# aiogram Documentation

***Release 2.0.1***

**Illemius / Alex Root Junior**

**Jan 12, 2019**

# Contents

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

# CHAPTER 1

# 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 2

# Features

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

# CHAPTER 3

# Contribute

- Issue Tracker
- Source Code

CHAPTER 4

# Contents

## 4.1 Installation Guide

### 4.1.1 Using PIP

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

### 4.1.2 From sources

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

or if you want to install development version (maybe unstable):

```
$ git clone https://github.com/aiogram/aiogram.git
$ cd aiogram
$ git checkout dev-2.x
$ python setup.py install
```

### 4.1.3 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 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
  ```

In addition, you don't need do nothing, *aiogram* is automatically starts using that if 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:

```
async def send_welcome(message: types.Message):
    """
    This handler will be called when client send `/start` or `/help` commands.
    """
    await message.reply("Hi!\nI'm EchoBot!\nPowered by aiogram.")
```

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

```
@dp.message_handler(regexp='(^cat[s]?$|puss)')
async def cats(message: types.Message):
    with open('data/cats.jpg', 'rb') as photo:
```

Last step: run long polling.

### 4.2.2 Summary

```python
"""
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(regexp='(^cat[s]?$|puss)')
async def cats(message: types.Message):
    with open('data/cats.jpg', 'rb') as photo:
        await bot.send_photo(message.chat.id, photo, caption='Cats is here ',
                             reply_to_message_id=message.message_id)


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

(continues on next page)

