- **correct_option_id** (typing.Optional[base.Integer]) – 0-based identifier of the correct answer option, required for polls in quiz mode
- **explanation** (typing.Optional[base.String]) – Text that is shown when a user chooses an incorrect answer or taps on the lamp icon in a quiz-style poll, 0-200 characters with at most 2 line feeds after entities parsing
- **explanation_parse_mode** (typing.Optional[base.String]) – Mode for parsing entities in the explanation. See formatting options for more details.
- **explanation_entities** (typing.Optional[typing.List[MessageEntity]]) – List of special entities that appear in message text, which can be specified instead of parse_mode

- **open_period** (typing.Optional[base.Integer]) – Amount of time in seconds the poll will be active after creation, 5-600. Can't be used together with close_date.
- **close_date** (typing.Union[base.Integer, datetime.datetime, datetime.timedelta, None]) – Point in time (Unix timestamp) when the poll will be automatically closed. Must be at least 5 and no more than 600 seconds in the future. Can't be used together with open_period.
- **is_closed** (typing.Optional[base.Boolean]) – Pass True, if the poll needs to be immediately closed
- **disable_notification** (typing.Optional[Boolean]) – Sends the message silently. Users will receive a notification with no sound.
- **protect_content** (typing.Optional[base.Boolean]) – Protects the contents of sent messages from forwarding and saving
- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **reply** (base.Boolean) – fill 'reply_to_message_id'

Returns
On success, the sent Message is returned

Return type
types.Message

async answer_dice(*emoji: Optional[String] = None*, *disable_notification: Optional[Boolean] = None*, *protect_content: Optional[Boolean] = None*, *allow_sending_without_reply: Optional[Boolean] = None*, *reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None*, *reply: Boolean = False*) → *Message*

Use this method to send an animated emoji that will display a random value. On success, the sent Message is returned.

Source: https://core.telegram.org/bots/api#senddice

Parameters

- **emoji** (typing.Optional[base.String]) – Emoji on which the dice throw animation is based. Currently, must be one of "", "", "", "", or "". Dice can have values 1-6 for "" and "", values 1-5 for "" and "", and values 1-64 for "". Defaults to ""
- **disable_notification** (typing.Optional[base.Boolean]) – Sends the message silently. Users will receive a notification with no sound
- **protect_content** (typing.Optional[base.Boolean]) – Protects the contents of sent messages from forwarding and saving
- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]) – Additional interface options. A JSON-serialized object for an inline keyboard,

custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user

- **reply** (base.Boolean) – fill 'reply_to_message_id'

Returns
On success, the sent Message is returned.

Return type
types.Message

async answer_chat_action(*action: String*, *message_thread_id: Optional[Integer] = None*) → Boolean

Use this method when you need to tell the user that something is happening on the bot's side. The status is set for 5 seconds or less (when a message arrives from your bot, Telegram clients clear its typing status).

We only recommend using this method when a response from the bot will take a noticeable amount of time to arrive.

Source: https://core.telegram.org/bots/api#sendchataction

Parameters

- **action** (base.String) – Type of action to broadcast
- **message_thread_id** (typing.Optional[base.Integer]) – Unique identifier for the target message thread; supergroups only

Returns
Returns True on success

Return type
base.Boolean

async reply(*text: String*, *parse_mode: Optional[String] = None*, *entities: Optional[List[MessageEntity]] = None*, *disable_web_page_preview: Optional[Boolean] = None*, *disable_notification: Optional[Boolean] = None*, *protect_content: Optional[Boolean] = None*, *allow_sending_without_reply: Optional[Boolean] = None*, *reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None*, *reply: Boolean = True*) → *Message*

Reply to this message

Parameters

- **text** (base.String) – Text of the message to be sent
- **parse_mode** (typing.Optional[base.String]) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.
- **entities** (typing.Optional[typing.List[MessageEntity]]) – List of special entities that appear in message text, which can be specified instead of parse_mode
- **disable_web_page_preview** (typing.Optional[base.Boolean]) – Disables link previews for links in this message
- **disable_notification** (typing.Optional[base.Boolean]) – Sends the message silently. Users will receive a notification with no sound
- **protect_content** (typing.Optional[base.Boolean]) – Protects the contents of sent messages from forwarding and saving
- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found

- **reply_markup** (typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **reply** (base.Boolean) – fill 'reply_to_message_id'

Returns
On success, the sent Message is returned

Return type
types.Message

async reply_photo(*photo: Union[InputFile, String], caption: Optional[String] = None, parse_mode: Optional[String] = None, caption_entities: Optional[List[MessageEntity]] = None, disable_notification: Optional[Boolean] = None, protect_content: Optional[Boolean] = None, allow_sending_without_reply: Optional[Boolean] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, reply: Boolean = True*) → *Message*

Use this method to send photos.

Source: https://core.telegram.org/bots/api#sendphoto

Parameters

- **photo** (typing.Union[base.InputFile, base.String]) – Photo to send
- **caption** (typing.Optional[base.String]) – Photo caption (may also be used when resending photos by file_id), 0-1024 characters
- **parse_mode** (typing.Optional[base.String]) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.
- **caption_entities** (typing.Optional[typing.List[MessageEntity]]) – List of special entities that appear in message text, which can be specified instead of parse_mode
- **disable_notification** (typing.Optional[base.Boolean]) – Sends the message silently. Users will receive a notification with no sound
- **protect_content** (typing.Optional[base.Boolean]) – Protects the contents of sent messages from forwarding and saving
- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **reply** (base.Boolean) – fill 'reply_to_message_id'

Returns
On success, the sent Message is returned

Return type
types.Message

async reply_audio(*audio: Union[InputFile, String], caption: Optional[String] = None, parse_mode: Optional[String] = None, caption_entities: Optional[List[MessageEntity]] = None, duration: Optional[Integer] = None, performer: Optional[String] = None, title: Optional[String] = None, thumb: Optional[Union[InputFile, String]] = None, disable_notification: Optional[Boolean] = None, protect_content: Optional[Boolean] = None, allow_sending_without_reply: Optional[Boolean] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, reply: Boolean = True*) → *Message*

Use this method to send audio files, if you want Telegram clients to display them in the music player. Your audio must be in the .mp3 format.

For sending voice messages, use the sendVoice method instead.

Source: https://core.telegram.org/bots/api#sendaudio

Parameters

- **audio** (typing.Union[base.InputFile, base.String]) – Audio file to send.
- **caption** (typing.Optional[base.String]) – Audio caption, 0-1024 characters after entities parsing
- **parse_mode** (typing.Optional[base.String]) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.
- **caption_entities** (typing.Optional[typing.List[MessageEntity]]) – List of special entities that appear in message text, which can be specified instead of parse_mode
- **duration** (typing.Optional[base.Integer]) – Duration of the audio in seconds
- **performer** (typing.Optional[base.String]) – Performer
- **title** (typing.Optional[base.String]) – Track name
- **thumb** (typing.Union[typing.Union[base.InputFile, base.String], None]) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320.
- **disable_notification** (typing.Optional[base.Boolean]) – Sends the message silently. Users will receive a notification with no sound.
- **protect_content** (typing.Optional[base.Boolean]) – Protects the contents of sent messages from forwarding and saving
- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **reply** (base.Boolean) – fill 'reply_to_message_id'

Returns

On success, the sent Message is returned.

Return type

types.Message

async reply_animation(*animation: Union[InputFile, String], duration: Optional[Integer] = None, width: Optional[Integer] = None, height: Optional[Integer] = None, thumb: Optional[Union[InputFile, String]] = None, caption: Optional[String] = None, parse_mode: Optional[String] = None, caption_entities: Optional[List[*MessageEntity*]] = None, disable_notification: Optional[Boolean] = None, protect_content: Optional[Boolean] = None, allow_sending_without_reply: Optional[Boolean] = None, reply_markup: Optional[Union[*InlineKeyboardMarkup*, *ReplyKeyboardMarkup*, *ReplyKeyboardRemove*, *ForceReply*]] = None, reply: Boolean = True*) → *Message*

Use this method to send animation files (GIF or H.264/MPEG-4 AVC video without sound).

On success, the sent Message is returned. Bots can currently send animation files of up to 50 MB in size, this limit may be changed in the future.

Source https://core.telegram.org/bots/api#sendanimation

Parameters

- **animation** (typing.Union[base.InputFile, base.String]) – Animation to send. Pass a file_id as String to send an animation that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get an animation from the Internet, or upload a new animation using multipart/form-data
- **duration** (typing.Optional[base.Integer]) – Duration of sent animation in seconds
- **width** (typing.Optional[base.Integer]) – Animation width
- **height** (typing.Optional[base.Integer]) – Animation height
- **thumb** (typing.Union[typing.Union[base.InputFile, base.String], None]) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320.
- **caption** (typing.Optional[base.String]) – Animation caption (may also be used when resending animation by file_id), 0-1024 characters
- **parse_mode** (typing.Optional[base.String]) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption
- **caption_entities** (typing.Optional[typing.List[MessageEntity]]) – List of special entities that appear in message text, which can be specified instead of parse_mode
- **disable_notification** (typing.Optional[base.Boolean]) – Sends the message silently. Users will receive a notification with no sound
- **protect_content** (typing.Optional[base.Boolean]) – Protects the contents of sent messages from forwarding and saving
- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (typing.Union[typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply], None]) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **reply** (base.Boolean) – fill 'reply_to_message_id'

Returns
On success, the sent Message is returned

Return type
`types.Message`

async reply_document(*document: Union[InputFile, String], thumb: Optional[Union[InputFile, String]] = None, caption: Optional[String] = None, parse_mode: Optional[String] = None, caption_entities: Optional[List[MessageEntity]] = None, disable_content_type_detection: Optional[Boolean] = None, disable_notification: Optional[Boolean] = None, protect_content: Optional[Boolean] = None, allow_sending_without_reply: Optional[Boolean] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, reply: Boolean = True*) → *Message*

Use this method to send general files. On success, the sent Message is returned. Bots can currently send files of any type of up to 50 MB in size, this limit may be changed in the future.

Source: https://core.telegram.org/bots/api#senddocument

Parameters

- **document** (`typing.Union[base.InputFile, base.String]`) – File to send
- **thumb** (`typing.Union[base.InputFile, base.String, None]`) – Thumbnail of the file sent
- **caption** (`typing.Optional[base.String]`) – Document caption (may also be used when resending documents by file_id), 0-1024 characters
- **disable_content_type_detection** (`typing.Optional[base.Boolean]`) – Disables automatic server-side content type detection for files uploaded using multipart/form-data
- **parse_mode** (`typing.Optional[base.String]`) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.
- **caption_entities** (`typing.Optional[typing.List[MessageEntity]]`) – List of special entities that appear in message text, which can be specified instead of parse_mode
- **disable_notification** (`typing.Optional[base.Boolean]`) – Sends the message silently. Users will receive a notification with no sound
- **protect_content** (`typing.Optional[base.Boolean]`) – Protects the contents of sent messages from forwarding and saving
- **allow_sending_without_reply** (`typing.Optional[base.Boolean]`) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply], None]`) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **reply** (`typing.Optional[base.Boolean]`) – True if the message is a reply

Returns
On success, the sent Message is returned

Return type
`types.Message`

async reply_video(*video: Union[InputFile, String], duration: Optional[Integer] = None, width: Optional[Integer] = None, height: Optional[Integer] = None, thumb: Optional[Union[InputFile, String]] = None, caption: Optional[String] = None, parse_mode: Optional[String] = None, caption_entities: Optional[List[MessageEntity]] = None, supports_streaming: Optional[Boolean] = None, disable_notification: Optional[Boolean] = None, protect_content: Optional[Boolean] = None, allow_sending_without_reply: Optional[Boolean] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, reply: Boolean = True*) → *Message*

Use this method to send video files, Telegram clients support mp4 videos (other formats may be sent as Document).

Source: https://core.telegram.org/bots/api#sendvideo

Parameters

- **video** (`typing.Union[base.InputFile, base.String]`) – Video to send.
- **duration** (`typing.Optional[base.Integer]`) – Duration of sent video in seconds
- **width** (`typing.Optional[base.Integer]`) – Video width
- **height** (`typing.Optional[base.Integer]`) – Video height
- **thumb** (`typing.Union[base.InputFile, base.String, None]`) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320.
- **caption** (`typing.Optional[base.String]`) – Video caption (may also be used when resending videos by file_id), 0-1024 characters after entities parsing
- **parse_mode** (`typing.Optional[base.String]`) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption
- **caption_entities** (`typing.Optional[typing.List[MessageEntity]]`) – List of special entities that appear in message text, which can be specified instead of parse_mode
- **supports_streaming** (`typing.Optional[base.Boolean]`) – Pass True, if the uploaded video is suitable for streaming
- **disable_notification** (`typing.Optional[base.Boolean]`) – Sends the message silently. Users will receive a notification with no sound.
- **protect_content** (`typing.Optional[base.Boolean]`) – Protects the contents of sent messages from forwarding and saving
- **allow_sending_without_reply** (`typing.Optional[base.Boolean]`) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]`) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **reply** (`base.Boolean`) – fill 'reply_to_message_id'

Returns

On success, the sent Message is returned.

Return type

`types.Message`

async reply_voice(*voice: Union[InputFile, String], caption: Optional[String] = None, parse_mode: Optional[String] = None, caption_entities: Optional[List[MessageEntity]] = None, duration: Optional[Integer] = None, disable_notification: Optional[Boolean] = None, protect_content: Optional[Boolean] = None, allow_sending_without_reply: Optional[Boolean] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, reply: Boolean = True*) → *Message*

Use this method to send audio files, if you want Telegram clients to display the file as a playable voice message.

For this to work, your audio must be in an .ogg file encoded with OPUS (other formats may be sent as Audio or Document).

Source: https://core.telegram.org/bots/api#sendvoice

Parameters

- **voice** (`typing.Union[base.InputFile, base.String]`) – Audio file to send.
- **caption** (`typing.Optional[base.String]`) – Voice message caption, 0-1024 characters after entities parsing
- **parse_mode** (`typing.Optional[base.String]`) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption
- **caption_entities** (`typing.Optional[typing.List[MessageEntity]]`) – List of special entities that appear in message text, which can be specified instead of parse_mode
- **duration** (`typing.Optional[base.Integer]`) – Duration of the voice message in seconds
- **disable_notification** (`typing.Optional[base.Boolean]`) – Sends the message silently. Users will receive a notification with no sound.
- **protect_content** (`typing.Optional[base.Boolean]`) – Protects the contents of sent messages from forwarding and saving
- **allow_sending_without_reply** (`typing.Optional[base.Boolean]`) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]`) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **reply** (`base.Boolean`) – fill 'reply_to_message_id'

Returns

On success, the sent Message is returned.

Return type

`types.Message`

async reply_video_note(*video_note: Union[InputFile, String], duration: Optional[Integer] = None, length: Optional[Integer] = None, thumb: Optional[Union[InputFile, String]] = None, disable_notification: Optional[Boolean] = None, protect_content: Optional[Boolean] = None, allow_sending_without_reply: Optional[Boolean] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, reply: Boolean = True*) → *Message*

As of v.4.0, Telegram clients support rounded square mp4 videos of up to 1 minute long. Use this method to send video messages.

Source: https://core.telegram.org/bots/api#sendvideonote

Parameters

- **video_note** (typing.Union[base.InputFile, base.String]) – Video note to send.
- **duration** (typing.Optional[base.Integer]) – Duration of sent video in seconds
- **length** (typing.Optional[base.Integer]) – Video width and height
- **thumb** (typing.Union[typing.Union[base.InputFile, base.String], None]) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320.
- **disable_notification** (typing.Optional[base.Boolean]) – Sends the message silently. Users will receive a notification with no sound.
- **protect_content** (typing.Optional[base.Boolean]) – Protects the contents of sent messages from forwarding and saving
- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (:obj:`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None] `) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **reply** (base.Boolean) – fill 'reply_to_message_id'

Returns
On success, the sent Message is returned.

Return type
types.Message

async reply_media_group(*media: Union[MediaGroup, List], disable_notification: Optional[Boolean] = None, protect_content: Optional[Boolean] = None, allow_sending_without_reply: Optional[Boolean] = None, reply: Boolean = True*) → List[*Message*]

Use this method to send a group of photos, videos, documents or audios as an album. Documents and audio files can be only group in an album with messages of the same type. On success, an array of Messages that were sent is returned.

Source: https://core.telegram.org/bots/api#sendmediagroup

Parameters

- **media** (typing.Union[types.MediaGroup, typing.List]) – A JSON-serialized array describing photos and videos to be sent
- **disable_notification** (typing.Optional[base.Boolean]) – Sends the message silently. Users will receive a notification with no sound.
- **protect_content** (typing.Optional[base.Boolean]) – Protects the contents of sent messages from forwarding and saving
- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found

- **reply** (base.Boolean) – fill 'reply_to_message_id'

Returns
On success, an array of the sent Messages is returned.

Return type
List[types.Message]

async reply_location(*latitude: Float, longitude: Float, live_period: Optional[Integer] = None, disable_notification: Optional[Boolean] = None, protect_content: Optional[Boolean] = None, horizontal_accuracy: Optional[Float] = None, heading: Optional[Integer] = None, proximity_alert_radius: Optional[Integer] = None, reply_markup: Optional[Union[*InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply*]] = None, reply: Boolean = True*) → *Message*

Use this method to send point on the map.

Source: https://core.telegram.org/bots/api#sendlocation

Parameters

- **latitude** (base.Float) – Latitude of the location
- **longitude** (base.Float) – Longitude of the location
- **horizontal_accuracy** (typing.Optional[base.Float]) – The radius of uncertainty for the location, measured in meters; 0-1500
- **live_period** (typing.Optional[base.Integer]) – Period in seconds for which the location will be updated
- **heading** (typing.Optional[base.Integer]) – For live locations, a direction in which the user is moving, in degrees. Must be between 1 and 360 if specified.
- **proximity_alert_radius** (typing.Optional[base.Integer]) – For live locations, a maximum distance for proximity alerts about approaching another chat member, in meters. Must be between 1 and 100000 if specified.
- **disable_notification** (typing.Optional[base.Boolean]) – Sends the message silently. Users will receive a notification with no sound.
- **protect_content** (typing.Optional[base.Boolean]) – Protects the contents of sent messages from forwarding and saving
- **reply_markup** (typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **reply** (base.Boolean) – fill 'reply_to_message_id'

Returns
On success, the sent Message is returned.

Return type
types.Message

async reply_venue(*latitude: Float*, *longitude: Float*, *title: String*, *address: String*, *foursquare_id: Optional[String] = None*, *foursquare_type: Optional[String] = None*, *google_place_id: Optional[String] = None*, *google_place_type: Optional[String] = None*, *disable_notification: Optional[Boolean] = None*, *protect_content: Optional[Boolean] = None*, *allow_sending_without_reply: Optional[Boolean] = None*, *reply_markup: Optional[Union[*InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply*]] = None*, *reply: Boolean = True*) → *Message*

Use this method to send information about a venue.

Source: https://core.telegram.org/bots/api#sendvenue

Parameters

- **latitude** (`base.Float`) – Latitude of the venue
- **longitude** (`base.Float`) – Longitude of the venue
- **title** (`base.String`) – Name of the venue
- **address** (`base.String`) – Address of the venue
- **foursquare_id** (`typing.Optional[base.String]`) – Foursquare identifier of the venue
- **foursquare_type** (`typing.Optional[base.String]`) – Foursquare type of the venue, if known
- **google_place_id** (`typing.Optional[base.String]`) – Google Places identifier of the venue
- **google_place_type** (`typing.Optional[base.String]`) – Google Places type of the venue. See supported types: https://developers.google.com/places/web-service/supported_types
- **disable_notification** (`typing.Optional[base.Boolean]`) – Sends the message silently. Users will receive a notification with no sound
- **protect_content** (`typing.Optional[base.Boolean]`) – Protects the contents of sent messages from forwarding and saving
- **allow_sending_without_reply** (`typing.Optional[base.Boolean]`) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]`) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **reply** (`base.Boolean`) – fill 'reply_to_message_id'

Returns

On success, the sent Message is returned.

Return type

`types.Message`

async reply_contact(*phone_number: String*, *first_name: String*, *last_name: Optional[String] = None*, *disable_notification: Optional[Boolean] = None*, *protect_content: Optional[Boolean] = None*, *allow_sending_without_reply: Optional[Boolean] = None*, *reply_markup: Optional[Union[*InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply*]] = None*, *reply: Boolean = True*) → *Message*

Use this method to send phone contacts.

Source: https://core.telegram.org/bots/api#sendcontact

Parameters

- **phone_number** (`base.String`) – Contact's phone number
- **first_name** (`base.String`) – Contact's first name
- **last_name** (`typing.Optional[base.String]`) – Contact's last name
- **disable_notification** (`typing.Optional[base.Boolean]`) – Sends the message silently. Users will receive a notification with no sound.
- **protect_content** (`typing.Optional[base.Boolean]`) – Protects the contents of sent messages from forwarding and saving
- **allow_sending_without_reply** (`typing.Optional[base.Boolean]`) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]`) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **reply** (`base.Boolean`) – fill 'reply_to_message_id'

Returns

On success, the sent Message is returned.

Return type

`types.Message`

async reply_poll(*question: String, options: List[String], is_anonymous: Optional[Boolean] = None, type: Optional[String] = None, allows_multiple_answers: Optional[Boolean] = None, correct_option_id: Optional[Integer] = None, explanation: Optional[String] = None, explanation_parse_mode: Optional[String] = None, explanation_entities: Optional[List[MessageEntity]] = None, open_period: Optional[Integer] = None, close_date: Optional[Union[Integer, datetime, timedelta]] = None, is_closed: Optional[Boolean] = None, disable_notification: Optional[Boolean] = None, protect_content: Optional[Boolean] = None, allow_sending_without_reply: Optional[Boolean] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, reply: Boolean = True*) → *Message*

Use this method to send a native poll. On success, the sent Message is returned.

Source: https://core.telegram.org/bots/api#sendpoll

Parameters

- **question** (`base.String`) – Poll question, 1-255 characters
- **options** (`typing.List[base.String]`) – List of answer options, 2-10 strings 1-100 characters each
- **is_anonymous** (`typing.Optional[base.Boolean]`) – True, if the poll needs to be anonymous, defaults to True
- **type** (`typing.Optional[base.String]`) – Poll type, "quiz" or "regular", defaults to "regular"

- **allows_multiple_answers** (typing.Optional[base.Boolean]) – True, if the poll allows multiple answers, ignored for polls in quiz mode, defaults to False
- **correct_option_id** (typing.Optional[base.Integer]) – 0-based identifier of the correct answer option, required for polls in quiz mode
- **explanation** (typing.Optional[base.String]) – Text that is shown when a user chooses an incorrect answer or taps on the lamp icon in a quiz-style poll, 0-200 characters with at most 2 line feeds after entities parsing
- **explanation_parse_mode** (typing.Optional[base.String]) – Mode for parsing entities in the explanation. See formatting options for more details.
- **explanation_entities** (typing.Optional[typing.List[MessageEntity]]) – List of special entities that appear in message text, which can be specified instead of parse_mode
- **open_period** (typing.Optional[base.Integer]) – Amount of time in seconds the poll will be active after creation, 5-600. Can't be used together with close_date.
- **close_date** (typing.Union[base.Integer, datetime.datetime, datetime.timedelta, None]) – Point in time (Unix timestamp) when the poll will be automatically closed. Must be at least 5 and no more than 600 seconds in the future. Can't be used together with open_period.
- **is_closed** (typing.Optional[base.Boolean]) – Pass True, if the poll needs to be immediately closed
- **disable_notification** (typing.Optional[Boolean]) – Sends the message silently. Users will receive a notification with no sound.
- **protect_content** (typing.Optional[base.Boolean]) – Protects the contents of sent messages from forwarding and saving
- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **reply** (base.Boolean) – fill 'reply_to_message_id'

Returns

On success, the sent Message is returned

Return type

types.Message

async reply_sticker(*sticker: Union[InputFile, String], disable_notification: Optional[Boolean] = None, protect_content: Optional[Boolean] = None, allow_sending_without_reply: Optional[Boolean] = None, reply_markup: Optional[Union[*InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply*]] = None, reply: Boolean = True*) → *Message*

Use this method to send .webp stickers.