(continued from previous page)

```
    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_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://aiogram.readthedocs.io/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: Union[asyncio.base_events.BaseEventLoop, asyncio.events.AbstractEventLoop, None] = None*, *connections_limit: Optional[Integer] = None*, *proxy: Optional[String] = None*, *proxy_auth: Optional[aiohttp.helpers.BasicAuth] = None*, *validate_token: Optional[Boolean] = True*, *parse_mode=None*)

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

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

**close**()

Close all client sessions

**download_file**(*file_path: String*, *destination: Optional[InputFile] = None*, *timeout: Optional[Integer] = 30*, *chunk_size: Optional[Integer] = 65536*, *seek: Optional[Boolean] = True*) → Union[_io.BytesIO, _io.FileIO]

Download file by file_path to destination

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

**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.

**Returns** destination

**`request`** (*method: String*, *data: Optional[Dict[KT, VT]] = None*, *files: Optional[Dict[KT, VT]] = None*, ***kwargs*) → Union[List[T], Dict[KT, VT], 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`

**`send_file`** (*file_type*, *method*, *file*, *payload*) → Union[Dict[KT, VT], 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: Union[asyncio.base_events.BaseEventLoop, asyncio.events.AbstractEventLoop, None] = None*, *connections_limit: Optional[Integer] = None*, *proxy: Optional[String] = None*, *proxy_auth: Optional[aiohttp.helpers.BasicAuth] = None*, *validate_token: Optional[Boolean] = True*, *parse_mode=None*)

Bases: `aiogram.bot.base.BaseBot`, `aiogram.utils.mixins.DataMixin`, `aiogram.utils.mixins.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

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

**`me`**

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

**Returns** `aiogram.types.User`

**`add_sticker_to_set`**(*user_id: Integer*, *name: String*, *png_sticker: Union[InputFile, String]*, *emojis: String*, *mask_position: Optional[aiogram.types.mask_position.MaskPosition] = None*) → Boolean

Use this method to add a new sticker to a set created by the bot.

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.
- **`emojis`** (`base.String`) – One or more emoji corresponding to the sticker
- **`mask_position`** (`typing.Union[types.MaskPosition, None]`) – A JSON-serialized object for position where the mask should be placed on faces

**Returns** Returns True on success

**Return type** `base.Boolean`

**`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.Union[base.String, None]`) – Text of the notification. If not specified, nothing will be shown to the user, 0-1024 characters
- **`show_alert`** (`typing.Union[base.Boolean, None]`) – 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.Union[base.String, None]`) – URL that will be opened by the user's client
- **cache_time** (`typing.Union[base.Integer, None]`) – 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`

**answer_inline_query**(*inline_query_id: String, results: List[aiogram.types.inline_query_result.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.Union[base.Integer, None]`) – 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.Union[base.Boolean, None]`) – 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.Union[base.String, None]`) – 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.Union[base.String, None]`) – 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.Union[base.String, None]`) – 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`

**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.Union[base.String, None]`) – 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`

**answer_shipping_query**(*shipping_query_id: String, ok: Boolean, shipping_options: Optional[List[aiogram.types.shipping_option.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.Union[base.String, None]`) – 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`

**create_new_sticker_set**(*user_id: Integer, name: String, title: String, png_sticker: Union[InputFile, String], emojis: String, contains_masks: Optional[Boolean] = None, mask_position: Optional[aiogram.types.mask_position.MaskPosition] = None*) → Boolean

Use this method to create new sticker set owned by a user. The bot will be able to edit the created sticker set.

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)
- **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.
- **emojis** (`base.String`) – One or more emoji corresponding to the sticker
- **contains_masks** (`typing.Union[base.Boolean, None]`) – Pass True, if a set of mask stickers should be created
- **mask_position** (`typing.Union[types.MaskPosition, None]`) – A JSON-serialized object for position where the mask should be placed on faces

**Returns** Returns True on success

**Return type** `base.Boolean`

**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`

**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`

**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 groups and supergroups. - 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.

The following methods and objects allow your bot to handle stickers and sticker sets.

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`

**`delete_sticker_from_set`**(*sticker: String*) → Boolean

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

The following methods and objects allow your bot to work in inline mode.

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`

**`delete_webhook`**() → Boolean

Use this method to remove webhook integration if you decide to switch back to getUpdates. Returns True on success. Requires no parameters.

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

**Returns** Returns True on success

**Return type** `base.Boolean`

**`download_file_by_id`**(*file_id: String*, *destination=None*, *timeout: Integer = 30*, *chunk_size: Integer = 65536*, *seek: Boolean = True*)

Download file by file_id to destination

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

**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

**Returns** destination

**`edit_message_caption`**(*chat_id: Union[Integer, String, None] = None*, *message_id: Optional[Integer] = None*, *inline_message_id: Optional[String] = None*, *caption: Optional[String] = None*, *parse_mode: Optional[String] = None*, *reply_markup: Optional[aiogram.types.inline_keyboard.InlineKeyboardMarkup] = None*) → aiogram.types.message.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.Union[base.Integer, None]`) – Required if inline_message_id is not specified. Identifier of the sent message

- **inline_message_id** (`typing.Union[base.String, None]`) – Required if chat_id and message_id are not specified. Identifier of the inline message
- **caption** (`typing.Union[base.String, None]`) – New caption of the message
- **parse_mode** (`typing.Union[base.String, None]`) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, None]`) – 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]`

**edit_message_live_location**(*latitude: Float*, *longitude: Float*, *chat_id: Union[Integer*, *String*, *None] = None*, *message_id: Optional[Integer] = None*, *inline_message_id: Optional[String] = None*, *reply_markup: Optional[aiogram.types.inline_keyboard.InlineKeyboardMarkup] = None*) → aiogram.types.message.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.Union[base.Integer, None]`) – Required if inline_message_id is not specified. Identifier of the sent message
- **inline_message_id** (`typing.Union[base.String, None]`) – 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
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, None]`) – 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]`

**edit_message_media**(*media: aiogram.types.input_media.InputMedia*, *chat_id: Union[Integer*, *String*, *None] = None*, *message_id: Optional[Integer] = None*, *inline_message_id: Optional[String] = None*, *reply_markup: Optional[aiogram.types.inline_keyboard.InlineKeyboardMarkup] = None*) → Union[aiogram.types.message.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.Union[base.Integer, None]`) – Required if inline_message_id is not specified. Identifier of the sent message
- **inline_message_id** (`typing.Union[base.String, None]`) – 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.Union[types.InlineKeyboardMarkup, None]`) – 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]`