Source: https://core.telegram.org/bots/api#sendsticker

Parameters

- **sticker** (typing.Union[base.InputFile, base.String]) – Sticker to send.

- **disable_notification** (typing.Optional[base.Boolean]) – Sends the message silently. Users will receive a notification with no sound.
- **protect_content** (typing.Optional[base.Boolean]) – Protects the contents of sent messages from forwarding and saving
- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **reply** (base.Boolean) – fill 'reply_to_message_id'

Returns
On success, the sent Message is returned.

Return type
types.Message

async reply_dice(*emoji: Optional[String] = None*, *disable_notification: Optional[Boolean] = None*, *protect_content: Optional[Boolean] = None*, *allow_sending_without_reply: Optional[Boolean] = None*, *reply_markup: Optional[Union[*InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply*]] = None*, *reply: Boolean = True*) → *Message*

Use this method to send an animated emoji that will display a random value. On success, the sent Message is returned.

Source: https://core.telegram.org/bots/api#senddice

Parameters

- **emoji** (typing.Optional[base.String]) – Emoji on which the dice throw animation is based. Currently, must be one of "", "", "", "", or "". Dice can have values 1-6 for "" and "", values 1-5 for "" and "", and values 1-64 for "". Defaults to ""
- **disable_notification** (typing.Optional[base.Boolean]) – Sends the message silently. Users will receive a notification with no sound
- **protect_content** (typing.Optional[base.Boolean]) – Protects the contents of sent messages from forwarding and saving
- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **reply** (base.Boolean) – fill 'reply_to_message_id'

Returns
On success, the sent Message is returned.

Return type
types.Message

async forward(*chat_id: Union[Integer, String], message_thread_id: Optional[Integer] = None, disable_notification: Optional[Boolean] = None, protect_content: Optional[Boolean] = None*) → *Message*

Forward this message

Source: https://core.telegram.org/bots/api#forwardmessage

Parameters

- **chat_id** (`typing.Union[base.Integer, base.String]`) – Unique identifier for the target chat or username of the target channel
- **message_thread_id** (`typing.Optional[base.Integer]`) – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **disable_notification** (`typing.Optional[base.Boolean]`) – Sends the message silently. Users will receive a notification with no sound
- **protect_content** (`typing.Optional[base.Boolean]`) – Protects the contents of the forwarded message from forwarding and saving

Returns
On success, the sent Message is returned

Return type
`types.Message`

async edit_text(*text: String, parse_mode: Optional[String] = None, entities: Optional[List[*MessageEntity*]] = None, disable_web_page_preview: Optional[Boolean] = None, reply_markup: Optional[*InlineKeyboardMarkup*] = None*) → Union[*Message*, Boolean]

Use this method to edit text and game messages sent by the bot or via the bot (for inline bots).

Source: https://core.telegram.org/bots/api#editmessagetext

Parameters

- **text** (`base.String`) – New text of the message
- **parse_mode** (`typing.Optional[base.String]`) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.
- **entities** (`typing.Optional[typing.List[MessageEntity]]`) – List of special entities that appear in message text, which can be specified instead of parse_mode
- **disable_web_page_preview** (`typing.Optional[base.Boolean]`) – Disables link previews for links in this message
- **reply_markup** (`typing.Optional[types.InlineKeyboardMarkup]`) – A JSON-serialized object for an inline keyboard.

Returns
On success, if edited message is sent by the bot, the edited Message is returned, otherwise True is returned.

Return type
`typing.Union[types.Message, base.Boolean]`

async edit_caption(*caption: String, parse_mode: Optional[String] = None, caption_entities: Optional[List[*MessageEntity*]] = None, reply_markup: Optional[*InlineKeyboardMarkup*] = None*) → Union[*Message*, Boolean]

Use this method to edit captions of messages sent by the bot or via the bot (for inline bots).

Source: https://core.telegram.org/bots/api#editmessagecaption

Parameters

- **caption** (typing.Optional[base.String]) – New caption of the message
- **parse_mode** (typing.Optional[base.String]) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.
- **caption_entities** (typing.Optional[typing.List[MessageEntity]]) – List of special entities that appear in message text, which can be specified instead of parse_mode
- **reply_markup** (typing.Optional[types.InlineKeyboardMarkup]) – A JSON-serialized object for an inline keyboard

Returns

On success, if edited message is sent by the bot, the edited Message is returned, otherwise True is returned.

Return type

typing.Union[types.Message, base.Boolean]

async edit_media(*media: InputMedia, reply_markup: Optional[InlineKeyboardMarkup] = None*) → Union[*Message*, Boolean]

Use this method to edit audio, document, photo, or video messages. If a message is a part of a message album, then it can be edited only to a photo or a video. Otherwise, message type can be changed arbitrarily. When inline message is edited, new file can't be uploaded. Use previously uploaded file via its file_id or specify a URL.

On success, if the edited message was sent by the bot, the edited Message is returned, otherwise True is returned.

Source https://core.telegram.org/bots/api#editmessagemedia

Parameters

- **media** (types.InputMedia) – A JSON-serialized object for a new media content of the message
- **reply_markup** (typing.Optional[types.InlineKeyboardMarkup]) – A JSON-serialized object for a new inline keyboard

Returns

On success, if the edited message was sent by the bot, the edited Message is returned, otherwise True is returned

Return type

typing.Union[types.Message, base.Boolean]

async edit_reply_markup(*reply_markup: Optional[InlineKeyboardMarkup] = None*) → Union[*Message*, Boolean]

Use this method to edit only the reply markup of messages sent by the bot or via the bot (for inline bots).

Source: https://core.telegram.org/bots/api#editmessagereplymarkup

Parameters

reply_markup (typing.Optional[types.InlineKeyboardMarkup]) – A JSON-serialized object for an inline keyboard

Returns
On success, if edited message is sent by the bot, the edited Message is returned, otherwise True is returned.

Return type
`typing.Union[types.Message, base.Boolean]`

async delete_reply_markup() → Union[*Message*, Boolean]

Use this method to delete reply markup of messages sent by the bot or via the bot (for inline bots).

Returns
On success, if edited message is sent by the bot, the edited Message is returned, otherwise True is returned.

Return type
`typing.Union[types.Message, base.Boolean]`

async edit_live_location(*latitude: Float*, *longitude: Float*, *reply_markup: Optional[*InlineKeyboardMarkup*] = None*) → Union[*Message*, Boolean]

Use this method to edit live location messages sent by the bot or via the bot (for inline bots). A location can be edited until its live_period expires or editing is explicitly disabled by a call to stopMessageLiveLocation.

Source: https://core.telegram.org/bots/api#editmessagelivelocation

Parameters

- **latitude** (`base.Float`) – Latitude of new location
- **longitude** (`base.Float`) – Longitude of new location
- **reply_markup** (`typing.Optional[types.InlineKeyboardMarkup]`) – A JSON-serialized object for a new inline keyboard.

Returns
On success, if the edited message was sent by the bot, the edited Message is returned, otherwise True is returned.

Return type
`typing.Union[types.Message, base.Boolean]`

async stop_live_location(*reply_markup: Optional[*InlineKeyboardMarkup*] = None*) → Union[*Message*, Boolean]

Use this method to stop updating a live location message sent by the bot or via the bot (for inline bots) before live_period expires.

Source: https://core.telegram.org/bots/api#stopmessagelivelocation

Parameters
reply_markup (`typing.Optional[types.InlineKeyboardMarkup]`) – A JSON-serialized object for a new inline keyboard.

Returns
On success, if the message was sent by the bot, the sent Message is returned, otherwise True is returned.

Return type
`typing.Union[types.Message, base.Boolean]`

async delete() → Boolean

Use this method to delete a message, including service messages, with the following limitations: - A message can only be deleted if it was sent less than 48 hours ago. - Bots can delete outgoing messages in private chats, groups, and supergroups. - Bots can delete incoming messages in private chats. - Bots

granted can_post_messages permissions can delete outgoing messages in channels. - If the bot is an administrator of a group, it can delete any message there. - If the bot has can_delete_messages permission in a supergroup or a channel, it can delete any message there.

Source: https://core.telegram.org/bots/api#deletemessage

Returns
Returns True on success

Return type
`base.Boolean`

async pin(*disable_notification: Optional[Boolean] = None*) → Boolean

Use this method to add a message to the list of pinned messages in a chat. If the chat is not a private chat, the bot must be an administrator in the chat for this to work and must have the 'can_pin_messages' admin right in a supergroup or 'can_edit_messages' admin right in a channel. Returns True on success.

Source: https://core.telegram.org/bots/api#pinchatmessage

Parameters
disable_notification (`typing.Optional[base.Boolean]`) – Pass True, if it is not necessary to send a notification to all group members about the new pinned message

Returns
Returns True on success

Return type
`base.Boolean`

async unpin() → Boolean

Use this method to remove a message from the list of pinned messages in a chat. If the chat is not a private chat, the bot must be an administrator in the chat for this to work and must have the 'can_pin_messages' admin right in a supergroup or 'can_edit_messages' admin right in a channel. Returns True on success.

Source: https://core.telegram.org/bots/api#unpinchatmessage

Returns
Returns True on success

Return type
`base.Boolean`

async send_copy(*chat_id: Union[str, int], message_thread_id: Optional[Integer] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Boolean] = None, disable_web_page_preview: Optional[bool] = None, reply_to_message_id: Optional[int] = None, allow_sending_without_reply: Optional[Boolean] = None, reply_markup: Optional[Union[*InlineKeyboardMarkup, ReplyKeyboardMarkup*]] = None*) → *Message*

Send copy of current message

Parameters

- **chat_id** –
- **message_thread_id** –
- **disable_notification** –
- **protect_content** –
- **disable_web_page_preview** – for text messages only
- **reply_to_message_id** –

- **allow_sending_without_reply** –
- **reply_markup** –

Returns

ContentType

class aiogram.types.message.**ContentType**

Bases: Helper

List of message content types

WARNING: Single elements

Key
TEXT

Key
AUDIO

Key
DOCUMENT

Key
GAME

Key
PHOTO

Key
STICKER

Key
VIDEO

Key
VIDEO_NOTE

Key
VOICE

Key
CONTACT

Key
LOCATION

Key
VENUE

Key
POLL

Key
DICE

Key
NEW_CHAT_MEMBERS

Key
LEFT_CHAT_MEMBER

Key
INVOICE

Key
SUCCESSFUL_PAYMENT

Key
CONNECTED_WEBSITE

Key
MIGRATE_TO_CHAT_ID

Key
MIGRATE_FROM_CHAT_ID

Key
UNKNOWN

Key
ANY

ContentTypes

`class aiogram.types.message.ContentTypes`

Bases: `Helper`

List of message content types

WARNING: List elements.

Key
TEXT

Key
AUDIO

Key
DOCUMENT

Key
GAME

Key
PHOTO

Key
STICKER

Key
VIDEO

Key
VIDEO_NOTE

Key
VOICE

Key
CONTACT

Key
LOCATION

Key
VENUE

Key
POLL

Key
DICE

Key
NEW_CHAT_MEMBERS

Key
LEFT_CHAT_MEMBER

Key
INVOICE

Key
SUCCESSFUL_PAYMENT

Key
CONNECTED_WEBSITE

Key
MIGRATE_TO_CHAT_ID

Key
MIGRATE_FROM_CHAT_ID

Key
UNKNOWN

Key
ANY

ParseMode

class aiogram.types.message.**ParseMode**

Bases: `Helper`

Parse modes

Key
MARKDOWN

Key
HTML

MaskPosition

class aiogram.types.mask_position.**MaskPosition**(*conf: Optional[Dict[str, Any]] = None*, ***kwargs: Any*)

Bases: *TelegramObject*

This object describes the position on faces where a mask should be placed by default.

https://core.telegram.org/bots/api#maskposition

Deserialize object

Parameters

- **conf** –
- **kwargs** –

UserProfilePhotos

class aiogram.types.user_profile_photos.**UserProfilePhotos**(*conf: Optional[Dict[str, Any]] = None*, ***kwargs: Any*)

Bases: *TelegramObject*

This object represent a user's profile pictures.

https://core.telegram.org/bots/api#userprofilephotos

Deserialize object

Parameters

- **conf** –
- **kwargs** –

Invoice

class aiogram.types.invoice.**Invoice**(*conf: Optional[Dict[str, Any]] = None*, ***kwargs: Any*)

Bases: *TelegramObject*

This object contains basic information about an invoice.

https://core.telegram.org/bots/api#invoice

Deserialize object

Parameters

- **conf** –
- **kwargs** –

AuthWidgetData

class aiogram.types.auth_widget_data.**AuthWidgetData**(*conf: Optional[Dict[str, Any]] = None*, ***kwargs: Any*)

Bases: *TelegramObject*

Deserialize object

Parameters

- **conf** –
- **kwargs** –

classmethod parse(*request: Request*) → *AuthWidgetData*

Parse request as Telegram auth widget data.

Parameters
request –

Returns
AuthWidgetData

Raise
aiohttp.web.HTTPBadRequest

4.5 Dispatcher

4.5.1 Filters

Basics

Filter factory greatly simplifies the reuse of filters when registering handlers.

Filters factory

class aiogram.dispatcher.filters.**FiltersFactory**(*dispatcher*)

Bases: object

Filters factory

bind(*callback: Union[Callable,* AbstractFilter*], validator: Optional[Callable] = None, event_handlers: Optional[List[Handler]] = None, exclude_event_handlers: Optional[Iterable[Handler]] = None*)

Register filter

Parameters

- **callback** – callable or subclass of *AbstractFilter*
- **validator** – custom validator.
- **event_handlers** – list of instances of Handler
- **exclude_event_handlers** – list of excluded event handlers (Handler)

unbind(*callback: Union[Callable,* AbstractFilter*]*)

Unregister filter

Parameters
callback – callable of subclass of *AbstractFilter*

resolve(*event_handler, *custom_filters, **full_config*) → List[Union[Callable, *AbstractFilter*]]

Resolve filters to filters-set

Parameters

- **event_handler** –
- **custom_filters** –
- **full_config** –

Returns

Builtin filters

aiogram has some builtin filters. Here you can see all of them:

Command

class aiogram.dispatcher.filters.**Command**(*commands: Union[Iterable[Union[str, BotCommand]], str, BotCommand], prefixes: Union[Iterable, str] = '/', ignore_case: bool = True, ignore_mention: bool = False, ignore_caption: bool = True*)

Bases: *Filter*

You can handle commands by using this filter.

If filter is successful processed the *Command.CommandObj* will be passed to the handler arguments.

By default this filter is registered for messages and edited messages handlers.

Filter can be initialized from filters factory or by simply creating instance of this class.

Examples:

```
@dp.message_handler(commands=['myCommand'])
@dp.message_handler(Command(['myCommand']))
@dp.message_handler(commands=['myCommand'], commands_prefix='!/')
```

Parameters

- **commands** – Command or list of commands always without leading slashes (prefix)
- **prefixes** – Allowed commands prefix. By default is slash. If you change the default behavior pass the list of prefixes to this argument.
- **ignore_case** – Ignore case of the command
- **ignore_mention** – Ignore mention in command (By default this filter pass only the commands addressed to current bot)
- **ignore_caption** – Ignore caption from message (in message types like photo, video, audio, etc) By default is True. If you want check commands in captions, you also should set required content_types.

  Examples:

```
@dp.message_handler(commands=['myCommand'], commands_ignore_
↪caption=False, content_types=ContentType.ANY)
@dp.message_handler(Command(['myCommand'], ignore_caption=False),␣
↪content_types=[ContentType.TEXT, ContentType.DOCUMENT])
```

classmethod validate(*full_config: Dict[str, Any]*) → Optional[Dict[str, Any]]

Validator for filters factory

From filters factory this filter can be registered with arguments:

- command
- commands_prefix (will be passed as prefixes)
- commands_ignore_mention (will be passed as ignore_mention)

- `commands_ignore_caption` (will be passed as `ignore_caption`)

Parameters
full_config –

Returns
config or empty dict

async check(*message: Message*)

Will be called when filters checks.

This method must be overridden.

Parameters
args –

Returns

class CommandObj(*prefix: str = '/', command: str = '', mention: Optional[str] = None, args: Optional[str] = None*)

Bases: `object`

Instance of this object is always has command and it prefix.

Can be passed as keyword argument `command` to the handler

prefix: str = '/'

Command without prefix and mention

command: str = ''

Mention (if available)

mention: str = None

Command argument

property mentioned: bool

This command has mention?
Returns

property text: str

Generate original text from object
Returns

CommandStart

class aiogram.dispatcher.filters.**CommandStart**(*deep_link: Optional[Union[str, Pattern[str]]] = None, encoded: bool = False*)

Bases: *Command*

This filter based on *Command* filter but can handle only `/start` command.

Also this filter can handle deep-linking arguments.

Example:

```
@dp.message_handler(CommandStart(re.compile(r'ref-([\d]+)')))
```

Parameters

- **deep_link** – string or compiled regular expression (by `re.compile(...)`).
- **encoded** – set True if you're waiting for encoded payload (default - False).

async check(*message:* Message)

If deep-linking is passed to the filter result of the matching will be passed as `deep_link` to the handler

Parameters

message –

Returns

CommandHelp

class aiogram.dispatcher.filters.**CommandHelp**

Bases: *Command*

This filter based on *Command* filter but can handle only `/help` command.

Filter can be initialized from filters factory or by simply creating instance of this class.

Examples:

```
@dp.message_handler(commands=['myCommand'])
@dp.message_handler(Command(['myCommand']))
@dp.message_handler(commands=['myCommand'], commands_prefix='!/')
```

Parameters

- **commands** – Command or list of commands always without leading slashes (prefix)
- **prefixes** – Allowed commands prefix. By default is slash. If you change the default behavior pass the list of prefixes to this argument.
- **ignore_case** – Ignore case of the command
- **ignore_mention** – Ignore mention in command (By default this filter pass only the commands addressed to current bot)
- **ignore_caption** – Ignore caption from message (in message types like photo, video, audio, etc) By default is True. If you want check commands in captions, you also should set required content_types.

  Examples:

```
@dp.message_handler(commands=['myCommand'], commands_ignore_
↪caption=False, content_types=ContentType.ANY)
@dp.message_handler(Command(['myCommand'], ignore_caption=False),␣
↪content_types=[ContentType.TEXT, ContentType.DOCUMENT])
```

CommandSettings

class aiogram.dispatcher.filters.**CommandSettings**

Bases: *Command*

This filter based on *Command* filter but can handle only `/settings` command.

Filter can be initialized from filters factory or by simply creating instance of this class.

Examples:

```
@dp.message_handler(commands=['myCommand'])
@dp.message_handler(Command(['myCommand']))
@dp.message_handler(commands=['myCommand'], commands_prefix='!/')
```

Parameters

- **commands** – Command or list of commands always without leading slashes (prefix)
- **prefixes** – Allowed commands prefix. By default is slash. If you change the default behavior pass the list of prefixes to this argument.
- **ignore_case** – Ignore case of the command
- **ignore_mention** – Ignore mention in command (By default this filter pass only the commands addressed to current bot)
- **ignore_caption** – Ignore caption from message (in message types like photo, video, audio, etc) By default is True. If you want check commands in captions, you also should set required content_types.

  Examples:

```
@dp.message_handler(commands=['myCommand'], commands_ignore_
↪caption=False, content_types=ContentType.ANY)
@dp.message_handler(Command(['myCommand'], ignore_caption=False),␣
↪content_types=[ContentType.TEXT, ContentType.DOCUMENT])
```

CommandPrivacy

class aiogram.dispatcher.filters.**CommandPrivacy**

Bases: *Command*

This filter based on *Command* filter but can handle only `/privacy` command.

Filter can be initialized from filters factory or by simply creating instance of this class.

Examples:

```
@dp.message_handler(commands=['myCommand'])
@dp.message_handler(Command(['myCommand']))
@dp.message_handler(commands=['myCommand'], commands_prefix='!/')
```

Parameters

- **commands** – Command or list of commands always without leading slashes (prefix)

- **prefixes** – Allowed commands prefix. By default is slash. If you change the default behavior pass the list of prefixes to this argument.
- **ignore_case** – Ignore case of the command
- **ignore_mention** – Ignore mention in command (By default this filter pass only the commands addressed to current bot)
- **ignore_caption** – Ignore caption from message (in message types like photo, video, audio, etc) By default is True. If you want check commands in captions, you also should set required content_types.

  Examples:

```
@dp.message_handler(commands=['myCommand'], commands_ignore_
↪caption=False, content_types=ContentType.ANY)
@dp.message_handler(Command(['myCommand'], ignore_caption=False),␣
↪content_types=[ContentType.TEXT, ContentType.DOCUMENT])
```

Text

class aiogram.dispatcher.filters.**Text**(*equals: Optional[Union[str, LazyProxy, Iterable[Union[str, LazyProxy]]]] = None, contains: Optional[Union[str, LazyProxy, Iterable[Union[str, LazyProxy]]]] = None, startswith: Optional[Union[str, LazyProxy, Iterable[Union[str, LazyProxy]]]] = None, endswith: Optional[Union[str, LazyProxy, Iterable[Union[str, LazyProxy]]]] = None, ignore_case=False*)

Bases: *Filter*

Simple text filter

Check text for one of pattern. Only one mode can be used in one filter. In every pattern, a single string is treated as a list with 1 element.

Parameters

- **equals** – True if object's text in the list
- **contains** – True if object's text contains all strings from the list
- **startswith** – True if object's text starts with any of strings from the list
- **endswith** – True if object's text ends with any of strings from the list
- **ignore_case** – case insensitive

classmethod validate(*full_config: Dict[str, Any]*)

Here method `validate` is optional. If you need to use filter from filters factory you need to override this method.

Parameters
full_config – dict with arguments passed to handler registrar

Returns
Current filter config

async check(*obj: Union[*Message, CallbackQuery, InlineQuery, *Poll]*)

Will be called when filters checks.

This method must be overridden.

Parameters
args –

Returns

HashTag

class aiogram.dispatcher.filters.**HashTag**(*hashtags=None*, *cashtags=None*)

Bases: *Filter*

Filter for hashtag's and cashtag's

classmethod validate(*full_config: Dict[str, Any]*)

Here method `validate` is optional. If you need to use filter from filters factory you need to override this method.

Parameters
full_config – dict with arguments passed to handler registrar

Returns
Current filter config

async check(*message:* Message)

Will be called when filters checks.

This method must be overridden.

Parameters
args –

Returns

Regexp

class aiogram.dispatcher.filters.**Regexp**(*regexp*)

Bases: *Filter*

Regexp filter for messages and callback query

classmethod validate(*full_config: Dict[str, Any]*)

Here method `validate` is optional. If you need to use filter from filters factory you need to override this method.

Parameters
full_config – dict with arguments passed to handler registrar

Returns
Current filter config

async check(*obj: Union[*Message, CallbackQuery, InlineQuery, *Poll]*)

Will be called when filters checks.

This method must be overridden.

Parameters
args –

Returns

RegexpCommandsFilter

class aiogram.dispatcher.filters.**RegexpCommandsFilter**(*regexp_commands*)

Bases: *BoundFilter*

Check commands by regexp in message

key = 'regexp_commands'

Unique name of the filter argument. You need to override this attribute.

async check(*message*)

Will be called when filters checks.

This method must be overridden.

Parameters
args –

Returns

ContentTypeFilter

class aiogram.dispatcher.filters.**ContentTypeFilter**(*content_types*)

Bases: *BoundFilter*

Check message content type

key = 'content_types'

Unique name of the filter argument. You need to override this attribute.

required = True

If `True` this filter will be added to the all of the registered handlers

default = ['text']

Default value for configure required filters

async check(*message*)

Will be called when filters checks.

This method must be overridden.

Parameters
args –

Returns

IsSenderContact

class aiogram.dispatcher.filters.**IsSenderContact**(*is_sender_contact: bool*)

Bases: *BoundFilter*

Filter check that the contact matches the sender

is_sender_contact=True - contact matches the sender *is_sender_contact=False* - result will be inverted

key = 'is_sender_contact'

Unique name of the filter argument. You need to override this attribute.

async check(*message:* Message) → bool

Will be called when filters checks.

This method must be overridden.

Parameters
args –

Returns

StateFilter

class aiogram.dispatcher.filters.**StateFilter**(*dispatcher*, *state*)

Bases: *BoundFilter*

Check user state

key = 'state'

Unique name of the filter argument. You need to override this attribute.

required = True

If True this filter will be added to the all of the registered handlers

async check(*obj*)

Will be called when filters checks.

This method must be overridden.

Parameters
args –

Returns

ExceptionsFilter

class aiogram.dispatcher.filters.**ExceptionsFilter**(*exception*)

Bases: *BoundFilter*

Filter for exceptions

key = 'exception'

Unique name of the filter argument. You need to override this attribute.

async check(*update*, *exception*)

Will be called when filters checks.

This method must be overridden.

Parameters
args –

Returns

IDFilter

class aiogram.dispatcher.filters.builtin.**IDFilter**(*user_id: Optional[Union[Iterable[Union[int, str]], str, int]] = None*, *chat_id: Optional[Union[Iterable[Union[int, str]], str, int]] = None*)

Bases: *Filter*

Parameters

- **user_id** –
- **chat_id** –

classmethod validate(*full_config: Dict[str, Any]*) → Optional[Dict[str, Any]]

Here method `validate` is optional. If you need to use filter from filters factory you need to override this method.

Parameters
full_config – dict with arguments passed to handler registrar

Returns
Current filter config

async check(*obj: Union[*Message, CallbackQuery, InlineQuery, *ChatMemberUpdated, ChatJoinRequest]*)

Will be called when filters checks.

This method must be overridden.

Parameters
args –

Returns

AdminFilter

class aiogram.dispatcher.filters.**AdminFilter**(*is_chat_admin: Optional[Union[Iterable[Union[int, str]], str, int, bool]] = None*)

Bases: *Filter*

Checks if user is admin in a chat. If is_chat_admin is not set, the filter will check in the current chat (correct only for messages). is_chat_admin is required for InlineQuery.

classmethod validate(*full_config: Dict[str, Any]*) → Optional[Dict[str, Any]]

Here method `validate` is optional. If you need to use filter from filters factory you need to override this method.

Parameters
full_config – dict with arguments passed to handler registrar

Returns
Current filter config

async check(*obj: Union[*Message, CallbackQuery, InlineQuery, *ChatMemberUpdated]*) → bool

Will be called when filters checks.

This method must be overridden.

Parameters
args –

Returns

IsReplyFilter

class aiogram.dispatcher.filters.**IsReplyFilter**(*is_reply*)

Bases: *BoundFilter*

Check if message is replied and send reply message to handler

key = 'is_reply'

Unique name of the filter argument. You need to override this attribute.

async check(*msg: Message*)

Will be called when filters checks.

This method must be overridden.

Parameters
args –

Returns

ForwardedMessageFilter

class aiogram.dispatcher.filters.**ForwardedMessageFilter**(*is_forwarded: bool*)

Bases: *BoundFilter*

key = 'is_forwarded'

Unique name of the filter argument. You need to override this attribute.

async check(*message: Message*)

Will be called when filters checks.

This method must be overridden.

Parameters
args –

Returns

ChatTypeFilter

class aiogram.dispatcher.filters.**ChatTypeFilter**(*chat_type: Container[ChatType]*)

Bases: *BoundFilter*

key = 'chat_type'

Unique name of the filter argument. You need to override this attribute.

async check(*obj: Union[Message, CallbackQuery, ChatMemberUpdated, InlineQuery]*)

Will be called when filters checks.

This method must be overridden.

Parameters
args –

Returns

MediaGroupFilter

class aiogram.dispatcher.filters.**MediaGroupFilter**(*is_media_group: bool*)

Bases: *BoundFilter*

Check if message is part of a media group.

is_media_group=True - the message is part of a media group *is_media_group=False* - the message is NOT part of a media group

key = 'is_media_group'

Unique name of the filter argument. You need to override this attribute.

async check(*message:* Message) → bool

Will be called when filters checks.

This method must be overridden.

Parameters
args –

Returns

Making own filters (Custom filters)

Own filter can be:

- any callable object
- any async function
- any anonymous function (Example: `lambda msg: msg.text == 'spam'`)
- Subclass of `AbstractFilter`, `Filter` or `BoundFilter`

AbstractFilter

class aiogram.dispatcher.filters.**AbstractFilter**

Bases: `ABC`

Abstract class for custom filters.

abstract classmethod validate(*full_config: Dict[str, Any]*) → Optional[Dict[str, Any]]

Validate and parse config.

This method will be called by the filters factory when you bind this filter. Must be overridden.

Parameters
full_config – dict with arguments passed to handler registrar

Returns
Current filter config

abstract async check(**args*) → bool

Will be called when filters checks.

This method must be overridden.

Parameters
args –

Returns

Filter

class aiogram.dispatcher.filters.**Filter**

Bases: *AbstractFilter*

You can make subclasses of that class for custom filters.

Method `check` must be overridden

classmethod validate(*full_config: Dict[str, Any]*) → Optional[Dict[str, Any]]

Here method `validate` is optional. If you need to use filter from filters factory you need to override this method.

Parameters
full_config – dict with arguments passed to handler registrar

Returns
Current filter config

BoundFilter

class aiogram.dispatcher.filters.**BoundFilter**

Bases: *Filter*

To easily create your own filters with one parameter, you can inherit from this filter.

You need to implement `__init__` method with single argument related with key attribute and `check` method where you need to implement filter logic.

key = None

Unique name of the filter argument. You need to override this attribute.

required = False

If `True` this filter will be added to the all of the registered handlers

default = None

Default value for configure required filters

classmethod validate(*full_config: Dict[str, Any]*) → Dict[str, Any]

If `cls.key` is not `None` and that is in config returns config with that argument.

Parameters
full_config –

Returns

```
class ChatIdFilter(BoundFilter):
    key = 'chat_id'

    def __init__(self, chat_id: typing.Union[typing.Iterable, int]):
        if isinstance(chat_id, int):
            chat_id = [chat_id]
        self.chat_id = chat_id
```

(continues on next page)

(continued from previous page)

```
    def check(self, message: types.Message) -> bool:
        return message.chat.id in self.chat_id


dp.filters_factory.bind(ChatIdFilter, event_handlers=[dp.message_handlers])
```

4.5.2 Finite state machine

Storage

Coming soon…

Available storage's

Coming soon…

Memory storage

class aiogram.contrib.fsm_storage.memory.**MemoryStorage**

Bases: BaseStorage

In-memory based states storage.

This type of storage is not recommended for usage in bots, because you will lost all states after restarting.

Redis storage

class aiogram.contrib.fsm_storage.redis.**RedisStorage2**(*host: str = 'localhost', port: int = 6379, db: Optional[int] = None, password: Optional[str] = None, ssl: Optional[bool] = None, pool_size: int = 10, loop: Optional[AbstractEventLoop] = None, prefix: str = 'fsm', state_ttl: Optional[int] = None, data_ttl: Optional[int] = None, bucket_ttl: Optional[int] = None, **kwargs*)

Bases: BaseStorage

Busted Redis-base storage for FSM. Works with Redis connection pool and customizable keys prefix.

Usage:

```
storage = RedisStorage2('localhost', 6379, db=5, pool_size=10, prefix='my_fsm_key')
dp = Dispatcher(bot, storage=storage)
```

And need to close Redis connection when shutdown

```
await dp.storage.close()
```

Mongo storage

class aiogram.contrib.fsm_storage.mongo.**MongoStorage**(*host='localhost', port=27017, db_name='aiogram_fsm', uri=None, username=None, password=None, index=True, **kwargs*)

Bases: `BaseStorage`

Mongo-based storage for FSM.

Usage:

```
storage = MongoStorage(host='localhost', port=27017, db_name='aiogram_fsm')
dp = Dispatcher(bot, storage=storage)
```

And need to close Mongo client connections when shutdown

```
await dp.storage.close()
await dp.storage.wait_closed()
```

Rethink DB storage

class aiogram.contrib.fsm_storage.rethinkdb.**RethinkDBStorage**(*host: str = 'localhost', port: int = 28015, db: str = 'aiogram', table: str = 'aiogram', auth_key: Optional[str] = None, user: Optional[str] = None, password: Optional[str] = None, timeout: int = 20, ssl: Optional[dict] = None, loop: Optional[AbstractEventLoop] = None*)

Bases: `BaseStorage`

RethinkDB-based storage for FSM.

Usage:

```
storage = RethinkDBStorage(db='aiogram', table='aiogram', user='aiogram', password=
↪'aiogram_secret')
dispatcher = Dispatcher(bot, storage=storage)
```

And need to close connection when shutdown

```
await storage.close()
await storage.wait_closed()
```

Making own storage's

Coming soon…

States

Coming soon…

State utils

Coming soon…

State

Coming soon…

States group

Coming soon…

4.5.3 Middleware

Bases

Coming soon…

Making own middleware's

Coming soon…

Available middleware's

Coming soon…

4.5.4 Webhook

Coming soon…

Bases

Coming soon…

Security

Coming soon…

Making requests when getting updates

Coming soon…

4.5.5 Basics

Coming soon…

4.5.6 Available handlers

Coming soon…

Handler class

Coming soon…

4.5.7 Features

Coming soon…

4.5.8 Dispatcher class

class aiogram.**Dispatcher**(*bot*, *loop=None*, *storage: Optional[BaseStorage] = None*, *run_tasks_by_default: bool = False*, *throttling_rate_limit=0.1*, *no_throttle_error=False*, *filters_factory=None*)

Bases: `DataMixin`, `ContextInstanceMixin`

Simple Updates dispatcher

It will process incoming updates: messages, edited messages, channel posts, edited channel posts, inline queries, chosen inline results, callback queries, shipping queries, pre-checkout queries.

async skip_updates()

You can skip old incoming updates from queue. This method is not recommended for using in production.

Note that the webhook will be deleted!

async process_updates(*updates*, *fast: bool = True*)

Process list of updates

Parameters

- **updates** –

- **fast** –

Returns

async process_update(*update:* Update)

Process single update object

Parameters
update –

Returns

async reset_webhook(*check=True*) → bool

Reset webhook

Parameters
check – check before deleting

Returns

async start_polling(*timeout=20*, *relax=0.1*, *limit=None*, *reset_webhook=None*, *fast: bool = True*, *error_sleep: int = 5*, *allowed_updates: Optional[List[str]] = None*)

Start long-polling

Parameters

- **timeout** –
- **relax** –
- **limit** –
- **reset_webhook** –
- **fast** –
- **error_sleep** –
- **allowed_updates** –

Returns

stop_polling()

Break long-polling process.

Returns

async wait_closed()

Wait for the long-polling to close

Returns

is_polling()

Check if polling is enabled

Returns

register_message_handler(*callback*, **custom_filters*, *commands=None*, *regexp=None*, *content_types=None*, *state=None*, *run_task=None*, ***kwargs*)

Register handler for message

```
# This handler works only if state is None (by default).
dp.register_message_handler(cmd_start, commands=['start', 'about'])
dp.register_message_handler(entry_point, commands=['setup'])

# This handler works only if current state is "first_step"
dp.register_message_handler(step_handler_1, state="first_step")

# If you want to handle all states by one handler, use `state="*"`.
dp.register_message_handler(cancel_handler, commands=['cancel'], state="*")
dp.register_message_handler(cancel_handler, lambda msg: msg.text.lower() ==
↪'cancel', state="*")
```

Parameters

- **callback** –
- **commands** – list of commands
- **regexp** – REGEXP
- **content_types** – List of content types.
- **custom_filters** – list of custom filters
- **kwargs** –
- **state** –

Returns
decorated function

message_handler(**custom_filters, commands=None, regexp=None, content_types=None, state=None, run_task=None, **kwargs*)

Decorator for message handler

Examples:

Simple commands handler:

```
@dp.message_handler(commands=['start', 'welcome', 'about'])
async def cmd_handler(message: types.Message):
```

Filter messages by regular expression:

```
@dp.message_handler(regexp='^[a-z]+-[0-9]+')
async def msg_handler(message: types.Message):
```

Filter messages by command regular expression:

```
@dp.message_handler(filters.RegexpCommandsFilter(regexp_commands=['item_([0-9]*)
↪']))
async def send_welcome(message: types.Message):
```

Filter by content type:

```
@dp.message_handler(content_types=ContentType.PHOTO | ContentType.DOCUMENT)
async def audio_handler(message: types.Message):
```

Filter by custom function:

```
@dp.message_handler(lambda message: message.text and 'hello' in message.text.
↪lower())
async def text_handler(message: types.Message):
```

Use multiple filters:

```
@dp.message_handler(commands=['command'], content_types=ContentType.TEXT)
async def text_handler(message: types.Message):
```

Register multiple filters set for one handler:

```
@dp.message_handler(commands=['command'])
@dp.message_handler(lambda message: demojize(message.text) == ':new_moon_with_
↪face:')
async def text_handler(message: types.Message):
```

This handler will be called if the message starts with '/command' OR is some emoji

By default content_type is `ContentType.TEXT`

Parameters

- **commands** – list of commands
- **regexp** – REGEXP
- **content_types** – List of content types.
- **custom_filters** – list of custom filters
- **kwargs** –
- **state** –
- **run_task** – run callback in task (no wait results)

Returns
decorated function

register_edited_message_handler(*callback*, **custom_filters*, *commands=None*, *regexp=None*, *content_types=None*, *state=None*, *run_task=None*, ***kwargs*)

Register handler for edited message

Parameters

- **callback** –
- **commands** – list of commands
- **regexp** – REGEXP
- **content_types** – List of content types.
- **state** –
- **custom_filters** – list of custom filters
- **run_task** – run callback in task (no wait results)
- **kwargs** –

Returns
decorated function

edited_message_handler(**custom_filters*, *commands=None*, *regexp=None*, *content_types=None*, *state=None*, *run_task=None*, ***kwargs*)

Decorator for edited message handler

You can use combination of different handlers

```
@dp.message_handler()
@dp.edited_message_handler()
async def msg_handler(message: types.Message):
```

Parameters

- **commands** – list of commands
- **regexp** – REGEXP
- **content_types** – List of content types.
- **state** –
- **custom_filters** – list of custom filters
- **run_task** – run callback in task (no wait results)
- **kwargs** –

Returns
decorated function

register_channel_post_handler(*callback*, **custom_filters*, *commands=None*, *regexp=None*, *content_types=None*, *state=None*, *run_task=None*, ***kwargs*)

Register handler for channel post

Parameters

- **callback** –
- **commands** – list of commands
- **regexp** – REGEXP
- **content_types** – List of content types.
- **state** –
- **custom_filters** – list of custom filters
- **run_task** – run callback in task (no wait results)
- **kwargs** –

Returns
decorated function

channel_post_handler(**custom_filters*, *commands=None*, *regexp=None*, *content_types=None*, *state=None*, *run_task=None*, ***kwargs*)

Decorator for channel post handler

Parameters

- **commands** – list of commands
- **regexp** – REGEXP
- **content_types** – List of content types.

- **state** –
- **custom_filters** – list of custom filters
- **run_task** – run callback in task (no wait results)
- **kwargs** –

Returns
decorated function

register_edited_channel_post_handler(*callback*, **custom_filters*, *commands=None*, *regexp=None*, *content_types=None*, *state=None*, *run_task=None*, ***kwargs*)

Register handler for edited channel post

Parameters

- **callback** –
- **commands** – list of commands
- **regexp** – REGEXP
- **content_types** – List of content types.
- **state** –
- **custom_filters** – list of custom filters
- **run_task** – run callback in task (no wait results)
- **kwargs** –

Returns
decorated function

edited_channel_post_handler(**custom_filters*, *commands=None*, *regexp=None*, *content_types=None*, *state=None*, *run_task=None*, ***kwargs*)

Decorator for edited channel post handler

Parameters

- **commands** – list of commands
- **regexp** – REGEXP
- **content_types** – List of content types.
- **custom_filters** – list of custom filters
- **state** –
- **run_task** – run callback in task (no wait results)
- **kwargs** –

Returns
decorated function

register_inline_handler(*callback*, **custom_filters*, *state=None*, *run_task=None*, ***kwargs*)

Register handler for inline query

Example:

```
dp.register_inline_handler(some_inline_handler, lambda inline_query: True)
```

Parameters

- **callback** –
- **custom_filters** – list of custom filters
- **state** –
- **run_task** – run callback in task (no wait results)
- **kwargs** –

Returns
decorated function

inline_handler(**custom_filters*, *state=None*, *run_task=None*, ***kwargs*)

Decorator for inline query handler

Example:

```
@dp.inline_handler(lambda inline_query: True)
async def some_inline_handler(inline_query: types.InlineQuery)
```

Parameters

- **state** –
- **custom_filters** – list of custom filters
- **run_task** – run callback in task (no wait results)
- **kwargs** –

Returns
decorated function

register_chosen_inline_handler(*callback*, **custom_filters*, *state=None*, *run_task=None*, ***kwargs*)

Register handler for chosen inline query

Example:

```
dp.register_chosen_inline_handler(some_chosen_inline_handler, lambda chosen_
↪inline_result: True)
```

Parameters

- **callback** –
- **state** –
- **custom_filters** –
- **run_task** – run callback in task (no wait results)
- **kwargs** –

Returns

chosen_inline_handler(**custom_filters*, *state=None*, *run_task=None*, ***kwargs*)

Decorator for chosen inline query handler

Example:

```
@dp.chosen_inline_handler(lambda chosen_inline_result: True)
async def some_chosen_inline_handler(chosen_inline_result: types.
↪ChosenInlineResult)
```

Parameters

- **state** –
- **custom_filters** –
- **run_task** – run callback in task (no wait results)
- **kwargs** –

Returns

register_callback_query_handler(*callback*, **custom_filters*, *state=None*, *run_task=None*, ***kwargs*)

Register handler for callback query

Example:

```
dp.register_callback_query_handler(some_callback_handler, lambda callback_
↪query: True)
```

Parameters

- **callback** –
- **state** –
- **custom_filters** –
- **run_task** – run callback in task (no wait results)
- **kwargs** –

callback_query_handler(**custom_filters*, *state=None*, *run_task=None*, ***kwargs*)

Decorator for callback query handler

Example:

```
@dp.callback_query_handler(lambda callback_query: True)
async def some_callback_handler(callback_query: types.CallbackQuery)
```

Parameters

- **state** –
- **custom_filters** –
- **run_task** – run callback in task (no wait results)
- **kwargs** –

register_shipping_query_handler(*callback*, **custom_filters*, *state=None*, *run_task=None*, ***kwargs*)

Register handler for shipping query

Example:

```
dp.register_shipping_query_handler(some_shipping_query_handler, lambda shipping_
↪query: True)
```

Parameters

- **callback** –
- **state** –
- **custom_filters** –
- **run_task** – run callback in task (no wait results)
- **kwargs** –

shipping_query_handler(**custom_filters, state=None, run_task=None, **kwargs*)

Decorator for shipping query handler

Example:

```
@dp.shipping_query_handler(lambda shipping_query: True)
async def some_shipping_query_handler(shipping_query: types.ShippingQuery)
```

Parameters

- **state** –
- **custom_filters** –
- **run_task** – run callback in task (no wait results)
- **kwargs** –

register_pre_checkout_query_handler(*callback, *custom_filters, state=None, run_task=None, **kwargs*)

Register handler for pre-checkout query

Example:

```
dp.register_pre_checkout_query_handler(some_pre_checkout_query_handler, lambda␣
↪shipping_query: True)
```

Parameters

- **callback** –
- **state** –
- **custom_filters** –
- **run_task** – run callback in task (no wait results)
- **kwargs** –

pre_checkout_query_handler(**custom_filters, state=None, run_task=None, **kwargs*)

Decorator for pre-checkout query handler

Example:

```
@dp.pre_checkout_query_handler(lambda shipping_query: True)
async def some_pre_checkout_query_handler(shipping_query: types.ShippingQuery)
```

Parameters

- **state** –
- **custom_filters** –
- **run_task** – run callback in task (no wait results)
- **kwargs** –

register_poll_handler(*callback*, **custom_filters*, *run_task=None*, ***kwargs*)

Register handler for poll

Example:

```
dp.register_poll_handler(some_poll_handler)
```

Parameters

- **callback** –
- **custom_filters** –
- **run_task** – run callback in task (no wait results)
- **kwargs** –

poll_handler(**custom_filters*, *run_task=None*, ***kwargs*)

Decorator for poll handler

Example:

```
@dp.poll_handler()
async def some_poll_handler(poll: types.Poll)
```

Parameters

- **custom_filters** –
- **run_task** – run callback in task (no wait results)
- **kwargs** –

register_poll_answer_handler(*callback*, **custom_filters*, *run_task=None*, ***kwargs*)

Register handler for poll_answer

Example:

```
dp.register_poll_answer_handler(some_poll_answer_handler)
```

Parameters

- **callback** –
- **custom_filters** –
- **run_task** – run callback in task (no wait results)

- **kwargs** –

poll_answer_handler(**custom_filters, run_task=None, **kwargs*)

Decorator for poll_answer handler

Example:

```
@dp.poll_answer_handler()
async def some_poll_answer_handler(poll_answer: types.PollAnswer)
```

Parameters

- **custom_filters** –
- **run_task** – run callback in task (no wait results)
- **kwargs** –

register_my_chat_member_handler(*callback: Callable, *custom_filters, run_task: Optional[bool] = None, **kwargs*) → None

Register handler for my_chat_member

Example:

```
dp.register_my_chat_member_handler(some_my_chat_member_handler)
```

Parameters

- **callback** –
- **custom_filters** –
- **run_task** – run callback in task (no wait results)
- **kwargs** –

my_chat_member_handler(**custom_filters, run_task=None, **kwargs*)

Decorator for my_chat_member handler

Example:

```
@dp.my_chat_member_handler()
async def some_handler(my_chat_member: types.ChatMemberUpdated)
```

Parameters

- **custom_filters** –
- **run_task** – run callback in task (no wait results)
- **kwargs** –

register_chat_member_handler(*callback: Callable, *custom_filters, run_task: Optional[bool] = None, **kwargs*) → None

Register handler for chat_member

Example:

```
dp.register_chat_member_handler(some_chat_member_handler)
```

Parameters

- **callback** –
- **custom_filters** –
- **run_task** – run callback in task (no wait results)
- **kwargs** –

chat_member_handler(**custom_filters*, *run_task=None*, ***kwargs*)

Decorator for chat_member handler

Example:

```
@dp.chat_member_handler()
async def some_handler(chat_member: types.ChatMemberUpdated)
```

Parameters

- **custom_filters** –
- **run_task** – run callback in task (no wait results)
- **kwargs** –

register_chat_join_request_handler(*callback: Callable*, **custom_filters*, *run_task: Optional[bool] = None*, ***kwargs*) → None

Register handler for chat_join_request

Example:

```
dp.register_chat_join_request(some_chat_join_request)
```

Parameters

- **callback** –
- **custom_filters** –
- **run_task** – run callback in task (no wait results)
- **kwargs** –

chat_join_request_handler(**custom_filters*, *run_task=None*, ***kwargs*)

Decorator for chat_join_request handler

Example:

```
@dp.chat_join_request()
async def some_handler(chat_member: types.ChatJoinRequest)
```

Parameters

- **custom_filters** –
- **run_task** – run callback in task (no wait results)

- **kwargs** –

register_errors_handler(*callback*, **custom_filters*, *exception=None*, *run_task=None*, ***kwargs*)

Register handler for errors

Parameters

- **callback** –
- **exception** – you can make handler for specific errors type
- **run_task** – run callback in task (no wait results)

errors_handler(**custom_filters*, *exception=None*, *run_task=None*, ***kwargs*)

Decorator for errors handler

Parameters

- **exception** – you can make handler for specific errors type
- **run_task** – run callback in task (no wait results)

Returns

current_state(**, chat: Optional[Union[str, int]] = None, user: Optional[Union[str, int]] = None*) → FSMContext

Get current state for user in chat as context

```
with dp.current_state(chat=message.chat.id, user=message.user.id) as state:
    pass

state = dp.current_state()
state.set_state('my_state')
```

Parameters

- **chat** –
- **user** –

Returns

async throttle(*key*, **, rate=None*, *user_id=None*, *chat_id=None*, *no_error=None*) → bool

Execute throttling manager. Returns True if limit has not exceeded otherwise raises ThrottleError or returns False

Parameters

- **key** – key in storage
- **rate** – limit (by default is equal to default rate limit)
- **user_id** – user id
- **chat_id** – chat id
- **no_error** – return boolean value instead of raising error

Returns

bool

async check_key(*key*, *chat_id=None*, *user_id=None*)

Get information about key in bucket

Parameters

- **key** –
- **chat_id** –
- **user_id** –

Returns

async release_key(*key*, *chat_id=None*, *user_id=None*)

Release blocked key

Parameters

- **key** –
- **chat_id** –
- **user_id** –

Returns

async_task(*func*)

Execute handler as task and return None. Use this decorator for slow handlers (with timeouts)

```
@dp.message_handler(commands=['command'])
@dp.async_task
async def cmd_with_timeout(message: types.Message):
    await asyncio.sleep(120)
    return SendMessage(message.chat.id, 'KABOOM').reply(message)
```

Parameters

func –

Returns

throttled(*on_throttled: Optional[Callable] = None*, *key=None*, *rate=None*, *user_id=None*, *chat_id=None*)

Meta-decorator for throttling. Invokes on_throttled if the handler was throttled.