**edit_message_reply_markup**(*chat_id: Union[Integer, String, None] = None, message_id: Optional[Integer] = None, inline_message_id: Optional[String] = None, reply_markup: Optional[aiogram.types.inline_keyboard.InlineKeyboardMarkup] = None*) → aiogram.types.message.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.Union[base.Integer, None]`) – Required if inline_message_id is not specified. Identifier of the sent message
- **inline_message_id** (`typing.Union[base.String, None]`) – Required if chat_id and message_id are not specified. Identifier of the inline message
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, None]`) – 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]`

**edit_message_text**(*text: String, chat_id: Union[Integer, String, None] = None, message_id: Optional[Integer] = None, inline_message_id: Optional[String] = None, parse_mode: Optional[String] = None, disable_web_page_preview: Optional[Boolean] = None, reply_markup: Optional[aiogram.types.inline_keyboard.InlineKeyboardMarkup] = None*) → aiogram.types.message.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.Union[base.Integer, None]`) – Required if inline_message_id is not specified. Identifier of the sent message
- **inline_message_id** (`typing.Union[base.String, None]`) – 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.Union[base.String, None]`) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.
- **disable_web_page_preview** (`typing.Union[base.Boolean, None]`) – Disables link previews for links in this message
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, None]`) – 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]`

**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`

**forward_message**(*chat_id: Union[Integer, String], from_chat_id: Union[Integer, String], message_id: Integer, disable_notification: Optional[Boolean] = None*) → aiogram.types.message.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
- **from_chat_id** (`typing.Union[base.Integer, base.String]`) – Unique identifier for the chat where the original message was sent
- **disable_notification** (`typing.Union[base.Boolean, None]`) – Sends the message silently. Users will receive a notification with no sound
- **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`

**get_chat** (*chat_id: Union[Integer, String]*) → aiogram.types.chat.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`

**get_chat_administrators** (*chat_id: Union[Integer, String]*) → List[aiogram.types.chat_member.ChatMember]

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]`

**get_chat_member** (*chat_id: Union[Integer, String], user_id: Integer*) → aiogram.types.chat_member.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`

**get_chat_members_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#getchatmemberscount

**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`

**get_file** (*file_id: String*) → aiogram.types.file.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`

**get_game_high_scores**(*user_id: Integer*, *chat_id: Optional[Integer] = None*, *message_id: Optional[Integer] = None*, *inline_message_id: Optional[String] = None*) → List[aiogram.types.game_high_score.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.Union[base.Integer, None]`) – Required if inline_message_id is not specified. Unique identifier for the target chat
- **message_id** (`typing.Union[base.Integer, None]`) – Required if inline_message_id is not specified. Identifier of the sent message
- **inline_message_id** (`typing.Union[base.String, None]`) – 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]`

**get_me**() → aiogram.types.user.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`

**get_sticker_set**(*name: String*) → aiogram.types.sticker_set.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`

**get_updates**(*offset: Optional[Integer] = None*, *limit: Optional[Integer] = None*, *timeout: Optional[Integer] = None*, *allowed_updates: Optional[List[String]] = None*) → List[aiogram.types.update.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.Union[base.Integer, None]`) – Identifier of the first update to be returned
- **limit** (`typing.Union[base.Integer, None]`) – Limits the number of updates to be retrieved
- **timeout** (`typing.Union[base.Integer, None]`) – 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]`

**get_user_profile_photos**(*user_id: Integer*, *offset: Optional[Integer] = None*, *limit: Optional[Integer] = None*) → aiogram.types.user_profile_photos.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.Union[base.Integer, None]`) – Sequential number of the first photo to be returned. By default, all photos are returned
- **limit** (`typing.Union[base.Integer, None]`) – 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`

**get_webhook_info**() → aiogram.types.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`

**kick_chat_member**(*chat_id: Union[Integer, String]*, *user_id: Integer*, *until_date: Optional[Integer] = 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 group 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.

Note: In regular groups (non-supergroups), this method will only work if the 'All Members Are Admins' setting is off in the target group. Otherwise members may only be removed by the group's creator or by the member that added them.

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

**Parameters**

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

- **user_id** (`base.Integer`) – Unique identifier of the target user
- **until_date** (`typing.Union[base.Integer, None]`) – Date when the user will be unbanned, unix time

**Returns** Returns True on success

**Return type** `base.Boolean`

**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`

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

Use this method to pin a message in a supergroup. 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#pinchatmessage

**Parameters**

- **chat_id** (`typing.Union[base.Integer, base.String]`) – Unique identifier for the target chat or username of the target supergroup
- **message_id** (`base.Integer`) – Identifier of a message to pin
- **disable_notification** (`typing.Union[base.Boolean, None]`) – 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`

**promote_chat_member** (*chat_id: Union[Integer, String], user_id: Integer, 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_invite_users: Optional[Boolean] = None, can_restrict_members: Optional[Boolean] = None, can_pin_messages: Optional[Boolean] = None, can_promote_members: 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
- **can_change_info** (`typing.Union[base.Boolean, None]`) – Pass True, if the administrator can change chat title, photo and other settings

- **can_post_messages** (typing.Union[base.Boolean, None]) – Pass True, if the administrator can create channel posts, channels only
- **can_edit_messages** (typing.Union[base.Boolean, None]) – Pass True, if the administrator can edit messages of other users, channels only
- **can_delete_messages** (typing.Union[base.Boolean, None]) – Pass True, if the administrator can delete messages of other users
- **can_invite_users** (typing.Union[base.Boolean, None]) – Pass True, if the administrator can invite new users to the chat
- **can_restrict_members** (typing.Union[base.Boolean, None]) – Pass True, if the administrator can restrict, ban or unban chat members
- **can_pin_messages** (typing.Union[base.Boolean, None]) – Pass True, if the administrator can pin messages, supergroups only
- **can_promote_members** (typing.Union[base.Boolean, None]) – 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

**restrict_chat_member**(*chat_id: Union[Integer, String], user_id: Integer, until_date: Optional[Integer] = 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
- **until_date** (typing.Union[base.Integer, None]) – Date when restrictions will be lifted for the user, unix time
- **can_send_messages** (typing.Union[base.Boolean, None]) – Pass True, if the user can send text messages, contacts, locations and venues
- **can_send_media_messages** (typing.Union[base.Boolean, None]) – 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.Union[base.Boolean, None]) – 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.Union[base.Boolean, None]) – 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`