Example:

```
async def handler_throttled(message: types.Message, **kwargs):
    await message.answer("Throttled!")

@dp.throttled(handler_throttled)
async def some_handler(message: types.Message):
    await message.answer("Didn't throttled!")
```

Parameters

- **on_throttled** – the callable object that should be either a function or return a coroutine
- **key** – key in storage
- **rate** – limit (by default is equal to default rate limit)
- **user_id** – user id

- **chat_id** – chat id

Returns
decorator

bind_filter(*callback: Union[Callable, AbstractFilter], validator: Optional[Callable] = None, event_handlers: Optional[List[Handler]] = None, exclude_event_handlers: Optional[Iterable[Handler]] = None*)

Register filter

Parameters

- **callback** – callable or subclass of `AbstractFilter`
- **validator** – custom validator.
- **event_handlers** – list of instances of `Handler`
- **exclude_event_handlers** – list of excluded event handlers (`Handler`)

unbind_filter(*callback: Union[Callable, AbstractFilter]*)

Unregister filter

Parameters
callback – callable of subclass of `AbstractFilter`

setup_middleware(*middleware*)

Setup middleware

Parameters
middleware –

Returns

4.6 Utils

4.6.1 Auth Widget

Implementation of Telegram site authorization checking mechanism for more information https://core.telegram.org/widgets/login#checking-authorization

Source: https://gist.github.com/JrooTJunior/887791de7273c9df5277d2b1ecadc839

`aiogram.utils.auth_widget.`**generate_hash**(*data: dict, token: str*) → str

Generate secret hash

Parameters

- **data** –
- **token** –

Returns

`aiogram.utils.auth_widget.`**check_token**(*data: dict, token: str*) → bool

Validate auth token

Parameters

- **data** –

- **token** –

Returns

`aiogram.utils.auth_widget.check_signature(token: str, hash: str, **kwargs) → bool`

Generate hexadecimal representation of the HMAC-SHA-256 signature of the data-check-string with the SHA256 hash of the bot's token used as a secret key

Parameters

- **token** –
- **hash** –
- **kwargs** – all params received on auth

Returns

`aiogram.utils.auth_widget.check_integrity(token: str, data: dict) → bool`

Verify the authentication and the integrity of the data received on user's auth

Parameters

- **token** – Bot's token
- **data** – all data that came on auth

Returns

4.6.2 Executor

`aiogram.utils.executor.start_polling(dispatcher, *, loop=None, skip_updates=False, reset_webhook=True, on_startup=None, on_shutdown=None, timeout=20, relax=0.1, fast=True, allowed_updates: Optional[List[str]] = None)`

Start bot in long-polling mode

Parameters

- **dispatcher** –
- **loop** –
- **skip_updates** –
- **reset_webhook** –
- **on_startup** –
- **on_shutdown** –
- **timeout** –
- **relax** –
- **fast** –
- **allowed_updates** –

`aiogram.utils.executor.set_webhook(dispatcher: Dispatcher, webhook_path: str, *, loop: Optional[AbstractEventLoop] = None, skip_updates: Optional[bool] = None, on_startup: Optional[Callable] = None, on_shutdown: Optional[Callable] = None, check_ip: bool = False, retry_after: Optional[Union[str, int]] = None, route_name: str = 'webhook_handler', web_app: Optional[Application] = None)`

Set webhook for bot

Parameters

- **dispatcher** – Dispatcher
- **webhook_path** – str
- **loop** – Optional[asyncio.AbstractEventLoop] (default: None)
- **skip_updates** – bool (default: None)
- **on_startup** – Optional[Callable] (default: None)
- **on_shutdown** – Optional[Callable] (default: None)
- **check_ip** – bool (default: False)
- **retry_after** – Optional[Union[str, int]] See https://tools.ietf.org/html/rfc7231#section-7.1.3 (default: None)
- **route_name** – str (default: 'webhook_handler')
- **web_app** – Optional[Application] (default: None)

Returns

`aiogram.utils.executor.start_webhook(dispatcher, webhook_path, *, loop=None, skip_updates=None, on_startup=None, on_shutdown=None, check_ip=False, retry_after=None, route_name='webhook_handler', **kwargs)`

Start bot in webhook mode

Parameters

- **dispatcher** –
- **webhook_path** –
- **loop** –
- **skip_updates** –
- **on_startup** –
- **on_shutdown** –
- **check_ip** –
- **route_name** –
- **kwargs** –

Returns

`aiogram.utils.executor.start(dispatcher, future, *, loop=None, skip_updates=None, on_startup=None, on_shutdown=None)`

Execute Future.

Parameters

- **dispatcher** – instance of Dispatcher
- **future** – future
- **loop** – instance of AbstractEventLoop
- **skip_updates** –

- **on_startup** –
- **on_shutdown** –

Returns

class aiogram.utils.executor.**Executor**(*dispatcher*, *skip_updates=None*, *check_ip=False*, *retry_after=None*, *loop=None*)

Main executor class

set_web_app(*application: Application*)

Change instance of aiohttp.web.Application

Parameters
application –

on_startup(*callback: callable*, *polling=True*, *webhook=True*)

Register a callback for the startup process

Parameters

- **callback** –
- **polling** – use with polling
- **webhook** – use with webhook

on_shutdown(*callback: callable*, *polling=True*, *webhook=True*)

Register a callback for the shutdown process

Parameters

- **callback** –
- **polling** – use with polling
- **webhook** – use with webhook

set_webhook(*webhook_path: ~typing.Optional[str] = None*, *request_handler: ~typing.Any = <class 'aiogram.dispatcher.webhook.WebhookRequestHandler'>*, *route_name: str = 'webhook_handler'*, *web_app: ~typing.Optional[~aiohttp.web_app.Application] = None*)

Set webhook for bot

Parameters

- **webhook_path** – Optional[str] (default: None)
- **request_handler** – Any (default: WebhookRequestHandler)
- **route_name** – str Name of webhook handler route (default: 'webhook_handler')
- **web_app** – Optional[Application] (default: None)

Returns

start_webhook(*webhook_path=None*, *request_handler=<class 'aiogram.dispatcher.webhook.WebhookRequestHandler'>*, *route_name='webhook_handler'*, ***kwargs*)

Start bot in webhook mode

Parameters

- **webhook_path** –
- **request_handler** –

- **route_name** – Name of webhook handler route
- **kwargs** –

Returns

start_polling(*reset_webhook=None*, *timeout=20*, *relax=0.1*, *fast=True*, *allowed_updates: Optional[List[str]] = None*)

Start bot in long-polling mode

Parameters

- **reset_webhook** –
- **timeout** –

start(*future*)

Execute Future.

Return the Future's result, or raise its exception.

Parameters
future –

Returns

4.6.3 Exceptions

- **TelegramAPIError**
  - ValidationError
  - Throttled
  - **BadRequest**
    - **MessageError**
      - MessageNotModified
      - MessageToForwardNotFound
      - MessageIdInvalid
      - MessageToDeleteNotFound
      - MessageToPinNotFound
      - MessageIdentifierNotSpecified
      - MessageTextIsEmpty
      - MessageCantBeEdited
      - MessageCantBeDeleted
      - MessageCantBeForwarded
      - MessageToEditNotFound
      - MessageToReplyNotFound
      - ToMuchMessages
    - **PollError**
      - PollCantBeStopped

- PollHasAlreadyClosed
- PollsCantBeSentToPrivateChats
- **PollSizeError**

  PollMustHaveMoreOptions

  PollCantHaveMoreOptions

  PollsOptionsLengthTooLong

  PollOptionsMustBeNonEmpty

  PollQuestionMustBeNonEmpty
- MessageWithPollNotFound (with MessageError)
- MessageIsNotAPoll (with MessageError)

* ObjectExpectedAsReplyMarkup
* InlineKeyboardExpected
* ChatNotFound
* ChatDescriptionIsNotModified
* InvalidQueryID
* InvalidPeerID
* InvalidHTTPUrlContent
* ButtonURLInvalid
* URLHostIsEmpty
* StartParamInvalid
* ButtonDataInvalid
* FileIsTooBig
* WrongFileIdentifier
* GroupDeactivated
* **BadWebhook**
  - WebhookRequireHTTPS
  - BadWebhookPort
  - BadWebhookAddrInfo
  - BadWebhookNoAddressAssociatedWithHostname
* **NotFound**
  - MethodNotKnown
* PhotoAsInputFileRequired
* InvalidStickersSet
* NoStickerInRequest
* ChatAdminRequired
* NeedAdministratorRightsInTheChannel

        * MethodNotAvailableInPrivateChats
        * CantDemoteChatCreator
        * CantRestrictSelf
        * NotEnoughRightsToRestrict
        * PhotoDimensions
        * UnavailableMembers
        * TypeOfFileMismatch
        * WrongRemoteFileIdSpecified
        * PaymentProviderInvalid
        * CurrencyTotalAmountInvalid
        * CantParseUrl
        * UnsupportedUrlProtocol
        * CantParseEntities
        * ResultIdDuplicate
        * MethodIsNotAvailable
    - **ConflictError**
        * TerminatedByOtherGetUpdates
        * CantGetUpdates
    - **Unauthorized**
        * BotKicked
        * BotBlocked
        * UserDeactivated
        * CantInitiateConversation
        * CantTalkWithBots
    - NetworkError
    - RetryAfter
    - MigrateToChat
    - RestartingTelegram
- **AIOGramWarning**
    - TimeoutWarning

exception aiogram.utils.exceptions.**TelegramAPIError**(*message=None*)

exception aiogram.utils.exceptions.**AIOGramWarning**

exception aiogram.utils.exceptions.**TimeoutWarning**

exception aiogram.utils.exceptions.**FSMStorageWarning**

exception aiogram.utils.exceptions.**ValidationError**(*message=None*)

exception aiogram.utils.exceptions.**BadRequest**(*message=None*)

exception aiogram.utils.exceptions.**MessageError**(*message=None*)

exception aiogram.utils.exceptions.**MessageNotModified**(*message=None*)

Will be raised when you try to set new text is equals to current text.

exception aiogram.utils.exceptions.**MessageToForwardNotFound**(*message=None*)

Will be raised when you try to forward very old or deleted or unknown message.

exception aiogram.utils.exceptions.**MessageIdInvalid**(*message=None*)

exception aiogram.utils.exceptions.**MessageToDeleteNotFound**(*message=None*)

Will be raised when you try to delete very old or deleted or unknown message.

exception aiogram.utils.exceptions.**MessageToPinNotFound**(*message=None*)

Will be raised when you try to pin deleted or unknown message.

exception aiogram.utils.exceptions.**MessageToReplyNotFound**(*message=None*)

Will be raised when you try to reply to very old or deleted or unknown message.

exception aiogram.utils.exceptions.**MessageIdentifierNotSpecified**(*message=None*)

exception aiogram.utils.exceptions.**MessageTextIsEmpty**(*message=None*)

exception aiogram.utils.exceptions.**MessageCantBeEdited**(*message=None*)

exception aiogram.utils.exceptions.**MessageCantBeDeleted**(*message=None*)

exception aiogram.utils.exceptions.**MessageCantBeForwarded**(*message=None*)

exception aiogram.utils.exceptions.**MessageToEditNotFound**(*message=None*)

exception aiogram.utils.exceptions.**MessageIsTooLong**(*message=None*)

exception aiogram.utils.exceptions.**ToMuchMessages**(*message=None*)

Will be raised when you try to send media group with more than 10 items.

exception aiogram.utils.exceptions.**ObjectExpectedAsReplyMarkup**(*message=None*)

exception aiogram.utils.exceptions.**InlineKeyboardExpected**(*message=None*)

exception aiogram.utils.exceptions.**PollError**(*message=None*)

exception aiogram.utils.exceptions.**PollCantBeStopped**(*message=None*)

exception aiogram.utils.exceptions.**PollHasAlreadyBeenClosed**(*message=None*)

exception aiogram.utils.exceptions.**PollsCantBeSentToPrivateChats**(*message=None*)

exception aiogram.utils.exceptions.**PollSizeError**(*message=None*)

exception aiogram.utils.exceptions.**PollMustHaveMoreOptions**(*message=None*)

exception aiogram.utils.exceptions.**PollCantHaveMoreOptions**(*message=None*)

exception aiogram.utils.exceptions.**PollOptionsMustBeNonEmpty**(*message=None*)

exception aiogram.utils.exceptions.**PollQuestionMustBeNonEmpty**(*message=None*)

exception aiogram.utils.exceptions.**PollOptionsLengthTooLong**(*message=None*)

exception aiogram.utils.exceptions.**PollQuestionLengthTooLong**(*message=None*)

exception aiogram.utils.exceptions.**PollCanBeRequestedInPrivateChatsOnly**(*message=None*)

exception aiogram.utils.exceptions.**MessageWithPollNotFound**(*message=None*)

Will be raised when you try to stop poll with message without poll

exception aiogram.utils.exceptions.**MessageIsNotAPoll**(*message=None*)

Will be raised when you try to stop poll with message without poll

exception aiogram.utils.exceptions.**ChatNotFound**(*message=None*)

exception aiogram.utils.exceptions.**ChatIdIsEmpty**(*message=None*)

exception aiogram.utils.exceptions.**InvalidUserId**(*message=None*)

exception aiogram.utils.exceptions.**ChatDescriptionIsNotModified**(*message=None*)

exception aiogram.utils.exceptions.**InvalidQueryID**(*message=None*)

exception aiogram.utils.exceptions.**InvalidPeerID**(*message=None*)

exception aiogram.utils.exceptions.**InvalidHTTPUrlContent**(*message=None*)

exception aiogram.utils.exceptions.**ButtonURLInvalid**(*message=None*)

exception aiogram.utils.exceptions.**URLHostIsEmpty**(*message=None*)

exception aiogram.utils.exceptions.**StartParamInvalid**(*message=None*)

exception aiogram.utils.exceptions.**ButtonDataInvalid**(*message=None*)

exception aiogram.utils.exceptions.**FileIsTooBig**(*message=None*)

exception aiogram.utils.exceptions.**WrongFileIdentifier**(*message=None*)

exception aiogram.utils.exceptions.**GroupDeactivated**(*message=None*)

exception aiogram.utils.exceptions.**PhotoAsInputFileRequired**(*message=None*)

Will be raised when you try to set chat photo from file ID.

exception aiogram.utils.exceptions.**InvalidStickersSet**(*message=None*)

exception aiogram.utils.exceptions.**NoStickerInRequest**(*message=None*)

exception aiogram.utils.exceptions.**ChatAdminRequired**(*message=None*)

exception aiogram.utils.exceptions.**NeedAdministratorRightsInTheChannel**(*message=None*)

exception aiogram.utils.exceptions.**NotEnoughRightsToPinMessage**(*message=None*)

exception aiogram.utils.exceptions.**MethodNotAvailableInPrivateChats**(*message=None*)

exception aiogram.utils.exceptions.**CantDemoteChatCreator**(*message=None*)

exception aiogram.utils.exceptions.**CantRestrictSelf**(*message=None*)

exception aiogram.utils.exceptions.**NotEnoughRightsToRestrict**(*message=None*)

exception aiogram.utils.exceptions.**PhotoDimensions**(*message=None*)

exception aiogram.utils.exceptions.**UnavailableMembers**(*message=None*)

exception aiogram.utils.exceptions.**TypeOfFileMismatch**(*message=None*)

exception aiogram.utils.exceptions.**WrongRemoteFileIdSpecified**(*message=None*)

exception aiogram.utils.exceptions.**PaymentProviderInvalid**(*message=None*)

exception aiogram.utils.exceptions.**CurrencyTotalAmountInvalid**(*message=None*)

exception aiogram.utils.exceptions.**BadWebhook**(*message=None*)

exception aiogram.utils.exceptions.**WebhookRequireHTTPS**(*message=None*)

exception aiogram.utils.exceptions.**BadWebhookPort**(*message=None*)

exception aiogram.utils.exceptions.**BadWebhookAddrInfo**(*message=None*)

exception aiogram.utils.exceptions.**BadWebhookNoAddressAssociatedWithHostname**(*message=None*)

exception aiogram.utils.exceptions.**CantParseUrl**(*message=None*)

exception aiogram.utils.exceptions.**UnsupportedUrlProtocol**(*message=None*)

exception aiogram.utils.exceptions.**CantParseEntities**(*message=None*)

exception aiogram.utils.exceptions.**ResultIdDuplicate**(*message=None*)

exception aiogram.utils.exceptions.**BotDomainInvalid**(*message=None*)

exception aiogram.utils.exceptions.**MethodIsNotAvailable**(*message=None*)

exception aiogram.utils.exceptions.**CantRestrictChatOwner**(*message=None*)

Raises when bot restricts the chat owner

exception aiogram.utils.exceptions.**UserIsAnAdministratorOfTheChat**(*message=None*)

Raises when bot restricts the chat admin

exception aiogram.utils.exceptions.**NotFound**(*message=None*)

exception aiogram.utils.exceptions.**MethodNotKnown**(*message=None*)

exception aiogram.utils.exceptions.**ConflictError**(*message=None*)

exception aiogram.utils.exceptions.**TerminatedByOtherGetUpdates**(*message=None*)

exception aiogram.utils.exceptions.**CantGetUpdates**(*message=None*)

exception aiogram.utils.exceptions.**Unauthorized**(*message=None*)

exception aiogram.utils.exceptions.**BotKicked**(*message=None*)

exception aiogram.utils.exceptions.**BotBlocked**(*message=None*)

exception aiogram.utils.exceptions.**UserDeactivated**(*message=None*)

exception aiogram.utils.exceptions.**CantInitiateConversation**(*message=None*)

exception aiogram.utils.exceptions.**CantTalkWithBots**(*message=None*)

exception aiogram.utils.exceptions.**NetworkError**(*message=None*)

exception aiogram.utils.exceptions.**RestartingTelegram**

exception aiogram.utils.exceptions.**RetryAfter**(*retry_after*)

exception aiogram.utils.exceptions.**MigrateToChat**(*chat_id*)

exception aiogram.utils.exceptions.**Throttled**(***kwargs*)

4.6.4 Markdown

aiogram.utils.markdown.**quote_html**(**content*, *sep=' '*) → str

Quote HTML symbols

All <, >, & and " symbols that are not a part of a tag or an HTML entity must be replaced with the corresponding HTML entities (< with < > with > & with &amp and " with &quot).