**`send_animation`**(*chat_id: Union[Integer, String], animation: Union[InputFile, String], duration: Optional[Integer] = None, width: Optional[Integer] = None, height: Optional[Integer] = None, thumb: Union[InputFile, String, None] = None, caption: Optional[String] = None, parse_mode: Optional[String] = None, disable_notification: Optional[Boolean] = None, reply_to_message_id: Optional[Integer] = None, reply_markup: Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup, aiogram.types.reply_keyboard.ReplyKeyboardMarkup, aiogram.types.reply_keyboard.ReplyKeyboardRemove, aiogram.types.force_reply.ForceReply, None] = None*) → aiogram.types.message.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)
- **`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.Union[base.Integer, None]`) – Duration of sent animation in seconds
- **`width`** (`typing.Union[base.Integer, None]`) – Animation width
- **`height`** (`typing.Union[base.Integer, None]`) – 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 90.
- **`caption`** (`typing.Union[base.String, None]`) – Animation caption (may also be used when resending animation by file_id), 0-1024 characters
- **`parse_mode`** (`typing.Union[base.String, None]`) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption
- **`disable_notification`** (`typing.Union[base.Boolean, None]`) – Sends the message silently. Users will receive a notification with no sound
- **`reply_to_message_id`** (`typing.Union[base.Integer, None]`) – If the message is a reply, ID of the original message
- **`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

**Returns** On success, the sent Message is returned

**Return type** `types.Message`

**`send_audio`**(*chat_id: Union[Integer, String], audio: Union[InputFile, String], caption: Optional[String] = None, parse_mode: Optional[String] = None, duration: Optional[Integer] = None, performer: Optional[String] = None, title: Optional[String] = None, thumb: Union[InputFile, String, None] = None, disable_notification: Optional[Boolean] = None, reply_to_message_id: Optional[Integer] = None, reply_markup: Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup, aiogram.types.reply_keyboard.ReplyKeyboardMarkup, aiogram.types.reply_keyboard.ReplyKeyboardRemove, aiogram.types.force_reply.ForceReply, None] = None*) → aiogram.types.message.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
- **`audio`** (`typing.Union[base.InputFile, base.String]`) – Audio file to send
- **`caption`** (`typing.Union[base.String, None]`) – Audio caption, 0-1024 characters
- **`parse_mode`** (`typing.Union[base.String, None]`) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.
- **`duration`** (`typing.Union[base.Integer, None]`) – Duration of the audio in seconds
- **`performer`** (`typing.Union[base.String, None]`) – Performer
- **`title`** (`typing.Union[base.String, None]`) – Track name
- **`thumb`** (`typing.Union[base.InputFile, base.String, None]`) – Thumbnail of the file sent
- **`disable_notification`** (`typing.Union[base.Boolean, None]`) – Sends the message silently. Users will receive a notification with no sound
- **`reply_to_message_id`** (`typing.Union[base.Integer, None]`) – If the message is a reply, ID of the original message
- **`reply_markup`** (`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]`) – Additional interface options

**Returns** On success, the sent Message is returned

**Return type** `types.Message`

**`send_chat_action`**(*chat_id: Union[Integer, String], action: String*) → 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**

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

**Returns** Returns True on success

**Return type** `base.Boolean`

**`send_contact`**(*chat_id: Union[Integer, String], phone_number: String, first_name: String, last_name: Optional[String] = None, vcard: Optional[String] = None, disable_notification: Optional[Boolean] = None, reply_to_message_id: Optional[Integer] = None, reply_markup: Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup, aiogram.types.reply_keyboard.ReplyKeyboardMarkup, aiogram.types.reply_keyboard.ReplyKeyboardRemove, aiogram.types.force_reply.ForceReply, None] = None*) → aiogram.types.message.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
- **`phone_number`** (`base.String`) – Contact's phone number
- **`first_name`** (`base.String`) – Contact's first name
- **`last_name`** (`typing.Union[base.String, None]`) – Contact's last name
- **`vcard`** (`typing.Union[base.String, None]`) – vcard
- **`disable_notification`** (`typing.Union[base.Boolean, None]`) – Sends the message silently. Users will receive a notification with no sound
- **`reply_to_message_id`** (`typing.Union[base.Integer, None]`) – If the message is a reply, ID of the original message
- **`reply_markup`** (`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]`) – Additional interface options

**Returns** On success, the sent Message is returned

**Return type** `types.Message`

**`send_document`**(*chat_id: Union[Integer, String], document: Union[InputFile, String], thumb: Union[InputFile, String, None] = None, caption: Optional[String] = None, parse_mode: Optional[String] = None, disable_notification: Optional[Boolean] = None, reply_to_message_id: Optional[Integer] = None, reply_markup: Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup, aiogram.types.reply_keyboard.ReplyKeyboardMarkup, aiogram.types.reply_keyboard.ReplyKeyboardRemove, aiogram.types.force_reply.ForceReply, None] = None*) → aiogram.types.message.Message

Use this method to send general files.

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
- **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.Union[base.String, None]`) – Document caption (may also be used when resending documents by file_id), 0-1024 characters
- **parse_mode** (`typing.Union[base.String, None]`) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.
- **disable_notification** (`typing.Union[base.Boolean, None]`) – Sends the message silently. Users will receive a notification with no sound
- **reply_to_message_id** (`typing.Union[base.Integer, None]`) – If the message is a reply, ID of the original message
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply], None]`) – Additional interface options

**Returns** On success, the sent Message is returned

**Return type** `types.Message`

**send_game**(*chat_id: Integer*, *game_short_name: String*, *disable_notification: Optional[Boolean] = None*, *reply_to_message_id: Optional[Integer] = None*, *reply_markup: Optional[aiogram.types.inline_keyboard.InlineKeyboardMarkup] = None*) → aiogram.types.message.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.Union[base.Boolean, None]`) – Sends the message silently. Users will receive a notification with no sound
- **reply_to_message_id** (`typing.Union[base.Integer, None]`) – If the message is a reply, ID of the original message
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, None]`) – 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`

**send_invoice**(*chat_id: Integer, title: String, description: String, payload: String, provider_token: String, start_parameter: String, currency: String, prices: List[aiogram.types.labeled_price.LabeledPrice], provider_data: Optional[Dict[KT, VT]] = 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, is_flexible: Optional[Boolean] = None, disable_notification: Optional[Boolean] = None, reply_to_message_id: Optional[Integer] = None, reply_markup: Optional[aiogram.types.inline_keyboard.InlineKeyboardMarkup] = None*) → aiogram.types.message.Message

Use this method to send invoices.

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

**Parameters**

- **chat_id** (`base.Integer`) – Unique identifier for the target private chat
- **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
- **start_parameter** (`base.String`) – Unique deep-linking parameter that can be used to generate this invoice when used as a start parameter
- **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.)
- **provider_data** (`typing.Union[typing.Dict, None]`) – JSON-encoded data about the invoice, which will be shared with the payment provider
- **photo_url** (`typing.Union[base.String, None]`) – URL of the product photo for the invoice
- **photo_size** (`typing.Union[base.Integer, None]`) – Photo size
- **photo_width** (`typing.Union[base.Integer, None]`) – Photo width
- **photo_height** (`typing.Union[base.Integer, None]`) – Photo height
- **need_name** (`typing.Union[base.Boolean, None]`) – Pass True, if you require the user’s full name to complete the order
- **need_phone_number** (`typing.Union[base.Boolean, None]`) – Pass True, if you require the user’s phone number to complete the order
- **need_email** (`typing.Union[base.Boolean, None]`) – Pass True, if you require the user’s email to complete the order
- **need_shipping_address** (`typing.Union[base.Boolean, None]`) – Pass True, if you require the user’s shipping address to complete the order

- **is_flexible** (`typing.Union[base.Boolean, None]`) – Pass True, if the final price depends on the shipping method
- **disable_notification** (`typing.Union[base.Boolean, None]`) – Sends the message silently. Users will receive a notification with no sound
- **reply_to_message_id** (`typing.Union[base.Integer, None]`) – If the message is a reply, ID of the original message
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, None]`) – 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`

**send_location**(*chat_id: Union[Integer, String], latitude: Float, longitude: Float, live_period: Optional[Integer] = None, disable_notification: Optional[Boolean] = None, reply_to_message_id: Optional[Integer] = None, reply_markup: Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup, aiogram.types.reply_keyboard.ReplyKeyboardMarkup, aiogram.types.reply_keyboard.ReplyKeyboardRemove, aiogram.types.force_reply.ForceReply, None] = None*) → aiogram.types.message.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
- **latitude** (`base.Float`) – Latitude of the location
- **longitude** (`base.Float`) – Longitude of the location
- **live_period** (`typing.Union[base.Integer, None]`) – Period in seconds for which the location will be updated
- **disable_notification** (`typing.Union[base.Boolean, None]`) – Sends the message silently. Users will receive a notification with no sound
- **reply_to_message_id** (`typing.Union[base.Integer, None]`) – If the message is a reply, ID of the original message
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]`) – Additional interface options

**Returns** On success, the sent Message is returned

**Return type** `types.Message`

**send_media_group**(*chat_id: Union[Integer, String], media: Union[aiogram.types.input_media.MediaGroup, List[T]], disable_notification: Optional[Boolean] = None, reply_to_message_id: Optional[Integer] = None*) → List[aiogram.types.message.Message]

Use this method to send a group of photos or videos as an album.

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
- **media** (`typing.Union[types.MediaGroup, typing.List]`) – A JSON-serialized array describing photos and videos to be sent
- **disable_notification** (`typing.Union[base.Boolean, None]`) – Sends the message silently. Users will receive a notification with no sound
- **reply_to_message_id** (`typing.Union[base.Integer, None]`) – If the message is a reply, ID of the original message

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

**Return type** typing.List[types.Message]

**send_message**(*chat_id: Union[Integer, String], text: String, parse_mode: Optional[String] = None, disable_web_page_preview: Optional[Boolean] = None, disable_notification: Optional[Boolean] = None, reply_to_message_id: Optional[Integer] = None, reply_markup: Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup, aiogram.types.reply_keyboard.ReplyKeyboardMarkup, aiogram.types.reply_keyboard.ReplyKeyboardRemove, aiogram.types.force_reply.ForceReply, None] = None*) → aiogram.types.message.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
- **text** (`base.String`) – Text of the message to be sent
- **parse_mode** (`typing.Union[base.String, None]`) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.
- **disable_web_page_preview** (`typing.Union[base.Boolean, None]`) – Disables link previews for links in this message
- **disable_notification** (`typing.Union[base.Boolean, None]`) – Sends the message silently. Users will receive a notification with no sound
- **reply_to_message_id** (`typing.Union[base.Integer, None]`) – If the message is a reply, ID of the original message
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]`) – Additional interface options

**Returns** On success, the sent Message is returned

**Return type** `types.Message`

**send_photo**(*chat_id: Union[Integer, String], photo: Union[InputFile, String], caption: Optional[String] = None, parse_mode: Optional[String] = None, disable_notification: Optional[Boolean] = None, reply_to_message_id: Optional[Integer] = None, reply_markup: Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup, aiogram.types.reply_keyboard.ReplyKeyboardMarkup, aiogram.types.reply_keyboard.ReplyKeyboardRemove, aiogram.types.force_reply.ForceReply, None] = None*) → aiogram.types.message.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
- **`photo`** (`typing.Union[base.InputFile, base.String]`) – Photo to send
- **`caption`** (`typing.Union[base.String, None]`) – Photo caption (may also be used when resending photos by file_id), 0-1024 characters
- **`parse_mode`** (`typing.Union[base.String, None]`) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.
- **`disable_notification`** (`typing.Union[base.Boolean, None]`) – Sends the message silently. Users will receive a notification with no sound
- **`reply_to_message_id`** (`typing.Union[base.Integer, None]`) – If the message is a reply, ID of the original message
- **`reply_markup`** (`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]`) – Additional interface options

**Returns** On success, the sent Message is returned

**Return type** `types.Message`

**`send_sticker`**(*chat_id: Union[Integer, String], sticker: Union[InputFile, String], disable_notification: Optional[Boolean] = None, reply_to_message_id: Optional[Integer] = None, reply_markup: Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup, aiogram.types.reply_keyboard.ReplyKeyboardMarkup, aiogram.types.reply_keyboard.ReplyKeyboardRemove, aiogram.types.force_reply.ForceReply, None] = None*) → aiogram.types.message.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
- **`sticker`** (`typing.Union[base.InputFile, base.String]`) – Sticker to send
- **`disable_notification`** (`typing.Union[base.Boolean, None]`) – Sends the message silently. Users will receive a notification with no sound
- **`reply_to_message_id`** (`typing.Union[base.Integer, None]`) – If the message is a reply, ID of the original message
- **`reply_markup`** (`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]`) – Additional interface options

**Returns** On success, the sent Message is returned

**Return type** `types.Message`

**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, disable_notification: Optional[Boolean] = None, reply_to_message_id: Optional[Integer] = None, reply_markup: Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup, aiogram.types.reply_keyboard.ReplyKeyboardMarkup, aiogram.types.reply_keyboard.ReplyKeyboardRemove, aiogram.types.force_reply.ForceReply, None] = None*) → aiogram.types.message.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
- **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.Union[base.String, None]`) – Foursquare identifier of the venue
- **foursquare_type** (`typing.Union[base.String, None]`) – Foursquare type of the venue, if known
- **disable_notification** (`typing.Union[base.Boolean, None]`) – Sends the message silently. Users will receive a notification with no sound
- **reply_to_message_id** (`typing.Union[base.Integer, None]`) – If the message is a reply, ID of the original message
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]`) – Additional interface options

**Returns** On success, the sent Message is returned

**Return type** `types.Message`

**send_video**(*chat_id: Union[Integer, String], video: Union[InputFile, String], duration: Optional[Integer] = None, width: Optional[Integer] = None, height: Optional[Integer] = None, thumb: Union[InputFile, String, None] = None, caption: Optional[String] = None, parse_mode: Optional[String] = None, supports_streaming: Optional[Boolean] = None, disable_notification: Optional[Boolean] = None, reply_to_message_id: Optional[Integer] = None, reply_markup: Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup, aiogram.types.reply_keyboard.ReplyKeyboardMarkup, aiogram.types.reply_keyboard.ReplyKeyboardRemove, aiogram.types.force_reply.ForceReply, None] = None*) → aiogram.types.message.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
- **video** (`typing.Union[base.InputFile, base.String]`) – Video to send