Parameters

- **content** –
- **sep** –

Returns

aiogram.utils.markdown.**escape_md**(**content*, *sep=' '*) → str

Escape markdown text

E.g. for usernames

Parameters

- **content** –
- **sep** –

Returns

aiogram.utils.markdown.**text**(**content*, *sep=' '*)

Join all elements with a separator

Parameters

- **content** –
- **sep** –

Returns

aiogram.utils.markdown.**bold**(**content*, *sep=' '*) → str

Make bold text (Markdown)

Parameters

- **content** –
- **sep** –

Returns

aiogram.utils.markdown.**hbold**(**content*, *sep=' '*) → str

Make bold text (HTML)

Parameters

- **content** –
- **sep** –

Returns

`aiogram.utils.markdown.italic(*content, sep=' ') → str`

Make italic text (Markdown)

Parameters

- **content** –
- **sep** –

Returns

`aiogram.utils.markdown.hitalic(*content, sep=' ') → str`

Make italic text (HTML)

Parameters

- **content** –
- **sep** –

Returns

`aiogram.utils.markdown.spoiler(*content, sep=' ') → str`

Make spoiler text (Markdown)

Parameters

- **content** –
- **sep** –

Returns

`aiogram.utils.markdown.hspoiler(*content, sep=' ') → str`

Make spoiler text (HTML)

Parameters

- **content** –
- **sep** –

Returns

`aiogram.utils.markdown.code(*content, sep=' ') → str`

Make mono-width text (Markdown)

Parameters

- **content** –
- **sep** –

Returns

`aiogram.utils.markdown.hcode(*content, sep=' ') → str`

Make mono-width text (HTML)

Parameters

- **content** –
- **sep** –

Returns

`aiogram.utils.markdown.pre(*content, sep='\n') → str`

Make mono-width text block (Markdown)

Parameters

- **content** –
- **sep** –

Returns

`aiogram.utils.markdown.hpre(*content, sep=\n')` → str

Make mono-width text block (HTML)

Parameters

- **content** –
- **sep** –

Returns

`aiogram.utils.markdown.underline(*content, sep=' ')` → str

Make underlined text (Markdown)

Parameters

- **content** –
- **sep** –

Returns

`aiogram.utils.markdown.hunderline(*content, sep=' ')` → str

Make underlined text (HTML)

Parameters

- **content** –
- **sep** –

Returns

`aiogram.utils.markdown.strikethrough(*content, sep=' ')` → str

Make strikethrough text (Markdown)

Parameters

- **content** –
- **sep** –

Returns

`aiogram.utils.markdown.hstrikethrough(*content, sep=' ')` → str

Make strikethrough text (HTML)

Parameters

- **content** –
- **sep** –

Returns

`aiogram.utils.markdown.link(title: str, url: str)` → str

Format URL (Markdown)

Parameters

- **title** –
- **url** –

Returns

`aiogram.utils.markdown.hlink(title: str, url: str)` → str

Format URL (HTML)

Parameters

- **title** –
- **url** –

Returns

aiogram.utils.markdown.**hide_link**(*url: str*) → str

Hide URL (HTML only) Can be used for adding an image to a text message

Parameters

url –

Returns

4.6.5 Helper

Example:

```
>>> from aiogram.utils.helper import Helper, ListItem, HelperMode, Item
>>> class MyHelper(Helper):
...     mode = HelperMode.lowerCamelCase
...     FOO_ITEM = ListItem()
...     BAR_ITEM = ListItem()
...     BAZ_ITEM = ListItem()
...     LOREM = Item()
...
>>> print(MyHelper.FOO_ITEM & MyHelper.BAR_ITEM)
<<<  ['fooItem', 'barItem']
>>> print(MyHelper.all())
<<<  ['barItem', 'bazItem', 'fooItem', 'lorem']
```

class aiogram.utils.helper.**Item**(*value=None*)

Helper item

If a value is not provided, it will be automatically generated based on a variable's name

class aiogram.utils.helper.**ListItem**(*value=None*)

This item is always a list

You can use &, | and + operators for that.

class aiogram.utils.helper.**ItemsList**(**seq*)

Patch for default list

This class provides +, &, |, +=, &=, |= operators for extending the list

4.6.6 Deprecated

aiogram.utils.deprecated.**deprecated**(*reason*, *stacklevel=2*) → Callable

This is a decorator which can be used to mark functions as deprecated. It will result in a warning being emitted when the function is used.

Source: https://stackoverflow.com/questions/2536307/decorators-in-the-python-standard-lib-deprecated-specifically

aiogram.utils.deprecated.**renamed_argument**(*old_name: str*, *new_name: str*, *until_version: str*, *stacklevel: int = 3*)

A meta-decorator to mark an argument as deprecated.

```
@renamed_argument("chat", "chat_id", "3.0")  # stacklevel=3 by default
@renamed_argument("user", "user_id", "3.0", stacklevel=4)
def some_function(user_id, chat_id=None):
    print(f"user_id={user_id}, chat_id={chat_id}")
```

(continues on next page)

(continued from previous page)

```
some_function(user=123)  #  prints 'user_id=123, chat_id=None' with warning
some_function(123)  #  prints 'user_id=123, chat_id=None' without warning
some_function(user_id=123)  #  prints 'user_id=123, chat_id=None' without warning
```

Parameters

- **old_name** –
- **new_name** –
- **until_version** – the version in which the argument is scheduled to be removed
- **stacklevel** – leave it to default if it's the first decorator used.

Increment with any new decorator used. :return: decorator

aiogram.utils.deprecated.**removed_argument**(*name: str*, *until_version: str*, *stacklevel: int = 3*)

A meta-decorator to mark an argument as removed.

```
@removed_argument("until_date", "3.0")  # stacklevel=3 by default
def some_function(user_id, chat_id=None):
    print(f"user_id={user_id}, chat_id={chat_id}")
```

Parameters

- **name** –
- **until_version** – the version in which the argument is scheduled to be removed
- **stacklevel** – leave it to default if it's the first decorator used.

Increment with any new decorator used. :return: decorator

class aiogram.utils.deprecated.**DeprecatedReadOnlyClassVar**(*warning_message: str*, *new_value_getter: Callable[[_OwnerCls], _VT]*)

DeprecatedReadOnlyClassVar[Owner, ValueType]

Parameters

- **warning_message** – Warning message when getter gets called
- **new_value_getter** – Any callable with (owner_class: Type[Owner]) -> ValueType signature that will be executed

Usage example:

```
>>> class MyClass:
...     some_attribute: DeprecatedReadOnlyClassVar[MyClass, int] =     ...
↪  DeprecatedReadOnlyClassVar(
...                  "Warning message.", lambda owner: 15)
...
>>> MyClass.some_attribute  # does warning.warn with `Warning message` and returns
↪15 in the current case
```

4.6.7 Payload

`aiogram.utils.payload.generate_payload(exclude=None, **kwargs)`

Generate payload

Usage: payload = generate_payload(**locals(), exclude=['foo'])

Parameters

- **exclude** –
- **kwargs** –

Returns

dict

`aiogram.utils.payload.prepare_arg(value)`

Stringify dicts/lists and convert datetime/timedelta to unix-time

Parameters

value –

Returns

4.6.8 Parts

`aiogram.utils.parts.split_text(text: str, length: int = 4096) → List[str]`

Split long text

Parameters

- **text** –
- **length** –

Returns

list of parts

Return type

`typing.List[str]`

`aiogram.utils.parts.safe_split_text(text: str, length: int = 4096, split_separator: str = ' ') → List[str]`

Split long text

Parameters

- **text** –
- **length** –

:param split_separator :return:

`aiogram.utils.parts.paginate(data: Iterable, page: int = 0, limit: int = 10) → Iterable`

Slice data over pages

Parameters

- **data** (`typing.Iterable`) – any iterable object
- **page** (`int`) – number of page
- **limit** (`int`) – items per page

Returns

sliced object

Return type

`typing.Iterable`

4.6.9 JSON

4.6.10 Emoji

4.6.11 Deep linking

Deep linking

Telegram bots have a deep linking mechanism, that allows for passing additional parameters to the bot on startup. It could be a command that launches the bot — or an auth token to connect the user's Telegram account to their account on some external service.

You can read detailed description in the source: https://core.telegram.org/bots#deep-linking

We have add some utils to get deep links more handy.

Basic link example:

```
from aiogram.utils.deep_linking import get_start_link
link = await get_start_link('foo')

# result: 'https://t.me/MyBot?start=foo'
```

Encoded link example:

```
from aiogram.utils.deep_linking import get_start_link

link = await get_start_link('foo', encode=True)
# result: 'https://t.me/MyBot?start=Zm9v'
```

Decode it back example:

```
from aiogram.utils.deep_linking import decode_payload
from aiogram.types import Message

@dp.message_handler(commands=["start"])
async def handler(message: Message):
    args = message.get_args()
    payload = decode_payload(args)
    await message.answer(f"Your payload: {payload}")
```

async aiogram.utils.deep_linking.**get_start_link**(*payload: str*, *encode=False*) → str

Get 'start' deep link with your payload.

If you need to encode payload or pass special characters -

set encode as True

Parameters

- **payload** – args passed with /start
- **encode** – encode payload with base64url

Returns

link

async aiogram.utils.deep_linking.**get_startgroup_link**(*payload: str*, *encode=False*) → str

Get 'startgroup' deep link with your payload.

If you need to encode payload or pass special characters -

set encode as True

Parameters

- **payload** – args passed with /start
- **encode** – encode payload with base64url

Returns

link

`aiogram.utils.deep_linking.encode_payload(payload: str) → str`

Encode payload with URL-safe base64url.

`aiogram.utils.deep_linking.decode_payload(payload: str) → str`

Decode payload with URL-safe base64url.

4.7 Examples

4.7.1 Echo bot

Listing 1: echo_bot.py

```
"""
This is a echo bot.
It echoes any incoming text messages.
"""

import logging

from aiogram import Bot, Dispatcher, executor, types

API_TOKEN = 'BOT TOKEN HERE'

# Configure logging
logging.basicConfig(level=logging.INFO)

# Initialize bot and dispatcher
bot = Bot(token=API_TOKEN)
dp = Dispatcher(bot)


@dp.message_handler(commands=['start', 'help'])
async def send_welcome(message: types.Message):
    """
    This handler will be called when user sends `/start` or `/help` command
    """
    await message.reply("Hi!\nI'm EchoBot!\nPowered by aiogram.")


@dp.message_handler(regexp='(^cat[s]?$|puss)')
async def cats(message: types.Message):
    with open('data/cats.jpg', 'rb') as photo:
        '''
        # Old fashioned way:
        await bot.send_photo(
```

(continues on next page)

(continued from previous page)

```
            message.chat.id,
            photo,
            caption='Cats are here ',
            reply_to_message_id=message.message_id,
        )
        '''

        await message.reply_photo(photo, caption='Cats are here ')


@dp.message_handler()
async def echo(message: types.Message):
    # old style:
    # await bot.send_message(message.chat.id, message.text)

    await message.answer(message.text)


if __name__ == '__main__':
    executor.start_polling(dp, skip_updates=True)
```

4.7.2 Inline bot

Listing 2: inline_bot.py

```
import hashlib
import logging

from aiogram import Bot, Dispatcher, executor
from aiogram.types import InlineQuery, \
    InputTextMessageContent, InlineQueryResultArticle

API_TOKEN = 'BOT_TOKEN_HERE'

logging.basicConfig(level=logging.DEBUG)

bot = Bot(token=API_TOKEN)
dp = Dispatcher(bot)


@dp.inline_handler()
async def inline_echo(inline_query: InlineQuery):
    # id affects both preview and content,
    # so it has to be unique for each result
    # (Unique identifier for this result, 1-64 Bytes)
    # you can set your unique id's
    # but for example i'll generate it based on text because I know, that
    # only text will be passed in this example
    text = inline_query.query or 'echo'
    input_content = InputTextMessageContent(text)
```

(continues on next page)

(continued from previous page)

```
    result_id: str = hashlib.md5(text.encode()).hexdigest()
    item = InlineQueryResultArticle(
        id=result_id,
        title=f'Result {text!r}',
        input_message_content=input_content,
    )
    # don't forget to set cache_time=1 for testing (default is 300s or 5m)
    await bot.answer_inline_query(inline_query.id, results=[item], cache_time=1)


if __name__ == '__main__':
    executor.start_polling(dp, skip_updates=True)
```

4.7.3 Advanced executor example

Listing 3: advanced_executor_example.py

```
#!/usr/bin/env python3
"""
**This example is outdated**
In this example used ArgumentParser for configuring Your bot.

Provided to start bot with webhook:
    python advanced_executor_example.py \
        --token TOKEN_HERE \
        --host 0.0.0.0 \
        --port 8084 \
        --host-name example.com \
        --webhook-port 443

Or long polling:
    python advanced_executor_example.py --token TOKEN_HERE

So... In this example found small trouble:
    can't get bot instance in handlers.


If you want to automatic change getting updates method use executor utils (from aiogram.
↪utils.executor)
"""
# TODO: Move token to environment variables.

import argparse
import logging
import ssl
import sys

from aiogram import Bot
from aiogram.dispatcher import Dispatcher
from aiogram.dispatcher.webhook import *
```

(continues on next page)

(continued from previous page)

```
from aiogram.utils.executor import start_polling, start_webhook

logging.basicConfig(level=logging.INFO)

# Configure arguments parser.
parser = argparse.ArgumentParser(description='Python telegram bot')
parser.add_argument('--token', '-t', nargs='?', type=str, default=None, help='Set␣
↪working directory')
parser.add_argument('--sock', help='UNIX Socket path')
parser.add_argument('--host', help='Webserver host')
parser.add_argument('--port', type=int, help='Webserver port')
parser.add_argument('--cert', help='Path to SSL certificate')
parser.add_argument('--pkey', help='Path to SSL private key')
parser.add_argument('--host-name', help='Set webhook host name')
parser.add_argument('--webhook-port', type=int, help='Port for webhook (default=port)')
parser.add_argument('--webhook-path', default='/webhook', help='Port for webhook␣
↪(default=port)')


async def cmd_start(message: types.Message):
    return SendMessage(message.chat.id, f"Hello, {message.from_user.full_name}!")


def setup_handlers(dispatcher: Dispatcher):
    # This example has only one messages handler
    dispatcher.register_message_handler(cmd_start, commands=['start', 'welcome'])


async def on_startup(dispatcher, url=None, cert=None):
    setup_handlers(dispatcher)

    bot = dispatcher.bot

    # Get current webhook status
    webhook = await bot.get_webhook_info()

    if url:
        # If URL is bad
        if webhook.url != url:
            # If URL doesnt match with by current remove webhook
            if not webhook.url:
                await bot.delete_webhook()

            # Set new URL for webhook
            if cert:
                with open(cert, 'rb') as cert_file:
                    await bot.set_webhook(url, certificate=cert_file)
            else:
                await bot.set_webhook(url)
    elif webhook.url:
        # Otherwise remove webhook.
        await bot.delete_webhook()
```

(continues on next page)

(continued from previous page)

```


async def on_shutdown(dispatcher):
    print('Shutdown.')


def main(arguments):
    args = parser.parse_args(arguments)
    token = args.token
    sock = args.sock
    host = args.host
    port = args.port
    cert = args.cert
    pkey = args.pkey
    host_name = args.host_name or host
    webhook_port = args.webhook_port or port
    webhook_path = args.webhook_path

    # Fi webhook path
    if not webhook_path.startswith('/'):
        webhook_path = '/' + webhook_path

    # Generate webhook URL
    webhook_url = f"https://{host_name}:{webhook_port}{webhook_path}"

    # Create bot & dispatcher instances.
    bot = Bot(token)
    dispatcher = Dispatcher(bot)

    if (sock or host) and host_name:
        if cert and pkey:
            ssl_context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)
            ssl_context.load_cert_chain(cert, pkey)
        else:
            ssl_context = None

        start_webhook(dispatcher, webhook_path,
                      on_startup=functools.partial(on_startup, url=webhook_url,␣
↪cert=cert),
                      on_shutdown=on_shutdown,
                      host=host, port=port, path=sock, ssl_context=ssl_context)
    else:
        start_polling(dispatcher, on_startup=on_startup, on_shutdown=on_shutdown)


if __name__ == '__main__':
    argv = sys.argv[1:]

    if not len(argv):
        parser.print_help()
        sys.exit(1)

```

(continues on next page)

(continued from previous page)

```
    main(argv)
```

4.7.4 Proxy and emojize

Listing 4: proxy_and_emojize.py

```
import logging

import aiohttp

from aiogram import Bot, types
from aiogram.dispatcher import Dispatcher
from aiogram.types import ParseMode
from aiogram.utils.emoji import emojize
from aiogram.utils.executor import start_polling
from aiogram.utils.markdown import bold, code, italic, text

# Configure bot here
API_TOKEN = 'BOT_TOKEN_HERE'
PROXY_URL = 'http://PROXY_URL'  # Or 'socks5://host:port'

# NOTE: If authentication is required in your proxy then uncomment next line and change␣
↪login/password for it
# PROXY_AUTH = aiohttp.BasicAuth(login='login', password='password')
# And add `proxy_auth=PROXY_AUTH` argument in line 30, like this:
# >>> bot = Bot(token=API_TOKEN, proxy=PROXY_URL, proxy_auth=PROXY_AUTH)
# Also you can use Socks5 proxy but you need manually install aiohttp_socks package.

# Get my ip URL
GET_IP_URL = 'http://bot.whatismyipaddress.com/'

logging.basicConfig(level=logging.INFO)

bot = Bot(token=API_TOKEN, proxy=PROXY_URL)

# If auth is required:
# bot = Bot(token=API_TOKEN, proxy=PROXY_URL, proxy_auth=PROXY_AUTH)
dp = Dispatcher(bot)


async def fetch(url, session):
    async with session.get(url) as response:
        return await response.text()


@dp.message_handler(commands=['start'])
async def cmd_start(message: types.Message):
    # fetching urls will take some time, so notify user that everything is OK
    await types.ChatActions.typing()

```

(continues on next page)

(continued from previous page)

```
    content = []

    # Make request (without proxy)
    async with aiohttp.ClientSession() as session:
        ip = await fetch(GET_IP_URL, session)
    content.append(text(':globe_showing_Americas:', bold('IP:'), code(ip)))
    # This line is formatted to ' *IP:* `YOUR IP`'

    # Make request through bot's proxy
    ip = await fetch(GET_IP_URL, await bot.get_session())
    content.append(text(':locked_with_key:', bold('IP:'), code(ip), italic('via proxy')))
    # This line is formatted to ' *IP:* `YOUR IP` _via proxy_'

    # Send content
    await bot.send_message(message.chat.id, emojize(text(*content, sep='\n')), parse_
↪mode=ParseMode.MARKDOWN)

    # In this example you can see emoji codes: ":globe_showing_Americas:" and ":locked_
↪with_key:"
    # You can find full emoji cheat sheet at https://www.webpagefx.com/tools/emoji-cheat-
↪sheet/
    # For representing emoji codes into real emoji use emoji util (aiogram.utils.emoji)
    # (you have to install emoji module)

    # For example emojize('Moon face :new_moon_face:') is transformed to 'Moon face '


if __name__ == '__main__':
    start_polling(dp, skip_updates=True)
```

4.7.5 Finite state machine example

Listing 5: finite_state_machine_example.py

```
import logging

import aiogram.utils.markdown as md
from aiogram import Bot, Dispatcher, types
from aiogram.contrib.fsm_storage.memory import MemoryStorage
from aiogram.dispatcher import FSMContext
from aiogram.dispatcher.filters import Text
from aiogram.dispatcher.filters.state import State, StatesGroup
from aiogram.types import ParseMode
from aiogram.utils import executor

logging.basicConfig(level=logging.INFO)

API_TOKEN = 'BOT TOKEN HERE'


```

(continues on next page)

(continued from previous page)

```
bot = Bot(token=API_TOKEN)

# For example use simple MemoryStorage for Dispatcher.
storage = MemoryStorage()
dp = Dispatcher(bot, storage=storage)


# States
class Form(StatesGroup):
    name = State()  # Will be represented in storage as 'Form:name'
    age = State()  # Will be represented in storage as 'Form:age'
    gender = State()  # Will be represented in storage as 'Form:gender'


@dp.message_handler(commands='start')
async def cmd_start(message: types.Message):
    """
    Conversation's entry point
    """
    # Set state
    await Form.name.set()

    await message.reply("Hi there! What's your name?")


# You can use state '*' if you need to handle all states
@dp.message_handler(state='*', commands='cancel')
@dp.message_handler(Text(equals='cancel', ignore_case=True), state='*')
async def cancel_handler(message: types.Message, state: FSMContext):
    """
    Allow user to cancel any action
    """
    current_state = await state.get_state()
    if current_state is None:
        return

    logging.info('Cancelling state %r', current_state)
    # Cancel state and inform user about it
    await state.finish()
    # And remove keyboard (just in case)
    await message.reply('Cancelled.', reply_markup=types.ReplyKeyboardRemove())


@dp.message_handler(state=Form.name)
async def process_name(message: types.Message, state: FSMContext):
    """
    Process user name
    """
    async with state.proxy() as data:
        data['name'] = message.text

    await Form.next()
```

(continues on next page)

(continued from previous page)

```
    await message.reply("How old are you?")


# Check age. Age gotta be digit
@dp.message_handler(lambda message: not message.text.isdigit(), state=Form.age)
async def process_age_invalid(message: types.Message):
    """
    If age is invalid
    """
    return await message.reply("Age gotta be a number.\nHow old are you? (digits only)")


@dp.message_handler(lambda message: message.text.isdigit(), state=Form.age)
async def process_age(message: types.Message, state: FSMContext):
    # Update state and data
    await Form.next()
    await state.update_data(age=int(message.text))

    # Configure ReplyKeyboardMarkup
    markup = types.ReplyKeyboardMarkup(resize_keyboard=True, selective=True)
    markup.add("Male", "Female")
    markup.add("Other")

    await message.reply("What is your gender?", reply_markup=markup)


@dp.message_handler(lambda message: message.text not in ["Male", "Female", "Other"],␣
↪state=Form.gender)
async def process_gender_invalid(message: types.Message):
    """
    In this example gender has to be one of: Male, Female, Other.
    """
    return await message.reply("Bad gender name. Choose your gender from the keyboard.")


@dp.message_handler(state=Form.gender)
async def process_gender(message: types.Message, state: FSMContext):
    async with state.proxy() as data:
        data['gender'] = message.text

        # Remove keyboard
        markup = types.ReplyKeyboardRemove()

        # And send message
        await bot.send_message(
            message.chat.id,
            md.text(
                md.text('Hi! Nice to meet you,', md.bold(data['name'])),
                md.text('Age:', md.code(data['age'])),
                md.text('Gender:', data['gender']),
                sep='\n',
            ),
```

(continues on next page)

(continued from previous page)

```
            reply_markup=markup,
            parse_mode=ParseMode.MARKDOWN,
        )

    # Finish conversation
    await state.finish()


if __name__ == '__main__':
    executor.start_polling(dp, skip_updates=True)
```

4.7.6 Throttling example

Listing 6: throttling_example.py

```
"""
Example for throttling manager.

You can use that for flood controlling.
"""

import logging

from aiogram import Bot, types
from aiogram.contrib.fsm_storage.memory import MemoryStorage
from aiogram.dispatcher import Dispatcher
from aiogram.utils.exceptions import Throttled
from aiogram.utils.executor import start_polling


API_TOKEN = 'BOT_TOKEN_HERE'

logging.basicConfig(level=logging.INFO)

bot = Bot(token=API_TOKEN)

# Throttling manager does not work without Leaky Bucket.
# You need to use a storage. For example use simple in-memory storage.
storage = MemoryStorage()
dp = Dispatcher(bot, storage=storage)


@dp.message_handler(commands=['start'])
async def send_welcome(message: types.Message):
    try:
        # Execute throttling manager with rate-limit equal to 2 seconds for key "start"
        await dp.throttle('start', rate=2)
    except Throttled:
        # If request is throttled, the `Throttled` exception will be raised
        await message.reply('Too many requests!')
```

(continues on next page)

(continued from previous page)

```
    else:
        # Otherwise do something
        await message.reply("Hi!\nI'm EchoBot!\nPowered by aiogram.")


@dp.message_handler(commands=['hi'])
@dp.throttled(lambda msg, loop, *args, **kwargs: loop.create_task(bot.send_message(msg.
↪from_user.id, "Throttled")),
              rate=5)
# loop is added to the function to run coroutines from it
async def say_hi(message: types.Message):
    await message.answer("Hi")


# the on_throttled object can be either a regular function or coroutine
async def hello_throttled(*args, **kwargs):
    # args will be the same as in the original handler
    # kwargs will be updated with parameters given to .throttled (rate, key, user_id,␣
↪chat_id)
    print(f"hello_throttled was called with args={args} and kwargs={kwargs}")
    message = args[0]  # as message was the first argument in the original handler
    await message.answer("Throttled")


@dp.message_handler(commands=['hello'])
@dp.throttled(hello_throttled, rate=4)
async def say_hello(message: types.Message):
    await message.answer("Hello!")


@dp.message_handler(commands=['help'])
@dp.throttled(rate=5)
# nothing will happen if the handler will be throttled
async def help_handler(message: types.Message):
    await message.answer('Help!')

if __name__ == '__main__':
    start_polling(dp, skip_updates=True)
```

4.7.7 I18n example

Listing 7: i18n_example.py

```
"""
Internationalize your bot

Step 1: extract texts
    # pybabel extract --input-dirs=. -o locales/mybot.pot

    Some useful options:
```

(continues on next page)

(continued from previous page)

```
    - Extract texts with pluralization support
    # -k __:1,2
    - Add comments for translators, you can use another tag if you want (TR)
    # --add-comments=NOTE
    - Disable comments with string location in code
    # --no-location
    - Set project name
    # --project=MySuperBot
    - Set version
    # --version=2.2

Step 2: create *.po files. E.g. create en, ru, uk locales.
    # pybabel init -i locales/mybot.pot -d locales -D mybot -l en
    # pybabel init -i locales/mybot.pot -d locales -D mybot -l ru
    # pybabel init -i locales/mybot.pot -d locales -D mybot -l uk

Step 3: translate texts located in locales/{language}/LC_MESSAGES/mybot.po
    To open .po file you can use basic text editor or any PO editor, e.g. https://poedit.
↪net/

Step 4: compile translations
    # pybabel compile -d locales -D mybot

Step 5: When you change the code of your bot you need to update po & mo files.
    Step 5.1: regenerate pot file:
        command from step 1
    Step 5.2: update po files
        # pybabel update -d locales -D mybot -i locales/mybot.pot
    Step 5.3: update your translations
        location and tools you know from step 3
    Step 5.4: compile mo files
        command from step 4
"""

from pathlib import Path

from aiogram import Bot, Dispatcher, executor, types
from aiogram.contrib.middlewares.i18n import I18nMiddleware

TOKEN = 'BOT_TOKEN_HERE'
I18N_DOMAIN = 'mybot'

BASE_DIR = Path(__file__).parent
LOCALES_DIR = BASE_DIR / 'locales'

bot = Bot(TOKEN, parse_mode=types.ParseMode.HTML)
dp = Dispatcher(bot)

# Setup i18n middleware
i18n = I18nMiddleware(I18N_DOMAIN, LOCALES_DIR)
dp.middleware.setup(i18n)

```

(continues on next page)

(continued from previous page)

```
# Alias for gettext method
_ = i18n.gettext


@dp.message_handler(commands='start')
async def cmd_start(message: types.Message):
    # Simply use `_('message')` instead of `'message'` and never use f-strings for␣
↪translatable texts.
    await message.reply(_('Hello, <b>{user}</b>!').format(user=message.from_user.full_
↪name))


@dp.message_handler(commands='lang')
async def cmd_lang(message: types.Message, locale):
    # For setting custom lang you have to modify i18n middleware
    await message.reply(_('Your current language: <i>{language}</i>').
↪format(language=locale))

# If you care about pluralization, here's small handler
# And also, there's and example of comments for translators. Most translation tools␣
↪support them.

# Alias for gettext method, parser will understand double underscore as plural (aka␣
↪ngettext)
__ = i18n.gettext


# some likes manager
LIKES_STORAGE = {'count': 0}


def get_likes() -> int:
    return LIKES_STORAGE['count']


def increase_likes() -> int:
    LIKES_STORAGE['count'] += 1
    return get_likes()


@dp.message_handler(commands='like')
async def cmd_like(message: types.Message, locale):
    likes = increase_likes()

    # NOTE: This is comment for a translator
    await message.reply(__('Aiogram has {number} like!', 'Aiogram has {number} likes!',␣
↪likes).format(number=likes))


if __name__ == '__main__':
    executor.start_polling(dp, skip_updates=True)
```

4.7.8 Regexp commands filter example

Listing 8: regexp_commands_filter_example.py

```
from aiogram import Bot, types
from aiogram.dispatcher import Dispatcher, filters
from aiogram.utils import executor


bot = Bot(token='BOT_TOKEN_HERE', parse_mode=types.ParseMode.HTML)
dp = Dispatcher(bot)


@dp.message_handler(filters.RegexpCommandsFilter(regexp_commands=['item_([0-9]*)']))
async def send_welcome(message: types.Message, regexp_command):
    await message.reply(f"You have requested an item with id <code>{regexp_command.
↪group(1)}</code>")


@dp.message_handler(commands='start')
async def create_deeplink(message: types.Message):
    bot_user = await bot.me
    bot_username = bot_user.username
    deeplink = f'https://t.me/{bot_username}?start=item_12345'
    text = (
        f'Either send a command /item_1234 or follow this link {deeplink} and then click␣
↪start\n'
        'It also can be hidden in a inline button\n\n'
        'Or just send <code>/start item_123</code>'
    )
    await message.reply(text, disable_web_page_preview=True)


if __name__ == '__main__':
    executor.start_polling(dp, skip_updates=True)
```

4.7.9 Check user language

Babel is required.

Listing 9: check_user_language.py

```
"""
Babel is required.
"""

import logging

from aiogram import Bot, Dispatcher, executor, md, types

API_TOKEN = 'BOT TOKEN HERE'

```

(continues on next page)

(continued from previous page)

```
logging.basicConfig(level=logging.INFO)


bot = Bot(token=API_TOKEN, parse_mode=types.ParseMode.MARKDOWN_V2)
dp = Dispatcher(bot)


@dp.message_handler()
async def check_language(message: types.Message):
    locale = message.from_user.locale

    await message.reply(md.text(
        md.bold('Info about your language:'),
        md.text('', md.bold('Code:'), md.code(locale.language)),
        md.text('', md.bold('Territory:'), md.code(locale.territory or 'Unknown')),
        md.text('', md.bold('Language name:'), md.code(locale.language_name)),
        md.text('', md.bold('English language name:'), md.code(locale.english_name)),
        sep='\n',
    ))


if __name__ == '__main__':
    executor.start_polling(dp, skip_updates=True)
```

4.7.10 Middleware and antiflood

Listing 10: middleware_and_antiflood.py

```
import asyncio

from aiogram import Bot, Dispatcher, executor, types
from aiogram.contrib.fsm_storage.redis import RedisStorage2
from aiogram.dispatcher import DEFAULT_RATE_LIMIT
from aiogram.dispatcher.handler import CancelHandler, current_handler
from aiogram.dispatcher.middlewares import BaseMiddleware
from aiogram.utils.exceptions import Throttled

TOKEN = 'BOT_TOKEN_HERE'

# In this example Redis storage is used
storage = RedisStorage2(db=5)

bot = Bot(token=TOKEN)
dp = Dispatcher(bot, storage=storage)


def rate_limit(limit: int, key=None):
    """
    Decorator for configuring rate limit and key in different functions.

```

(continues on next page)

(continued from previous page)

```
    :param limit:
    :param key:
    :return:
    """

    def decorator(func):
        setattr(func, 'throttling_rate_limit', limit)
        if key:
            setattr(func, 'throttling_key', key)
        return func

    return decorator


class ThrottlingMiddleware(BaseMiddleware):
    """
    Simple middleware
    """

    def __init__(self, limit=DEFAULT_RATE_LIMIT, key_prefix='antiflood_'):
        self.rate_limit = limit
        self.prefix = key_prefix
        super(ThrottlingMiddleware, self).__init__()

    async def on_process_message(self, message: types.Message, data: dict):
        """
        This handler is called when dispatcher receives a message

        :param message:
        """
        # Get current handler
        handler = current_handler.get()

        # Get dispatcher from context
        dispatcher = Dispatcher.get_current()
        # If handler was configured, get rate limit and key from handler
        if handler:
            limit = getattr(handler, 'throttling_rate_limit', self.rate_limit)
            key = getattr(handler, 'throttling_key', f"{self.prefix}_{handler.__name__}")
        else:
            limit = self.rate_limit
            key = f"{self.prefix}_message"

        # Use Dispatcher.throttle method.
        try:
            await dispatcher.throttle(key, rate=limit)
        except Throttled as t:
            # Execute action
            await self.message_throttled(message, t)

            # Cancel current handler
            raise CancelHandler()
```

(continues on next page)

(continued from previous page)

```

    async def message_throttled(self, message: types.Message, throttled: Throttled):
        """
        Notify user only on first exceed and notify about unlocking only on last exceed

        :param message:
        :param throttled:
        """
        handler = current_handler.get()
        dispatcher = Dispatcher.get_current()
        if handler:
            key = getattr(handler, 'throttling_key', f"{self.prefix}_{handler.__name__}")
        else:
            key = f"{self.prefix}_message"

        # Calculate how many time is left till the block ends
        delta = throttled.rate - throttled.delta

        # Prevent flooding
        if throttled.exceeded_count <= 2:
            await message.reply('Too many requests! ')

        # Sleep.
        await asyncio.sleep(delta)

        # Check lock status
        thr = await dispatcher.check_key(key)

        # If current message is not last with current key - do not send message
        if thr.exceeded_count == throttled.exceeded_count:
            await message.reply('Unlocked.')


@dp.message_handler(commands=['start'])
@rate_limit(5, 'start')  # this is not required but you can configure throttling manager␣
↪for current handler using it
async def cmd_test(message: types.Message):
    # You can use this command every 5 seconds
    await message.reply('Test passed! You can use this command every 5 seconds.')


if __name__ == '__main__':
    # Setup middleware
    dp.middleware.setup(ThrottlingMiddleware())

    # Start long-polling
    executor.start_polling(dp)
```

4.7.11 Webhook example

Listing 11: webhook_example.py

```
import logging

from aiogram import Bot, types
from aiogram.contrib.middlewares.logging import LoggingMiddleware
from aiogram.dispatcher import Dispatcher
from aiogram.dispatcher.webhook import SendMessage
from aiogram.utils.executor import start_webhook


API_TOKEN = 'BOT_TOKEN_HERE'

# webhook settings
WEBHOOK_HOST = 'https://your.domain'
WEBHOOK_PATH = '/path/to/api'
WEBHOOK_URL = f"{WEBHOOK_HOST}{WEBHOOK_PATH}"

# webserver settings
WEBAPP_HOST = 'localhost'  # or ip
WEBAPP_PORT = 3001

logging.basicConfig(level=logging.INFO)

bot = Bot(token=API_TOKEN)
dp = Dispatcher(bot)
dp.middleware.setup(LoggingMiddleware())


@dp.message_handler()
async def echo(message: types.Message):
    # Regular request
    # await bot.send_message(message.chat.id, message.text)

    # or reply INTO webhook
    return SendMessage(message.chat.id, message.text)


async def on_startup(dp):
    await bot.set_webhook(WEBHOOK_URL)
    # insert code here to run it after start


async def on_shutdown(dp):
    logging.warning('Shutting down..')

    # insert code here to run it before shutdown

    # Remove webhook (not acceptable in some cases)
    await bot.delete_webhook()

```

(continues on next page)

(continued from previous page)

```
    # Close DB connection (if used)
    await dp.storage.close()
    await dp.storage.wait_closed()

    logging.warning('Bye!')


if __name__ == '__main__':
    start_webhook(
        dispatcher=dp,
        webhook_path=WEBHOOK_PATH,
        on_startup=on_startup,
        on_shutdown=on_shutdown,
        skip_updates=True,
        host=WEBAPP_HOST,
        port=WEBAPP_PORT,
    )
```

4.7.12 Webhook example old

Listing 12: webhook_example_2.py

```
"""
Example outdated
"""

import asyncio
import ssl
import sys

from aiohttp import web

import aiogram
from aiogram import Bot, types
from aiogram.contrib.fsm_storage.memory import MemoryStorage
from aiogram.dispatcher import Dispatcher
from aiogram.dispatcher.webhook import get_new_configured_app, SendMessage
from aiogram.types import ChatType, ParseMode, ContentTypes
from aiogram.utils.markdown import hbold, bold, text, link

TOKEN = 'BOT TOKEN HERE'

WEBHOOK_HOST = 'example.com'  # Domain name or IP addres which your bot is located.
WEBHOOK_PORT = 443  # Telegram Bot API allows only for usage next ports: 443, 80, 88 or␣
↪8443
WEBHOOK_URL_PATH = '/webhook'  # Part of URL

# This options needed if you use self-signed SSL certificate
# Instructions: https://core.telegram.org/bots/self-signed
WEBHOOK_SSL_CERT = './webhook_cert.pem'  # Path to the ssl certificate
```

(continues on next page)

(continued from previous page)

```
WEBHOOK_SSL_PRIV = './webhook_pkey.pem'  # Path to the ssl private key

WEBHOOK_URL = f"https://{WEBHOOK_HOST}:{WEBHOOK_PORT}{WEBHOOK_URL_PATH}"

# Web app settings:
#   Use LAN address to listen webhooks
#   User any available port in range from 1024 to 49151 if you're using proxy, or WEBHOOK_
↪PORT if you're using direct webhook handling
WEBAPP_HOST = 'localhost'
WEBAPP_PORT = 3001

BAD_CONTENT = ContentTypes.PHOTO & ContentTypes.DOCUMENT & ContentTypes.STICKER &␣
↪ContentTypes.AUDIO

bot = Bot(TOKEN)
storage = MemoryStorage()
dp = Dispatcher(bot, storage=storage)


async def cmd_start(message: types.Message):
    # Yep. aiogram allows to respond into webhook.
    # https://core.telegram.org/bots/api#making-requests-when-getting-updates
    return SendMessage(chat_id=message.chat.id, text='Hi from webhook!',
                       reply_to_message_id=message.message_id)


async def cmd_about(message: types.Message):
    # In this function markdown utils are userd for formatting message text
    return SendMessage(message.chat.id, text(
        bold('Hi! I\'m just a simple telegram bot.'),
        '',
        text('I\'m powered by', bold('Python', Version(*sys.version_info[:]))),
        text('With', link(text('aiogram', aiogram.VERSION), 'https://github.com/aiogram/
↪aiogram')),
        sep='\n'
    ), parse_mode=ParseMode.MARKDOWN)


async def cancel(message: types.Message):
    # Get current state context
    state = dp.current_state(chat=message.chat.id, user=message.from_user.id)

    # If current user in any state - cancel it.
    if await state.get_state() is not None:
        await state.set_state(state=None)
        return SendMessage(message.chat.id, 'Current action is canceled.')
        # Otherwise do nothing


async def unknown(message: types.Message):
    """
    Handler for unknown messages.
```

(continues on next page)

(continued from previous page)

```
    """
    return SendMessage(message.chat.id,
                       f"I don\'t know what to do with content type `{message.content_type()}`. Sorry :c")


async def cmd_id(message: types.Message):
    """
    Return info about user.
    """
    if message.reply_to_message:
        target = message.reply_to_message.from_user
        chat = message.chat
    elif message.forward_from and message.chat.type == ChatType.PRIVATE:
        target = message.forward_from
        chat = message.forward_from or message.chat
    else:
        target = message.from_user
        chat = message.chat

    result_msg = [hbold('Info about user:'),
                  f"First name: {target.first_name}"]
    if target.last_name:
        result_msg.append(f"Last name: {target.last_name}")
    if target.username:
        result_msg.append(f"Username: {target.mention}")
    result_msg.append(f"User ID: {target.id}")

    result_msg.extend([hbold('Chat:'),
                       f"Type: {chat.type}",
                       f"Chat ID: {chat.id}"])
    if chat.type != ChatType.PRIVATE:
        result_msg.append(f"Title: {chat.title}")
    else:
        result_msg.append(f"Title: {chat.full_name}")
    return SendMessage(message.chat.id, '\n'.join(result_msg), reply_to_message_id=message.message_id,
                       parse_mode=ParseMode.HTML)


async def on_startup(app):
    # Demonstrate one of the available methods for registering handlers
    # This command available only in main state (state=None)
    dp.register_message_handler(cmd_start, commands=['start'])

    # This handler is available in all states at any time.
    dp.register_message_handler(cmd_about, commands=['help', 'about'], state='*')
    dp.register_message_handler(unknown, content_types=BAD_CONTENT,
                                func=lambda message: message.chat.type == ChatType.PRIVATE)

    # You are able to register one function handler for multiple conditions
```

(continues on next page)

(continued from previous page)

```
    dp.register_message_handler(cancel, commands=['cancel'], state='*')
    dp.register_message_handler(cancel, func=lambda message: message.text.lower().
↪strip() in ['cancel'], state='*')

    dp.register_message_handler(cmd_id, commands=['id'], state='*')
    dp.register_message_handler(cmd_id, func=lambda message: message.forward_from or
                                                             message.reply_to_message and
                                                             message.chat.type ==␣
↪ChatType.PRIVATE, state='*')

    # Get current webhook status
    webhook = await bot.get_webhook_info()

    # If URL is bad
    if webhook.url != WEBHOOK_URL:
        # If URL doesnt match current - remove webhook
        if not webhook.url:
            await bot.delete_webhook()

        # Set new URL for webhook
        await bot.set_webhook(WEBHOOK_URL, certificate=open(WEBHOOK_SSL_CERT, 'rb'))
        # If you want to use free certificate signed by LetsEncrypt you need to set only␣
↪URL without sending certificate.


async def on_shutdown(app):
    """
    Graceful shutdown. This method is recommended by aiohttp docs.
    """
    # Remove webhook.
    await bot.delete_webhook()

    # Close Redis connection.
    await dp.storage.close()
    await dp.storage.wait_closed()


if __name__ == '__main__':
    # Get instance of :class:`aiohttp.web.Application` with configured router.
    app = get_new_configured_app(dispatcher=dp, path=WEBHOOK_URL_PATH)

    # Setup event handlers.
    app.on_startup.append(on_startup)
    app.on_shutdown.append(on_shutdown)

    # Generate SSL context
    context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)
    context.load_cert_chain(WEBHOOK_SSL_CERT, WEBHOOK_SSL_PRIV)

    # Start web-application.
    web.run_app(app, host=WEBAPP_HOST, port=WEBAPP_PORT, ssl_context=context)
    # Note:
```

(continues on next page)

(continued from previous page)

```
    #   If you start your bot using nginx or Apache web server, SSL context is not␣
↪required.
    #   Otherwise you need to set `ssl_context` parameter.
```

4.7.13 Payments

Listing 13: payments.py

```
from aiogram import Bot
from aiogram import types
from aiogram.dispatcher import Dispatcher
from aiogram.types.message import ContentTypes
from aiogram.utils import executor


BOT_TOKEN = 'BOT_TOKEN_HERE'
PAYMENTS_PROVIDER_TOKEN = '123456789:TEST:1422'

bot = Bot(BOT_TOKEN)
dp = Dispatcher(bot)

# Setup prices
prices = [
    types.LabeledPrice(label='Working Time Machine', amount=5750),
    types.LabeledPrice(label='Gift wrapping', amount=500),
]

# Setup shipping options
shipping_options = [
    types.ShippingOption(id='instant', title='WorldWide Teleporter').add(types.
↪LabeledPrice('Teleporter', 1000)),
    types.ShippingOption(id='pickup', title='Local pickup').add(types.LabeledPrice(
↪'Pickup', 300)),
]


@dp.message_handler(commands=['start'])
async def cmd_start(message: types.Message):
    await bot.send_message(message.chat.id,
                           "Hello, I'm the demo merchant bot."
                           " I can sell you a Time Machine."
                           " Use /buy to order one, /terms for Terms and Conditions")


@dp.message_handler(commands=['terms'])
async def cmd_terms(message: types.Message):
    await bot.send_message(message.chat.id,
                           'Thank you for shopping with our demo bot. We hope you like␣
↪your new time machine!\n'
                           '1. If your time machine was not delivered on time, please␣
```

(continues on next page)

(continued from previous page)

```
↪rethink your concept of time'
                           ' and try again.\n'
                           '2. If you find that your time machine is not working, kindly␣
↪contact our future service'
                           ' workshops on Trappist-1e. They will be accessible anywhere␣
↪between'
                           ' May 2075 and November 4000 C.E.\n'
                           '3. If you would like a refund, kindly apply for one␣
↪yesterday and we will have sent it'
                           ' to you immediately.')


@dp.message_handler(commands=['buy'])
async def cmd_buy(message: types.Message):
    await bot.send_message(message.chat.id,
                           "Real cards won't work with me, no money will be debited from␣
↪your account."
                           " Use this test card number to pay for your Time Machine:␣
↪`4242 4242 4242 4242`"
                           "\n\nThis is your demo invoice:", parse_mode='Markdown')
    await bot.send_invoice(message.chat.id, title='Working Time Machine',
                           description='Want to visit your great-great-great-
↪grandparents?'
                                       ' Make a fortune at the races?'
                                       ' Shake hands with Hammurabi and take a stroll in␣
↪the Hanging Gardens?'
                                       ' Order our Working Time Machine today!',
                           provider_token=PAYMENTS_PROVIDER_TOKEN,
                           currency='usd',
                           photo_url='https://telegra.ph/file/d08ff863531f10bf2ea4b.jpg',
                           photo_height=512,  # !=0/None or picture won't be shown
                           photo_width=512,
                           photo_size=512,
                           is_flexible=True,  # True If you need to set up Shipping Fee
                           prices=prices,
                           start_parameter='time-machine-example',
                           payload='HAPPY FRIDAYS COUPON')


@dp.shipping_query_handler(lambda query: True)
async def shipping(shipping_query: types.ShippingQuery):
    await bot.answer_shipping_query(shipping_query.id, ok=True, shipping_
↪options=shipping_options,
                                    error_message='Oh, seems like our Dog couriers are␣
↪having a lunch right now.'
                                                  ' Try again later!')


@dp.pre_checkout_query_handler(lambda query: True)
async def checkout(pre_checkout_query: types.PreCheckoutQuery):
    await bot.answer_pre_checkout_query(pre_checkout_query.id, ok=True,
                                        error_message="Aliens tried to steal your card's␣
```

(continues on next page)

(continued from previous page)

```
↪CVV,"
                                                        " but we successfully protected␣
↪your credentials,"
                                                        " try to pay again in a few␣
↪minutes, we need a small rest.")


@dp.message_handler(content_types=ContentTypes.SUCCESSFUL_PAYMENT)
async def got_payment(message: types.Message):
    await bot.send_message(message.chat.id,
                           'Hoooooray! Thanks for payment! We will proceed your order␣
↪for `{} {}`'
                           ' as fast as possible! Stay in touch.'
                           '\n\nUse /buy again to get a Time Machine for your friend!'.
↪format(
                               message.successful_payment.total_amount / 100, message.
↪successful_payment.currency),
                           parse_mode='Markdown')


if __name__ == '__main__':
    executor.start_polling(dp, skip_updates=True)
```

4.7.14 Broadcast example

Listing 14: broadcast_example.py

```
import asyncio
import logging

from aiogram import Bot, Dispatcher, types
from aiogram.utils import exceptions, executor

API_TOKEN = 'BOT TOKEN HERE'

logging.basicConfig(level=logging.INFO)
log = logging.getLogger('broadcast')

bot = Bot(token=API_TOKEN, parse_mode=types.ParseMode.HTML)
dp = Dispatcher(bot)


def get_users():
    """
    Return users list

    In this example returns some random ID's
    """
    yield from (61043901, 78238238, 78378343, 98765431, 12345678)

```

(continues on next page)