- **duration** (`typing.Union[base.Integer, None]`) – Duration of sent video in seconds
- **width** (`typing.Union[base.Integer, None]`) – Video width
- **height** (`typing.Union[base.Integer, None]`) – Video height
- **thumb** (`typing.Union[base.InputFile, base.String, None]`) – Thumbnail of the file sent
- **caption** (`typing.Union[base.String, None]`) – Video caption (may also be used when resending videos by file_id), 0-1024 characters
- **parse_mode** (`typing.Union[base.String, None]`) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.
- **supports_streaming** (`typing.Union[base.Boolean, None]`) – Pass True, if the uploaded video is suitable for streaming
- **disable_notification** (`typing.Union[base.Boolean, None]`) – Sends the message silently. Users will receive a notification with no sound
- **reply_to_message_id** (`typing.Union[base.Integer, None]`) – If the message is a reply, ID of the original message
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]`) – Additional interface options

**Returns** On success, the sent Message is returned

**Return type** `types.Message`

**send_video_note**(*chat_id: Union[Integer, String], video_note: Union[InputFile, String], duration: Optional[Integer] = None, length: Optional[Integer] = None, thumb: Union[InputFile, String, None] = None, disable_notification: Optional[Boolean] = None, reply_to_message_id: Optional[Integer] = None, reply_markup: Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup, aiogram.types.reply_keyboard.ReplyKeyboardMarkup, aiogram.types.reply_keyboard.ReplyKeyboardRemove, aiogram.types.force_reply.ForceReply, None] = None*) → aiogram.types.message.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
- **video_note** (`typing.Union[base.InputFile, base.String]`) – Video note to send
- **duration** (`typing.Union[base.Integer, None]`) – Duration of sent video in seconds

- **length** (`typing.Union[base.Integer, None]`) – Video width and height
- **thumb** (`typing.Union[base.InputFile, base.String, None]`) – Thumbnail of the file sent
- **disable_notification** (`typing.Union[base.Boolean, None]`) – Sends the message silently. Users will receive a notification with no sound
- **reply_to_message_id** (`typing.Union[base.Integer, None]`) – If the message is a reply, ID of the original message
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]`) – Additional interface options

**Returns** On success, the sent Message is returned

**Return type** `types.Message`

**send_voice**(*chat_id: Union[Integer, String], voice: Union[InputFile, String], caption: Optional[String] = None, parse_mode: Optional[String] = None, duration: Optional[Integer] = None, disable_notification: Optional[Boolean] = None, reply_to_message_id: Optional[Integer] = None, reply_markup: Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup, aiogram.types.reply_keyboard.ReplyKeyboardMarkup, aiogram.types.reply_keyboard.ReplyKeyboardRemove, aiogram.types.force_reply.ForceReply, None] = None*) → aiogram.types.message.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
- **voice** (`typing.Union[base.InputFile, base.String]`) – Audio file to send
- **caption** (`typing.Union[base.String, None]`) – Voice message caption, 0-1024 characters
- **parse_mode** (`typing.Union[base.String, None]`) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.
- **duration** (`typing.Union[base.Integer, None]`) – Duration of the voice message in seconds
- **disable_notification** (`typing.Union[base.Boolean, None]`) – Sends the message silently. Users will receive a notification with no sound
- **reply_to_message_id** (`typing.Union[base.Integer, None]`) – If the message is a reply, ID of the original message
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]`) – Additional interface options

**Returns** On success, the sent Message is returned

**Return type** `types.Message`

**`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.Union[base.String, None]`) – New chat description, 0-255 characters

**Returns** Returns True on success

**Return type** `base.Boolean`

**`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`

**`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`

**`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`

**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*) → aiogram.types.message.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.Union[base.Boolean, None]`) – Pass True, if the high score is allowed to decrease This can be useful when fixing mistakes or banning cheaters
- **disable_edit_message** (`typing.Union[base.Boolean, None]`) – Pass True, if the game message should not be automatically edited to include the current scoreboard
- **chat_id** (`typing.Union[base.Integer, None]`) – Required if inline_message_id is not specified. Unique identifier for the target chat
- **message_id** (`typing.Union[base.Integer, None]`) – Required if inline_message_id is not specified. Identifier of the sent message
- **inline_message_id** (`typing.Union[base.String, None]`) – 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]`

**set_passport_data_errors**(*user_id: Integer, errors: List[aiogram.types.passport_element_error.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`

**`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`

**`set_webhook`**(*url: String*, *certificate: Optional[InputFile] = None*, *max_connections: Optional[Integer] = None*, *allowed_updates: Optional[List[String]] = 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.

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.Union[base.InputFile, None]`) – Upload your public key certificate so that the root certificate in use can be checked
- **`max_connections`** (`typing.Union[base.Integer, None]`) – Maximum allowed number of simultaneous HTTPS connections to the webhook for update delivery, 1-100.
- **`allowed_updates`** (`typing.Union[typing.List[base.String], None]`) – List the types of updates you want your bot to receive

**Returns** Returns true

**Return type** `base.Boolean`

**`stop_message_live_location`**(*chat_id: Union[Integer, String, None] = None*, *message_id: Optional[Integer] = None*, *inline_message_id: Optional[String] = None*, *reply_markup: Optional[aiogram.types.inline_keyboard.InlineKeyboardMarkup] = None*) → aiogram.types.message.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.Union[base.Integer, None]`) – Required if inline_message_id is not specified. Identifier of the sent message
- **inline_message_id** (`typing.Union[base.String, None]`) – Required if chat_id and message_id are not specified. Identifier of the inline message
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, None]`) – 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]`

**unban_chat_member**(*chat_id: Union[Integer, String], user_id: Integer*) → 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.

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

**Returns** Returns True on success

**Return type** `base.Boolean`

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

Use this method to unpin a message in a supergroup chat. 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#unpinchatmessage

**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`

**upload_sticker_file**(*user_id: Integer, png_sticker: InputFile*) → aiogram.types.file.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`

## API Helpers

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

Validate BOT token

**Parameters** **`token`** –

**Returns**

`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: `aiogram.utils.helper.Helper`

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

List is updated to Bot API 4.1

**`static api_url`**(*token*, *method*)

Generate API URL with included token and method name

**Parameters**

- **`token`** –
- **`method`** –

**Returns**

**`static file_url`**(*token*, *path*)

Generate File URL with included token and file path

**Parameters**

- **`token`** –
- **`path`** –

**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

## 4.4.2 Telegram data types

### Bases

### Base TelegramObject

### MetaTelegramObject

**class** `aiogram.types.base.`**`MetaTelegramObject`**

Bases: `type`

Metaclass for telegram objects

### TelegramObject

**class** `aiogram.types.base.`**`TelegramObject`**(*conf=None*, ***kwargs*)

Bases: `aiogram.utils.mixins.ContextInstanceMixin`

Abstract class for telegram objects

Deserialize object