(continued from previous page)

```python

async def send_message(user_id: int, text: str, disable_notification: bool = False) ->␣
↪bool:
    """
    Safe messages sender

    :param user_id:
    :param text:
    :param disable_notification:
    :return:
    """
    try:
        await bot.send_message(user_id, text, disable_notification=disable_notification)
    except exceptions.BotBlocked:
        log.error(f"Target [ID:{user_id}]: blocked by user")
    except exceptions.ChatNotFound:
        log.error(f"Target [ID:{user_id}]: invalid user ID")
    except exceptions.RetryAfter as e:
        log.error(f"Target [ID:{user_id}]: Flood limit is exceeded. Sleep {e.timeout}␣
↪seconds.")
        await asyncio.sleep(e.timeout)
        return await send_message(user_id, text)  # Recursive call
    except exceptions.UserDeactivated:
        log.error(f"Target [ID:{user_id}]: user is deactivated")
    except exceptions.TelegramAPIError:
        log.exception(f"Target [ID:{user_id}]: failed")
    else:
        log.info(f"Target [ID:{user_id}]: success")
        return True
    return False


async def broadcaster() -> int:
    """
    Simple broadcaster

    :return: Count of messages
    """
    count = 0
    try:
        for user_id in get_users():
            if await send_message(user_id, '<b>Hello!</b>'):
                count += 1
            await asyncio.sleep(.05)  # 20 messages per second (Limit: 30 messages per␣
↪second)
    finally:
        log.info(f"{count} messages successful sent.")

    return count


if __name__ == '__main__':
```

(continues on next page)

(continued from previous page)

```
    # Execute broadcaster
    executor.start(dp, broadcaster())
```

4.7.15 Media group

Listing 15: media_group.py

```
import asyncio

from aiogram import Bot, Dispatcher, executor, filters, types


API_TOKEN = 'BOT_TOKEN_HERE'

bot = Bot(token=API_TOKEN)
dp = Dispatcher(bot)


@dp.message_handler(filters.CommandStart())
async def send_welcome(message: types.Message):
    # So... At first I want to send something like this:
    await message.reply("Do you want to see many pussies? Are you ready?")

    # Wait a little...
    await asyncio.sleep(1)

    # Good bots should send chat actions...
    await types.ChatActions.upload_photo()

    # Create media group
    media = types.MediaGroup()

    # Attach local file
    media.attach_photo(types.InputFile('data/cat.jpg'), 'Cat!')
    # More local files and more cats!
    media.attach_photo(types.InputFile('data/cats.jpg'), 'More cats!')

    # You can also use URL's
    # For example: get random puss:
    media.attach_photo('http://lorempixel.com/400/200/cats/', 'Random cat.')

    # And you can also use file ID:
    # media.attach_photo('<file_id>', 'cat-cat-cat.')

    # Done! Send media group
    await message.reply_media_group(media=media)


if __name__ == '__main__':
    executor.start_polling(dp, skip_updates=True)
```

4.7.16 Local server

Listing 16: local_server.py

```
import logging

from aiogram import Bot, Dispatcher, executor, types
from aiogram.bot.api import TelegramAPIServer
from aiogram.types import ContentType

API_TOKEN = 'BOT TOKEN HERE'

# Configure logging
logging.basicConfig(level=logging.INFO)

# Create private Bot API server endpoints wrapper
local_server = TelegramAPIServer.from_base('http://localhost')

# Initialize bot with using local server
bot = Bot(token=API_TOKEN, server=local_server)
# ... and dispatcher
dp = Dispatcher(bot)


@dp.message_handler(content_types=ContentType.ANY)
async def echo(message: types.Message):
    await message.copy_to(message.chat.id)


if __name__ == '__main__':
    executor.start_polling(dp, skip_updates=True)
```

4.8 Contribution

TODO

4.9 Links

TODO

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

aiogram.bot.api, 72
aiogram.utils.auth_widget, 200
aiogram.utils.deep_linking, 216
aiogram.utils.deprecated, 213
aiogram.utils.emoji, 216
aiogram.utils.exceptions, 204
aiogram.utils.executor, 201
aiogram.utils.helper, 213
aiogram.utils.json, 216
aiogram.utils.markdown, 210
aiogram.utils.parts, 215
aiogram.utils.payload, 215

INDEX

A

AbstractFilter (*class in aiogram.dispatcher.filters*), 181
add() (*aiogram.types.inline_keyboard.InlineKeyboardMarkup method*), 84
add() (*aiogram.types.reply_keyboard.ReplyKeyboardMarkup method*), 87
add() (*aiogram.types.shipping_option.ShippingOption method*), 133
add_sticker_to_set() (*aiogram.bot.bot.Bot method*), 63
AdminFilter (*class in aiogram.dispatcher.filters*), 179
aiogram.bot.api
    module, 72
aiogram.utils.auth_widget
    module, 200
aiogram.utils.deep_linking
    module, 216
aiogram.utils.deprecated
    module, 213
aiogram.utils.emoji
    module, 216
aiogram.utils.exceptions
    module, 204
aiogram.utils.executor
    module, 201
aiogram.utils.helper
    module, 213
aiogram.utils.json
    module, 216
aiogram.utils.markdown
    module, 210
aiogram.utils.parts
    module, 215
aiogram.utils.payload
    module, 215
AIOGramWarning, 206
AllowedUpdates (*class in aiogram.types.update*), 130
Animation (*class in aiogram.types.animation*), 107
answer() (*aiogram.types.callback_query.CallbackQuery method*), 80
answer() (*aiogram.types.inline_query.InlineQuery method*), 106
answer() (*aiogram.types.message.Message method*), 136
answer_animation() (*aiogram.types.message.Message method*), 138
answer_audio() (*aiogram.types.message.Message method*), 137
answer_callback_query() (*aiogram.bot.bot.Bot method*), 55
answer_chat_action() (*aiogram.types.message.Message method*), 149
answer_contact() (*aiogram.types.message.Message method*), 146
answer_dice() (*aiogram.types.message.Message method*), 148
answer_document() (*aiogram.types.message.Message method*), 140
answer_inline_query() (*aiogram.bot.bot.Bot method*), 65
answer_location() (*aiogram.types.message.Message method*), 144
answer_media_group() (*aiogram.types.message.Message method*), 143
answer_photo() (*aiogram.types.message.Message method*), 137
answer_poll() (*aiogram.types.message.Message method*), 147
answer_pre_checkout_query() (*aiogram.bot.bot.Bot method*), 69
answer_shipping_query() (*aiogram.bot.bot.Bot method*), 69
answer_sticker() (*aiogram.types.message.Message method*), 146
answer_venue() (*aiogram.types.message.Message method*), 145
answer_video() (*aiogram.types.message.Message method*), 141
answer_video_note() (*aiogram.types.message.Message method*), 142

answer_voice() (*aiogram.types.message.Message method*), 142
answer_web_app_query() (*aiogram.bot.bot.Bot method*), 65
api_url() (*aiogram.bot.api.TelegramAPIServer method*), 72
approve_chat_join_request() (*aiogram.bot.bot.Bot method*), 47
as_json() (*aiogram.types.base.TelegramObject method*), 74
async_task() (*aiogram.Dispatcher method*), 199
attach() (*aiogram.types.input_media.MediaGroup method*), 110
attach_audio() (*aiogram.types.input_media.MediaGroup method*), 110
attach_document() (*aiogram.types.input_media.MediaGroup method*), 110
attach_many() (*aiogram.types.input_media.MediaGroup method*), 110
attach_photo() (*aiogram.types.input_media.MediaGroup method*), 111
attach_video() (*aiogram.types.input_media.MediaGroup method*), 111
Audio (*class in aiogram.types.audio*), 100
AuthWidgetData (*class in aiogram.types.auth_widget_data*), 169

B

BadRequest, 206
BadWebhook, 209
BadWebhookAddrInfo, 209
BadWebhookNoAddressAssociatedWithHostname, 209
BadWebhookPort, 209
ban_chat_member() (*aiogram.bot.bot.Bot method*), 41
ban_chat_sender_chat() (*aiogram.bot.bot.Bot method*), 44
ban_sender_chat() (*aiogram.types.chat.Chat method*), 96
BaseBot (*class in aiogram.bot.base*), 17
BaseField (*class in aiogram.types.fields*), 75
bind() (*aiogram.dispatcher.filters.FiltersFactory method*), 170
bind_filter() (*aiogram.Dispatcher method*), 200
bold() (*in module aiogram.utils.markdown*), 210
Bot (*class in aiogram.bot.bot*), 19
BotBlocked, 209
BotDomainInvalid, 209
BotKicked, 209
BoundFilter (*class in aiogram.dispatcher.filters*), 182
ButtonDataInvalid, 208
ButtonURLInvalid, 208

C

calc_timeout() (*aiogram.types.chat.ChatActions class method*), 98
callback_query_handler() (*aiogram.Dispatcher method*), 193
CallbackGame (*class in aiogram.types.callback_game*), 87
CallbackQuery (*class in aiogram.types.callback_query*), 80
CantDemoteChatCreator, 208
CantGetUpdates, 209
CantInitiateConversation, 209
CantParseEntities, 209
CantParseUrl, 209
CantRestrictChatOwner, 209
CantRestrictSelf, 208
CantTalkWithBots, 209
channel_post_handler() (*aiogram.Dispatcher method*), 190
Chat (*class in aiogram.types.chat*), 89
chat_join_request_handler() (*aiogram.Dispatcher method*), 197
chat_member_handler() (*aiogram.Dispatcher method*), 197
ChatActions (*class in aiogram.types.chat*), 98
ChatAdminRequired, 208
ChatDescriptionIsNotModified, 208
ChatIdIsEmpty, 208
ChatMember (*class in aiogram.types.chat_member*), 132
ChatMemberStatus (*class in aiogram.types.chat_member*), 133
ChatNotFound, 208
ChatPhoto (*class in aiogram.types.chat_photo*), 133
ChatType (*class in aiogram.types.chat*), 97
ChatTypeFilter (*class in aiogram.dispatcher.filters*), 180
check() (*aiogram.dispatcher.filters.AbstractFilter method*), 181
check() (*aiogram.dispatcher.filters.AdminFilter method*), 179
check() (*aiogram.dispatcher.filters.builtin.IDFilter method*), 179
check() (*aiogram.dispatcher.filters.ChatTypeFilter method*), 180
check() (*aiogram.dispatcher.filters.Command method*), 172
check() (*aiogram.dispatcher.filters.CommandStart method*), 173
check() (*aiogram.dispatcher.filters.ContentTypeFilter method*), 177
check() (*aiogram.dispatcher.filters.ExceptionsFilter method*), 178
check() (*aiogram.dispatcher.filters.ForwardedMessageFilter method*), 180

check() (*aiogram.dispatcher.filters.HashTag method*), 176
check() (*aiogram.dispatcher.filters.IsReplyFilter method*), 180
check() (*aiogram.dispatcher.filters.IsSenderContact method*), 177
check() (*aiogram.dispatcher.filters.MediaGroupFilter method*), 181
check() (*aiogram.dispatcher.filters.Regexp method*), 176
check() (*aiogram.dispatcher.filters.RegexpCommandsFilter method*), 177
check() (*aiogram.dispatcher.filters.StateFilter method*), 178
check() (*aiogram.dispatcher.filters.Text method*), 175
check_integrity() (*in module aiogram.utils.auth_widget*), 201
check_key() (*aiogram.Dispatcher method*), 198
check_result() (*in module aiogram.bot.api*), 72
check_signature() (*in module aiogram.utils.auth_widget*), 201
check_token() (*in module aiogram.bot.api*), 72
check_token() (*in module aiogram.utils.auth_widget*), 200
choose_sticker() (*aiogram.types.chat.ChatActions class method*), 100
chosen_inline_handler() (*aiogram.Dispatcher method*), 192
ChosenInlineResult (*class in aiogram.types.chosen_inline_result*), 131
clean() (*aiogram.types.base.TelegramObject method*), 74
close() (*aiogram.bot.base.BaseBot method*), 18
close_bot() (*aiogram.bot.bot.Bot method*), 23
close_forum_topic() (*aiogram.bot.bot.Bot method*), 54
close_general_forum_topic() (*aiogram.bot.bot.Bot method*), 50
code() (*in module aiogram.utils.markdown*), 211
command (*aiogram.dispatcher.filters.Command.CommandObj attribute*), 172
Command (*class in aiogram.dispatcher.filters*), 171
Command.CommandObj (*class in aiogram.dispatcher.filters*), 172
CommandHelp (*class in aiogram.dispatcher.filters*), 173
CommandPrivacy (*class in aiogram.dispatcher.filters*), 174
CommandSettings (*class in aiogram.dispatcher.filters*), 174
CommandStart (*class in aiogram.dispatcher.filters*), 172
compose_data() (*in module aiogram.bot.api*), 73
ConflictError, 209
Contact (*class in aiogram.types.contact*), 134
ContentType (*class in aiogram.types.message*), 166
ContentTypeFilter (*class in aiogram.dispatcher.filters*), 177
ContentTypes (*class in aiogram.types.message*), 167
copy_message() (*aiogram.bot.bot.Bot method*), 24
create() (*aiogram.types.force_reply.ForceReply class method*), 101
create_chat_invite_link() (*aiogram.bot.bot.Bot method*), 45
create_forum_topic() (*aiogram.bot.bot.Bot method*), 53
create_invite_link() (*aiogram.types.chat.Chat method*), 96
create_invoice_link() (*aiogram.bot.bot.Bot method*), 68
create_new_sticker_set() (*aiogram.bot.bot.Bot method*), 62
CurrencyTotalAmountInvalid, 209
current_state() (*aiogram.Dispatcher method*), 198

D

DateTimeField (*class in aiogram.types.fields*), 77
decline_chat_join_request() (*aiogram.bot.bot.Bot method*), 47
decode_payload() (*in module aiogram.utils.deep_linking*), 217
default (*aiogram.dispatcher.filters.BoundFilter attribute*), 182
default (*aiogram.dispatcher.filters.ContentTypeFilter attribute*), 177
delete() (*aiogram.types.message.Message method*), 164
delete_chat_photo() (*aiogram.bot.bot.Bot method*), 48
delete_chat_sticker_set() (*aiogram.bot.bot.Bot method*), 52
delete_forum_topic() (*aiogram.bot.bot.Bot method*), 54
delete_from_set() (*aiogram.types.sticker.Sticker method*), 105
delete_message() (*aiogram.bot.bot.Bot method*), 60
delete_message() (*aiogram.types.chat.Chat method*), 96
delete_my_commands() (*aiogram.bot.bot.Bot method*), 56
delete_photo() (*aiogram.types.chat.Chat method*), 90
delete_reply_markup() (*aiogram.types.message.Message method*), 164
delete_sticker_from_set() (*aiogram.bot.bot.Bot method*), 64
delete_sticker_set() (*aiogram.types.chat.Chat method*), 95
delete_webhook() (*aiogram.bot.bot.Bot method*), 22
deprecated() (*in module aiogram.utils.deprecated*), 213

DeprecatedReadOnlyClassVar (*class in aiogram.utils.deprecated*), 214
deserialize() (*aiogram.types.fields.BaseField method*), 75
deserialize() (*aiogram.types.fields.DateTimeField method*), 78
deserialize() (*aiogram.types.fields.Field method*), 76
deserialize() (*aiogram.types.fields.ListField method*), 77
deserialize() (*aiogram.types.fields.ListOfLists method*), 77
deserialize() (*aiogram.types.fields.TextField method*), 78
Dispatcher (*class in aiogram*), 186
do() (*aiogram.types.chat.Chat method*), 95
Document (*class in aiogram.types.document*), 100
download() (*aiogram.types.mixins.Downloadable method*), 78
download_big() (*aiogram.types.chat_photo.ChatPhoto method*), 134
download_file() (*aiogram.bot.base.BaseBot method*), 18
download_file_by_id() (*aiogram.bot.bot.Bot method*), 20
download_small() (*aiogram.types.chat_photo.ChatPhoto method*), 134
Downloadable (*class in aiogram.types.mixins*), 78

E

edit_caption() (*aiogram.types.message.Message method*), 162
edit_chat_invite_link() (*aiogram.bot.bot.Bot method*), 46
edit_forum_topic() (*aiogram.bot.bot.Bot method*), 53
edit_general_forum_topic() (*aiogram.bot.bot.Bot method*), 50
edit_invite_link() (*aiogram.types.chat.Chat method*), 96
edit_live_location() (*aiogram.types.message.Message method*), 164
edit_media() (*aiogram.types.message.Message method*), 163
edit_message_caption() (*aiogram.bot.bot.Bot method*), 58
edit_message_live_location() (*aiogram.bot.bot.Bot method*), 35
edit_message_media() (*aiogram.bot.bot.Bot method*), 59
edit_message_reply_markup() (*aiogram.bot.bot.Bot method*), 60
edit_message_text() (*aiogram.bot.bot.Bot method*), 58
edit_reply_markup() (*aiogram.types.message.Message method*), 163
edit_text() (*aiogram.types.message.Message method*), 162
edited_channel_post_handler() (*aiogram.Dispatcher method*), 191
edited_message_handler() (*aiogram.Dispatcher method*), 189
encode_payload() (*in module aiogram.utils.deep_linking*), 217
EncryptedCredentials (*class in aiogram.types.encrypted_credentials*), 80
EncryptedPassportElement (*class in aiogram.types.encrypted_passport_element*), 86
errors_handler() (*aiogram.Dispatcher method*), 198
escape_md() (*in module aiogram.utils.markdown*), 210
ExceptionsFilter (*class in aiogram.dispatcher.filters*), 178
Executor (*class in aiogram.utils.executor*), 203
export() (*aiogram.types.fields.BaseField method*), 75
export_chat_invite_link() (*aiogram.bot.bot.Bot method*), 45
export_invite_link() (*aiogram.types.chat.Chat method*), 96

F

Field (*class in aiogram.types.fields*), 76
File (*class in aiogram.types.file*), 86
file_url() (*aiogram.bot.api.TelegramAPIServer method*), 72
FileIsTooBig, 208
Filter (*class in aiogram.dispatcher.filters*), 182
FiltersFactory (*class in aiogram.dispatcher.filters*), 170
find_location() (*aiogram.types.chat.ChatActions class method*), 99
ForceReply (*class in aiogram.types.force_reply*), 101
forward() (*aiogram.types.message.Message method*), 161
forward_message() (*aiogram.bot.bot.Bot method*), 24
ForwardedMessageFilter (*class in aiogram.dispatcher.filters*), 180
from_id (*aiogram.types.message.Message property*), 135
from_url() (*aiogram.types.input_file.InputFile class method*), 126
FSMStorageWarning, 206
full_name (*aiogram.types.user.User property*), 85

G

Game (*class in aiogram.types.game*), 86

GameHighScore (*class in aiogram.types.game_high_score*), 105
generate_hash() (*in module aiogram.utils.auth_widget*), 200
generate_payload() (*in module aiogram.utils.payload*), 215
get_administrators() (*aiogram.types.chat.Chat method*), 94
get_args() (*aiogram.types.message.Message method*), 135
get_chat() (*aiogram.bot.bot.Bot method*), 51
get_chat_administrators() (*aiogram.bot.bot.Bot method*), 51
get_chat_member() (*aiogram.bot.bot.Bot method*), 52
get_chat_member_count() (*aiogram.bot.bot.Bot method*), 51
get_chat_members_count() (*aiogram.bot.bot.Bot method*), 52
get_chat_menu_button() (*aiogram.bot.bot.Bot method*), 57
get_command() (*aiogram.types.message.Message method*), 135
get_custom_emoji_stickers() (*aiogram.bot.bot.Bot method*), 62
get_file() (*aiogram.bot.bot.Bot method*), 41
get_file() (*aiogram.types.input_file.InputFile method*), 126
get_file() (*aiogram.types.mixins.Downloadable method*), 79
get_filename() (*aiogram.types.input_file.InputFile method*), 126
get_forum_topic_icon_stickers() (*aiogram.bot.bot.Bot method*), 53
get_full_command() (*aiogram.types.message.Message method*), 135
get_game_high_scores() (*aiogram.bot.bot.Bot method*), 71
get_me() (*aiogram.bot.bot.Bot method*), 22
get_member() (*aiogram.types.chat.Chat method*), 95
get_member_count() (*aiogram.types.chat.Chat method*), 95
get_members_count() (*aiogram.types.chat.Chat method*), 95
get_my_commands() (*aiogram.bot.bot.Bot method*), 56
get_my_default_administrator_rights() (*aiogram.bot.bot.Bot method*), 57
get_start_link() (*in module aiogram.utils.deep_linking*), 216
get_startgroup_link() (*in module aiogram.utils.deep_linking*), 216
get_sticker_set() (*aiogram.bot.bot.Bot method*), 61
get_text() (*aiogram.types.message_entity.MessageEntity method*), 81
get_updates() (*aiogram.bot.bot.Bot method*), 20
get_url() (*aiogram.types.chat.Chat method*), 89
get_url() (*aiogram.types.mixins.Downloadable method*), 79
get_user_profile_photos() (*aiogram.bot.bot.Bot method*), 40
get_value() (*aiogram.types.fields.BaseField method*), 75
get_webhook_info() (*aiogram.bot.bot.Bot method*), 22
GroupDeactivated, 208
guess_filename() (*in module aiogram.bot.api*), 73

H

HashTag (*class in aiogram.dispatcher.filters*), 176
hbold() (*in module aiogram.utils.markdown*), 210
hcode() (*in module aiogram.utils.markdown*), 211
hide_general_forum_topic() (*aiogram.bot.bot.Bot method*), 50
hide_link() (*in module aiogram.utils.markdown*), 212
hitalic() (*in module aiogram.utils.markdown*), 211
hlink() (*in module aiogram.utils.markdown*), 212
hpre() (*in module aiogram.utils.markdown*), 211
hspoiler() (*in module aiogram.utils.markdown*), 211
hstrikethrough() (*in module aiogram.utils.markdown*), 212
html_text (*aiogram.types.message.Message property*), 135
hunderline() (*in module aiogram.utils.markdown*), 212

I

IDFilter (*class in aiogram.dispatcher.filters.builtin*), 179
inline_handler() (*aiogram.Dispatcher method*), 192
InlineKeyboardButton (*class in aiogram.types.inline_keyboard*), 84
InlineKeyboardExpected, 207
InlineKeyboardMarkup (*class in aiogram.types.inline_keyboard*), 83
InlineQuery (*class in aiogram.types.inline_query*), 106
InlineQueryResult (*class in aiogram.types.inline_query_result*), 112
InlineQueryResultArticle (*class in aiogram.types.inline_query_result*), 112
InlineQueryResultAudio (*class in aiogram.types.inline_query_result*), 115
InlineQueryResultCachedAudio (*class in aiogram.types.inline_query_result*), 125
InlineQueryResultCachedDocument (*class in aiogram.types.inline_query_result*), 123
InlineQueryResultCachedGif (*class in aiogram.types.inline_query_result*), 121
InlineQueryResultCachedMpeg4Gif (*class in aiogram.types.inline_query_result*), 122
InlineQueryResultCachedPhoto (*class in aiogram.types.inline_query_result*), 120