**Parameters**

- **`conf`** –
- **`kwargs`** –

**`props`**

Get props

**Returns** dict with props

**`props_aliases`**

Get aliases for props

**Returns**

**`values`**

Get values

**Returns**

**classmethod `to_object`**(*data*)

Deserialize object

**Parameters `data`** –

**Returns**

**`to_python`**() → Dict[KT, VT]

Get object as JSON serializable

**Returns**

**clean**()
: Remove empty values

**as_json**() → str
: Get object as JSON string

  **Returns** JSON

  **Return type** `str`

**iter_keys**()
: Iterate over keys

  **Returns**

**iter_values**()
: 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**

**serialize**(*value*)
: Serialize value to python

**Parameters** **value** –

**Returns**

**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: *aiogram.types.fields.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: *aiogram.types.fields.Field*

Field contains list ob objects

**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: `aiogram.types.fields.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: `aiogram.types.fields.Field`

In this field st_ored 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.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: *aiogram.types.fields.Field*

**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

**download**(*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

**get_file**()

Get file information

**Returns** aiogram.types.File

## Types

### StickerSet

**class** aiogram.types.sticker_set.**StickerSet**(*conf=None*, ***kwargs*)

Bases: *aiogram.types.base.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=None*, ***kwargs*)

Bases: *aiogram.types.base.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=None*, ***kwargs*)

Bases: *aiogram.types.base.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** –

**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.Union[base.String, None]) – Text of the notification. If not specified, nothing will be shown to the user, 0-200 characters

- **show_alert** (typing.Union[base.Boolean, None]) – 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.Union[base.String, None]) – URL that will be opened by the user's client.
- **cache_time** (typing.Union[base.Integer, None]) – 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=None*, ***kwargs*)

Bases: *aiogram.types.base.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**(*conf=None*, ***kwargs*)

Bases: *aiogram.types.base.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: `aiogram.utils.helper.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** CODE

**Key** PRE

**Key** TEXT_LINK

**Key** TEXT_MENTION

### ShippingQuery

**class** `aiogram.types.shipping_query.`**`ShippingQuery`**(*conf=None*, ***kwargs*)
Bases: *`aiogram.types.base.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=None*, ***kwargs*)
Bases: *`aiogram.types.base.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*)

Bases: *aiogram.types.base.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

**add**(**args*)
Add buttons

**Parameters** **args** –

**Returns** self

**Return type** types.InlineKeyboardMarkup

**row**(**args*)
Add row

**Parameters** **args** –

**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: String = None, callback_data: String = None, switch_inline_query: String = None, switch_inline_query_current_chat: String = None, callback_game: aiogram.types.callback_game.CallbackGame = None, pay: Boolean = None*)

Bases: *aiogram.types.base.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

## User

**class** `aiogram.types.user.`**`User`**(*conf=None*, ***kwargs*)

Bases: *`aiogram.types.base.TelegramObject`*

This object represents a Telegram user or bot.

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

Deserialize object

**Parameters**

- **`conf`** –
- **`kwargs`** –

**`full_name`**

You can get full name of user.

**Returns** str

**`mention`**

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

**Returns** str

**`locale`**

Get user's locale

**Returns** `babel.core.Locale`

## Video

**class** `aiogram.types.video.`**`Video`**(*conf=None*, ***kwargs*)

Bases: *`aiogram.types.base.TelegramObject`*, *`aiogram.types.mixins.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=None*, ***kwargs*)

Bases: *`aiogram.types.base.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=None*, ***kwargs*)

Bases: *aiogram.types.base.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=None*, ***kwargs*)

Bases: *aiogram.types.base.TelegramObject*, *aiogram.types.mixins.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: *aiogram.types.base.TelegramObject*

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

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

### CallbackGame

**class** aiogram.types.callback_game.**CallbackGame**(*conf=None*, ***kwargs*)

Bases: *aiogram.types.base.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[aiogram.types.reply_keyboard.Keybo = None, resize_keyboard: Boolean = None, one_time_keyboard: Boolean = None, selective: Boolean = None, row_width: Integer = 3*)

Bases: *aiogram.types.base.TelegramObject*

This object represents a custom keyboard with reply options (see Introduction to bots for details and examples).

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

**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_contact: Boolean = None*, *request_location: Boolean = None*)

Bases: *aiogram.types.base.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 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.

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

### ReplyKeyboardRemove

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

Bases: *aiogram.types.base.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

### Chat

**class** aiogram.types.chat.**Chat**(*conf=None*, ***kwargs*)

Bases: *aiogram.types.base.TelegramObject*

This object represents a chat.

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

Deserialize object

**Parameters**

- **conf** –
- **kwargs** –

**mention**

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

**delete_photo**()

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`

**do** (*action*)

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.

**Returns** Returns True on success.

**Return type** `base.Boolean`

**export_invite_link** ()

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`

**get_administrators** ()

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[types.ChatMember]`

**get_member** (*user_id*)

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`

**get_members_count** ()

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

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

**Returns** Returns Int on success.

**Return type** `base.Integer`

**get_url** ()

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`

**kick** (*user_id: Integer, until_date: Optional[Integer] = None*)

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 group 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.

Note: In regular groups (non-supergroups), this method will only work if the 'All Members Are Admins' setting is off in the target group. Otherwise members may only be removed by the group's creator or by the member that added them.

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, None]`) – Date when the user will be unbanned, unix time.

**Returns** Returns True on success.

**Return type** `base.Boolean`

**leave**()

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`

**pin_message**(*message_id: int*, *disable_notification: bool = False*)

Use this method to pin a message in a supergroup. 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#pinchatmessage

**Parameters**

- **message_id** (`base.Integer`) – Identifier of a message to pin
- **disable_notification** (`typing.Union[base.Boolean, None]`) – 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`

**promote**(*user_id: Integer*, *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_invite_users: Optional[Boolean] = None*, *can_restrict_members: Optional[Boolean] = None*, *can_pin_messages: Optional[Boolean] = None*, *can_promote_members: 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
- **can_change_info** (`typing.Union[base.Boolean, None]`) – Pass True, if the administrator can change chat title, photo and other settings
- **can_post_messages** (`typing.Union[base.Boolean, None]`) – Pass True, if the administrator can create channel posts, channels only