InlineQueryResultCachedSticker (*class in aiogram.types.inline_query_result*), 123
InlineQueryResultCachedVideo (*class in aiogram.types.inline_query_result*), 124
InlineQueryResultCachedVoice (*class in aiogram.types.inline_query_result*), 124
InlineQueryResultContact (*class in aiogram.types.inline_query_result*), 119
InlineQueryResultDocument (*class in aiogram.types.inline_query_result*), 117
InlineQueryResultGame (*class in aiogram.types.inline_query_result*), 120
InlineQueryResultGif (*class in aiogram.types.inline_query_result*), 113
InlineQueryResultLocation (*class in aiogram.types.inline_query_result*), 118
InlineQueryResultMpeg4Gif (*class in aiogram.types.inline_query_result*), 114
InlineQueryResultPhoto (*class in aiogram.types.inline_query_result*), 113
InlineQueryResultVenue (*class in aiogram.types.inline_query_result*), 119
InlineQueryResultVideo (*class in aiogram.types.inline_query_result*), 115
InlineQueryResultVoice (*class in aiogram.types.inline_query_result*), 116
InputContactMessageContent (*class in aiogram.types.input_message_content*), 128
InputFile (*class in aiogram.types.input_file*), 126
InputLocationMessageContent (*class in aiogram.types.input_message_content*), 128
InputMedia (*class in aiogram.types.input_media*), 107
InputMediaAnimation (*class in aiogram.types.input_media*), 108
InputMediaAudio (*class in aiogram.types.input_media*), 109
InputMediaDocument (*class in aiogram.types.input_media*), 108
InputMediaPhoto (*class in aiogram.types.input_media*), 109
InputMediaVideo (*class in aiogram.types.input_media*), 109
InputMessageContent (*class in aiogram.types.input_message_content*), 128
InputTextMessageContent (*class in aiogram.types.input_message_content*), 129
InputVenueMessageContent (*class in aiogram.types.input_message_content*), 129
insert() (*aiogram.types.inline_keyboard.InlineKeyboardMarkup method*), 84
insert() (*aiogram.types.reply_keyboard.ReplyKeyboardMarkup method*), 88
InvalidHTTPUrlContent, 208
InvalidPeerID, 208
InvalidQueryID, 208
InvalidStickersSet, 208
InvalidUserId, 208
Invoice (*class in aiogram.types.invoice*), 169
is_channel() (*aiogram.types.chat.ChatType class method*), 97
is_command() (*aiogram.types.message.Message method*), 135
is_forward() (*aiogram.types.message.Message method*), 135
is_group() (*aiogram.types.chat.ChatType class method*), 97
is_group_or_super_group() (*aiogram.types.chat.ChatType class method*), 97
is_polling() (*aiogram.Dispatcher method*), 187
is_private() (*aiogram.types.chat.ChatType class method*), 97
is_super_group() (*aiogram.types.chat.ChatType class method*), 97
IsReplyFilter (*class in aiogram.dispatcher.filters*), 180
IsSenderContact (*class in aiogram.dispatcher.filters*), 177
italic() (*in module aiogram.utils.markdown*), 211
Item (*class in aiogram.utils.helper*), 213
ItemsList (*class in aiogram.utils.helper*), 213
iter_keys() (*aiogram.types.base.TelegramObject method*), 74
iter_values() (*aiogram.types.base.TelegramObject method*), 74

K

key (*aiogram.dispatcher.filters.BoundFilter attribute*), 182
key (*aiogram.dispatcher.filters.ChatTypeFilter attribute*), 180
key (*aiogram.dispatcher.filters.ContentTypeFilter attribute*), 177
key (*aiogram.dispatcher.filters.ExceptionsFilter attribute*), 178
key (*aiogram.dispatcher.filters.ForwardedMessageFilter attribute*), 180
key (*aiogram.dispatcher.filters.IsReplyFilter attribute*), 180
key (*aiogram.dispatcher.filters.IsSenderContact attribute*), 177
key (*aiogram.dispatcher.filters.MediaGroupFilter attribute*), 181
key (*aiogram.dispatcher.filters.RegexpCommandsFilter attribute*), 177
key (*aiogram.dispatcher.filters.StateFilter attribute*), 178
KeyboardButton (*class in aiogram.types.reply_keyboard*), 88
kick() (*aiogram.types.chat.Chat method*), 91

kick_chat_member() (*aiogram.bot.bot.Bot method*), 42

L

LabeledPrice (*class in aiogram.types.labeled_price*), 87
leave() (*aiogram.types.chat.Chat method*), 94
leave_chat() (*aiogram.bot.bot.Bot method*), 51
link() (*aiogram.types.message.Message method*), 136
link() (*in module aiogram.utils.markdown*), 212
ListField (*class in aiogram.types.fields*), 76
ListItem (*class in aiogram.utils.helper*), 213
ListOfLists (*class in aiogram.types.fields*), 77
locale (*aiogram.types.user.User property*), 85
Location (*class in aiogram.types.location*), 107
log_out() (*aiogram.bot.bot.Bot method*), 22

M

MaskPosition (*class in aiogram.types.mask_position*), 168
md_text (*aiogram.types.message.Message property*), 135
me (*aiogram.bot.bot.Bot property*), 20
MediaGroup (*class in aiogram.types.input_media*), 110
MediaGroupFilter (*class in aiogram.dispatcher.filters*), 181
MemoryStorage (*class in aiogram.contrib.fsm_storage.memory*), 183
mention (*aiogram.dispatcher.filters.Command.CommandObj attribute*), 172
mention (*aiogram.types.chat.Chat property*), 89
mention (*aiogram.types.user.User property*), 85
mentioned (*aiogram.dispatcher.filters.Command.CommandObj property*), 172
Message (*class in aiogram.types.message*), 135
message_handler() (*aiogram.Dispatcher method*), 188
MessageCantBeDeleted, 207
MessageCantBeEdited, 207
MessageCantBeForwarded, 207
MessageEntity (*class in aiogram.types.message_entity*), 81
MessageEntityType (*class in aiogram.types.message_entity*), 82
MessageError, 207
MessageIdentifierNotSpecified, 207
MessageIdInvalid, 207
MessageIsNotAPoll, 208
MessageIsTooLong, 207
MessageNotModified, 207
MessageTextIsEmpty, 207
MessageToDeleteNotFound, 207
MessageToEditNotFound, 207
MessageToForwardNotFound, 207
MessageToPinNotFound, 207
MessageToReplyNotFound, 207
MessageWithPollNotFound, 208
MetaTelegramObject (*class in aiogram.types.base*), 73
MethodIsNotAvailable, 209
MethodNotAvailableInPrivateChats, 208
MethodNotKnown, 209
Methods (*class in aiogram.bot.api*), 73
MigrateToChat, 210
module
    aiogram.bot.api, 72
    aiogram.utils.auth_widget, 200
    aiogram.utils.deep_linking, 216
    aiogram.utils.deprecated, 213
    aiogram.utils.emoji, 216
    aiogram.utils.exceptions, 204
    aiogram.utils.executor, 201
    aiogram.utils.helper, 213
    aiogram.utils.json, 216
    aiogram.utils.markdown, 210
    aiogram.utils.parts, 215
    aiogram.utils.payload, 215
MongoStorage (*class in aiogram.contrib.fsm_storage.mongo*), 184
my_chat_member_handler() (*aiogram.Dispatcher method*), 196

N

NeedAdministratorRightsInTheChannel, 208
NetworkError, 209
NoStickerInRequest, 208
NotEnoughRightsToPinMessage, 208
NotEnoughRightsToRestrict, 208
NotFound, 209

O

ObjectExpectedAsReplyMarkup, 207
on_shutdown() (*aiogram.utils.executor.Executor method*), 203
on_startup() (*aiogram.utils.executor.Executor method*), 203
OrderInfo (*class in aiogram.types.order_info*), 104

P

paginate() (*in module aiogram.utils.parts*), 215
parse() (*aiogram.types.auth_widget_data.AuthWidgetData class method*), 169
parse() (*aiogram.types.message_entity.MessageEntity method*), 81
parse_entities() (*aiogram.types.message.Message method*), 135
ParseMode (*class in aiogram.types.message*), 168
PassportData (*class in aiogram.types.passport_data*), 83

PassportElementError (*class in aiogram.types.passport_element_error*), 101
PassportElementErrorDataField (*class in aiogram.types.passport_element_error*), 102
PassportElementErrorFile (*class in aiogram.types.passport_element_error*), 102
PassportElementErrorFiles (*class in aiogram.types.passport_element_error*), 102
PassportElementErrorFrontSide (*class in aiogram.types.passport_element_error*), 103
PassportElementErrorReverseSide (*class in aiogram.types.passport_element_error*), 103
PassportElementErrorSelfie (*class in aiogram.types.passport_element_error*), 103
PassportFile (*class in aiogram.types.passport_file*), 132
PaymentProviderInvalid, 209
PhotoAsInputFileRequired, 208
PhotoDimensions, 208
PhotoSize (*class in aiogram.types.photo_size*), 130
pin() (*aiogram.types.message.Message method*), 165
pin_chat_message() (*aiogram.bot.bot.Bot method*), 49
pin_message() (*aiogram.types.chat.Chat method*), 93
poll_answer_handler() (*aiogram.Dispatcher method*), 196
poll_handler() (*aiogram.Dispatcher method*), 195
PollCanBeRequestedInPrivateChatsOnly, 208
PollCantBeStopped, 207
PollCantHaveMoreOptions, 207
PollError, 207
PollHasAlreadyBeenClosed, 207
PollMustHaveMoreOptions, 207
PollOptionsLengthTooLong, 207
PollOptionsMustBeNonEmpty, 207
PollQuestionLengthTooLong, 208
PollQuestionMustBeNonEmpty, 207
PollsCantBeSentToPrivateChats, 207
PollSizeError, 207
pre() (*in module aiogram.utils.markdown*), 211
pre_checkout_query_handler() (*aiogram.Dispatcher method*), 194
PreCheckoutQuery (*class in aiogram.types.pre_checkout_query*), 127
prefix (*aiogram.dispatcher.filters.Command.CommandObj attribute*), 172
prepare_arg() (*in module aiogram.utils.payload*), 215
process_update() (*aiogram.Dispatcher method*), 187
process_updates() (*aiogram.Dispatcher method*), 186
promote() (*aiogram.types.chat.Chat method*), 92
promote_chat_member() (*aiogram.bot.bot.Bot method*), 43
props (*aiogram.types.base.TelegramObject property*), 74
props_aliases (*aiogram.types.base.TelegramObject property*), 74

Q

quote_html() (*in module aiogram.utils.markdown*), 210

R

record_audio() (*aiogram.types.chat.ChatActions class method*), 99
record_video() (*aiogram.types.chat.ChatActions class method*), 99
record_video_note() (*aiogram.types.chat.ChatActions class method*), 100
record_voice() (*aiogram.types.chat.ChatActions class method*), 99
RedisStorage2 (*class in aiogram.contrib.fsm_storage.redis*), 183
Regexp (*class in aiogram.dispatcher.filters*), 176
RegexpCommandsFilter (*class in aiogram.dispatcher.filters*), 177
register_callback_query_handler() (*aiogram.Dispatcher method*), 193
register_channel_post_handler() (*aiogram.Dispatcher method*), 190
register_chat_join_request_handler() (*aiogram.Dispatcher method*), 197
register_chat_member_handler() (*aiogram.Dispatcher method*), 196
register_chosen_inline_handler() (*aiogram.Dispatcher method*), 192
register_edited_channel_post_handler() (*aiogram.Dispatcher method*), 191
register_edited_message_handler() (*aiogram.Dispatcher method*), 189
register_errors_handler() (*aiogram.Dispatcher method*), 198
register_inline_handler() (*aiogram.Dispatcher method*), 191
register_message_handler() (*aiogram.Dispatcher method*), 187
register_my_chat_member_handler() (*aiogram.Dispatcher method*), 196
register_poll_answer_handler() (*aiogram.Dispatcher method*), 195
register_poll_handler() (*aiogram.Dispatcher method*), 195
register_pre_checkout_query_handler() (*aiogram.Dispatcher method*), 194

register_shipping_query_handler() (*aiogram.Dispatcher method*), 193
release_key() (*aiogram.Dispatcher method*), 199
removed_argument() (*in module aiogram.utils.deprecated*), 214
renamed_argument() (*in module aiogram.utils.deprecated*), 213
reopen_forum_topic() (*aiogram.bot.bot.Bot method*), 54
reopen_general_forum_topic() (*aiogram.bot.bot.Bot method*), 50
reply() (*aiogram.types.message.Message method*), 149
reply_animation() (*aiogram.types.message.Message method*), 151
reply_audio() (*aiogram.types.message.Message method*), 150
reply_contact() (*aiogram.types.message.Message method*), 158
reply_dice() (*aiogram.types.message.Message method*), 161
reply_document() (*aiogram.types.message.Message method*), 153
reply_location() (*aiogram.types.message.Message method*), 157
reply_media_group() (*aiogram.types.message.Message method*), 156
reply_photo() (*aiogram.types.message.Message method*), 150
reply_poll() (*aiogram.types.message.Message method*), 159
reply_sticker() (*aiogram.types.message.Message method*), 160
reply_venue() (*aiogram.types.message.Message method*), 157
reply_video() (*aiogram.types.message.Message method*), 153
reply_video_note() (*aiogram.types.message.Message method*), 155
reply_voice() (*aiogram.types.message.Message method*), 155
ReplyKeyboardMarkup (*class in aiogram.types.reply_keyboard*), 87
ReplyKeyboardRemove (*class in aiogram.types.reply_keyboard*), 89
request() (*aiogram.bot.base.BaseBot method*), 18
request_timeout() (*aiogram.bot.base.BaseBot method*), 18
required (*aiogram.dispatcher.filters.BoundFilter attribute*), 182
required (*aiogram.dispatcher.filters.ContentTypeFilter attribute*), 177
required (*aiogram.dispatcher.filters.StateFilter attribute*), 178
reset_webhook() (*aiogram.Dispatcher method*), 187
resolve() (*aiogram.dispatcher.filters.FiltersFactory method*), 170
ResponseParameters (*class in aiogram.types.response_parameters*), 104
RestartingTelegram, 210
restrict() (*aiogram.types.chat.Chat method*), 91
restrict_chat_member() (*aiogram.bot.bot.Bot method*), 42
ResultIdDuplicate, 209
RethinkDBStorage (*class in aiogram.contrib.fsm_storage.rethinkdb*), 184
RetryAfter, 210
revoke_chat_invite_link() (*aiogram.bot.bot.Bot method*), 47
revoke_invite_link() (*aiogram.types.chat.Chat method*), 96
row() (*aiogram.types.inline_keyboard.InlineKeyboardMarkup method*), 84
row() (*aiogram.types.reply_keyboard.ReplyKeyboardMarkup method*), 88

S

safe_split_text() (*in module aiogram.utils.parts*), 215
save() (*aiogram.types.input_file.InputFile method*), 126
send_animation() (*aiogram.bot.bot.Bot method*), 30
send_audio() (*aiogram.bot.bot.Bot method*), 26
send_chat_action() (*aiogram.bot.bot.Bot method*), 40
send_contact() (*aiogram.bot.bot.Bot method*), 37
send_copy() (*aiogram.types.message.Message method*), 165
send_dice() (*aiogram.bot.bot.Bot method*), 39
send_document() (*aiogram.bot.bot.Bot method*), 27
send_file() (*aiogram.bot.base.BaseBot method*), 19
send_game() (*aiogram.bot.bot.Bot method*), 70
send_invoice() (*aiogram.bot.bot.Bot method*), 66
send_location() (*aiogram.bot.bot.Bot method*), 34
send_media_group() (*aiogram.bot.bot.Bot method*), 33
send_message() (*aiogram.bot.bot.Bot method*), 23
send_photo() (*aiogram.bot.bot.Bot method*), 25
send_poll() (*aiogram.bot.bot.Bot method*), 38
send_sticker() (*aiogram.bot.bot.Bot method*), 61
send_venue() (*aiogram.bot.bot.Bot method*), 36
send_video() (*aiogram.bot.bot.Bot method*), 28
send_video_note() (*aiogram.bot.bot.Bot method*), 32
send_voice() (*aiogram.bot.bot.Bot method*), 31
serialize() (*aiogram.types.fields.BaseField method*), 75
serialize() (*aiogram.types.fields.DateTimeField method*), 77
serialize() (*aiogram.types.fields.Field method*), 76

serialize() (*aiogram.types.fields.ListField method*), 76
serialize() (*aiogram.types.fields.ListOfLists method*), 77
serialize() (*aiogram.types.fields.TextField method*), 78
set_administrator_custom_title() (*aiogram.types.chat.Chat method*), 93
set_chat_administrator_custom_title() (*aiogram.bot.bot.Bot method*), 44
set_chat_description() (*aiogram.bot.bot.Bot method*), 48
set_chat_menu_button() (*aiogram.bot.bot.Bot method*), 57
set_chat_permissions() (*aiogram.bot.bot.Bot method*), 45
set_chat_photo() (*aiogram.bot.bot.Bot method*), 47
set_chat_sticker_set() (*aiogram.bot.bot.Bot method*), 52
set_chat_title() (*aiogram.bot.bot.Bot method*), 48
set_description() (*aiogram.types.chat.Chat method*), 90
set_game_score() (*aiogram.bot.bot.Bot method*), 71
set_my_commands() (*aiogram.bot.bot.Bot method*), 55
set_my_default_administrator_rights() (*aiogram.bot.bot.Bot method*), 57
set_passport_data_errors() (*aiogram.bot.bot.Bot method*), 70
set_permissions() (*aiogram.types.chat.Chat method*), 93
set_photo() (*aiogram.types.chat.Chat method*), 89
set_position_in_set() (*aiogram.types.sticker.Sticker method*), 105
set_sticker_position_in_set() (*aiogram.bot.bot.Bot method*), 64
set_sticker_set() (*aiogram.types.chat.Chat method*), 95
set_sticker_set_thumb() (*aiogram.bot.bot.Bot method*), 64
set_title() (*aiogram.types.chat.Chat method*), 90
set_value() (*aiogram.types.fields.BaseField method*), 75
set_web_app() (*aiogram.utils.executor.Executor method*), 203
set_webhook() (*aiogram.bot.bot.Bot method*), 21
set_webhook() (*aiogram.utils.executor.Executor method*), 203
set_webhook() (*in module aiogram.utils.executor*), 201
setup_middleware() (*aiogram.Dispatcher method*), 200
shifted_id (*aiogram.types.chat.Chat property*), 89
shipping_query_handler() (*aiogram.Dispatcher method*), 194
ShippingAddress (*class in aiogram.types.shipping_address*), 104
ShippingOption (*class in aiogram.types.shipping_option*), 133
ShippingQuery (*class in aiogram.types.shipping_query*), 83
skip_updates() (*aiogram.Dispatcher method*), 186
split_text() (*in module aiogram.utils.parts*), 215
spoiler() (*in module aiogram.utils.markdown*), 211
start() (*aiogram.utils.executor.Executor method*), 204
start() (*in module aiogram.utils.executor*), 202
start_polling() (*aiogram.Dispatcher method*), 187
start_polling() (*aiogram.utils.executor.Executor method*), 204
start_polling() (*in module aiogram.utils.executor*), 201
start_webhook() (*aiogram.utils.executor.Executor method*), 203
start_webhook() (*in module aiogram.utils.executor*), 202
StartParamInvalid, 208
StateFilter (*class in aiogram.dispatcher.filters*), 178
Sticker (*class in aiogram.types.sticker*), 105
StickerSet (*class in aiogram.types.sticker_set*), 79
stop_live_location() (*aiogram.types.message.Message method*), 164
stop_message_live_location() (*aiogram.bot.bot.Bot method*), 35
stop_poll() (*aiogram.bot.bot.Bot method*), 60
stop_polling() (*aiogram.Dispatcher method*), 187
strikethrough() (*in module aiogram.utils.markdown*), 212
SuccessfulPayment (*class in aiogram.types.successful_payment*), 81

T

TelegramAPIError, 206
TelegramAPIServer (*class in aiogram.bot.api*), 72
TelegramObject (*class in aiogram.types.base*), 74
TerminatedByOtherGetUpdates, 209
text (*aiogram.dispatcher.filters.Command.CommandObj property*), 172
Text (*class in aiogram.dispatcher.filters*), 175
text() (*in module aiogram.utils.markdown*), 210
TextField (*class in aiogram.types.fields*), 78
throttle() (*aiogram.Dispatcher method*), 198
Throttled, 210
throttled() (*aiogram.Dispatcher method*), 199
TimeoutWarning, 206
to_object() (*aiogram.types.base.TelegramObject class method*), 74
to_object() (*aiogram.types.chat_member.ChatMember class method*), 132

to_object() (*aiogram.types.input_file.InputFile class method*), 127
to_python() (*aiogram.types.base.TelegramObject method*), 74
to_python() (*aiogram.types.input_file.InputFile method*), 127
to_python() (*aiogram.types.input_media.MediaGroup method*), 111
ToMuchMessages, 207
TypeOfFileMismatch, 209
typing() (*aiogram.types.chat.ChatActions class method*), 98

U

Unauthorized, 209
UnavailableMembers, 209
unban() (*aiogram.types.chat.Chat method*), 91
unban_chat_member() (*aiogram.bot.bot.Bot method*), 42
unban_chat_sender_chat() (*aiogram.bot.bot.Bot method*), 45
unban_sender_chat() (*aiogram.types.chat.Chat method*), 96
unbind() (*aiogram.dispatcher.filters.FiltersFactory method*), 170
unbind_filter() (*aiogram.Dispatcher method*), 200
underline() (*in module aiogram.utils.markdown*), 212
unhide_general_forum_topic() (*aiogram.bot.bot.Bot method*), 51
unpin() (*aiogram.types.message.Message method*), 165
unpin_all_chat_messages() (*aiogram.bot.bot.Bot method*), 49
unpin_all_forum_topic_messages() (*aiogram.bot.bot.Bot method*), 55
unpin_all_messages() (*aiogram.types.chat.Chat method*), 94
unpin_chat_message() (*aiogram.bot.bot.Bot method*), 49
unpin_message() (*aiogram.types.chat.Chat method*), 94
UnsupportedUrlProtocol, 209
Update (*class in aiogram.types.update*), 130
update_chat() (*aiogram.types.chat.Chat method*), 89
upload_audio() (*aiogram.types.chat.ChatActions class method*), 99
upload_document() (*aiogram.types.chat.ChatActions class method*), 99
upload_photo() (*aiogram.types.chat.ChatActions class method*), 98
upload_sticker_file() (*aiogram.bot.bot.Bot method*), 62
upload_video() (*aiogram.types.chat.ChatActions class method*), 99
upload_video_note() (*aiogram.types.chat.ChatActions class method*), 100
upload_voice() (*aiogram.types.chat.ChatActions class method*), 99
url (*aiogram.types.message.Message property*), 136
URLHostIsEmpty, 208
User (*class in aiogram.types.user*), 85
UserDeactivated, 209
UserIsAnAdministratorOfTheChat, 209
UserProfilePhotos (*class in aiogram.types.user_profile_photos*), 169

V

validate() (*aiogram.dispatcher.filters.AbstractFilter class method*), 181
validate() (*aiogram.dispatcher.filters.AdminFilter class method*), 179
validate() (*aiogram.dispatcher.filters.BoundFilter class method*), 182
validate() (*aiogram.dispatcher.filters.builtin.IDFilter class method*), 179
validate() (*aiogram.dispatcher.filters.Command class method*), 171
validate() (*aiogram.dispatcher.filters.Filter class method*), 182
validate() (*aiogram.dispatcher.filters.HashTag class method*), 176
validate() (*aiogram.dispatcher.filters.Regexp class method*), 176
validate() (*aiogram.dispatcher.filters.Text class method*), 175
ValidationError, 206
values (*aiogram.types.base.TelegramObject property*), 74
Venue (*class in aiogram.types.venue*), 131
Video (*class in aiogram.types.video*), 85
VideoNote (*class in aiogram.types.video_note*), 131
Voice (*class in aiogram.types.voice*), 127

W

wait_closed() (*aiogram.Dispatcher method*), 187
WebhookInfo (*class in aiogram.types.webhook_info*), 132
WebhookRequireHTTPS, 209
WrongFileIdentifier, 208
WrongRemoteFileIdSpecified, 209