- **can_edit_messages** (typing.Union[base.Boolean, None]) – Pass True, if the administrator can edit messages of other users, channels only
- **can_delete_messages** (typing.Union[base.Boolean, None]) – Pass True, if the administrator can delete messages of other users
- **can_invite_users** (typing.Union[base.Boolean, None]) – Pass True, if the administrator can invite new users to the chat
- **can_restrict_members** (typing.Union[base.Boolean, None]) – Pass True, if the administrator can restrict, ban or unban chat members
- **can_pin_messages** (typing.Union[base.Boolean, None]) – Pass True, if the administrator can pin messages, supergroups only
- **can_promote_members** (typing.Union[base.Boolean, None]) – 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

**restrict**(*user_id: Integer*, *until_date: Optional[Integer] = 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
- **until_date** (typing.Union[base.Integer, None]) – Date when restrictions will be lifted for the user, unix time.
- **can_send_messages** (typing.Union[base.Boolean, None]) – Pass True, if the user can send text messages, contacts, locations and venues
- **can_send_media_messages** (typing.Union[base.Boolean, None]) – 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.Union[base.Boolean, None]) – 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.Union[base.Boolean, None]) – 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

**set_description**(*description*)

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.Union[base.String, None]`) – New chat description, 0-255 characters

**Returns** Returns True on success.

**Return type** `base.Boolean`

**`set_photo`**(*photo*)

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`

**`set_title`**(*title*)

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`

**`unban`**(*user_id: Integer*)

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.

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

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

**Returns** Returns True on success.

**Return type** `base.Boolean`

**`unpin_message`**()

Use this method to unpin a message in a supergroup chat. 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#unpinchatmessage

**Returns** Returns True on success.

**Return type** `base.Boolean`

**`update_chat`**()

User this method to update Chat data

**Returns** None

### ChatType

**class** aiogram.types.chat.**ChatType**
Bases: `aiogram.utils.helper.Helper`

List of chat types

**Key** PRIVATE

**Key** GROUP

**Key** SUPER_GROUP

**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: `aiogram.utils.helper.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**

**classmethod find_location**(*sleep=None*)
Do find location

**Parameters** **sleep** – sleep timeout

**Returns**

**classmethod record_audio**(*sleep=None*)
Do record audio

**Parameters** **sleep** – sleep timeout

**Returns**

**classmethod record_video**(*sleep=None*)
Do record video

**Parameters** **sleep** – sleep timeout

**Returns**

**classmethod record_video_note**(*sleep=None*)
Do record video note

**Parameters** **sleep** – sleep timeout

**Returns**

**classmethod typing**(*sleep=None*)
Do typing

**Parameters** **sleep** – sleep timeout

**Returns**

**classmethod upload_audio**(*sleep=None*)
Do upload audio

**Parameters** **sleep** – sleep timeout

**Returns**

**classmethod upload_document**(*sleep=None*)
Do upload document

**Parameters** **`sleep`** – sleep timeout

**Returns**

**`classmethod upload_photo`**(*sleep=None*)
Do upload_photo

**Parameters** **`sleep`** – sleep timeout

**Returns**

**`classmethod upload_video`**(*sleep=None*)
Do upload video

**Parameters** **`sleep`** – sleep timeout

**Returns**

**`classmethod upload_video_note`**(*sleep=None*)
Do upload video note

**Parameters** **`sleep`** – sleep timeout

**Returns**

## Document

**`class`** `aiogram.types.document.`**`Document`**(*conf=None*, ***kwargs*)
Bases: *`aiogram.types.base.TelegramObject`*, *`aiogram.types.mixins.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=None*, ***kwargs*)
Bases: *`aiogram.types.base.TelegramObject`*, *`aiogram.types.mixins.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=None*, ***kwargs*)

Bases: *aiogram.types.base.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.

Example: A poll bot for groups runs in privacy mode (only receives commands, replies to its messages and mentions). There could be two ways to create a new poll

The last option is definitely more attractive. And if you use ForceReply in your bot's questions, it will receive the user's answers even if it only receives replies, commands and mentions — without any extra work for the user.

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

Deserialize object

**Parameters**

- **conf** –
- **kwargs** –

**classmethod create**(*selective: Optional[Boolean] = None*)

Create new force reply

**Parameters selective** –

**Returns**

### PassportElementError

**class** aiogram.types.passport_element_error.**PassportElementError**(*conf=None*, ***kwargs*)

Bases: *aiogram.types.base.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: *aiogram.types.passport_element_error.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

### PassportElementErrorFile

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

Bases: *aiogram.types.passport_element_error.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

### PassportElementErrorFiles

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

Bases: *aiogram.types.passport_element_error.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

### PassportElementErrorFrontSide

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

Bases: *aiogram.types.passport_element_error.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

### PassportElementErrorReverseSide

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

Bases: *aiogram.types.passport_element_error.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

### PassportElementErrorSelfie

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

Bases: *aiogram.types.passport_element_error.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

## ShippingAddress

**class** `aiogram.types.shipping_address.`**`ShippingAddress`**(*conf=None*, ***kwargs*)

Bases: `aiogram.types.base.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=None*, ***kwargs*)

Bases: `aiogram.types.base.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=None*, ***kwargs*)

Bases: `aiogram.types.base.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=None*, ***kwargs*)

Bases: `aiogram.types.base.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=None*, ***kwargs*)

Bases: *aiogram.types.base.TelegramObject*, *aiogram.types.mixins.Downloadable*

This object represents a sticker.

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

Deserialize object

**Parameters**

- **conf** –
- **kwargs** –

### InlineQuery

**class** aiogram.types.inline_query.**InlineQuery**(*conf=None*, ***kwargs*)

Bases: *aiogram.types.base.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** –

**answer**(*results: List[aiogram.types.inline_query_result.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.Union[base.Integer, None]) – 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.Union[base.Boolean, None]`) – 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.Union[base.String, None]`) – 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.Union[base.String, None]`) – 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.Union[base.String, None]`) – 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=None*, ***kwargs*)

Bases: `aiogram.types.base.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=None*, ***kwargs*)

Bases: `aiogram.types.base.TelegramObject`, `aiogram.types.mixins.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: `aiogram.types.base.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

### InputMediaAnimation

**class** aiogram.types.input_media.**InputMediaAnimation**(*media: InputFile*, *thumb: Union[InputFile, String] = None*, *caption: String = None*, *width: Integer = None*, *height: Integer = None*, *duration: Integer = None*, *parse_mode: Boolean = None*, ***kwargs*)

Bases: `aiogram.types.input_media.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

### InputMediaDocument

**class** aiogram.types.input_media.**InputMediaDocument**(*media: InputFile*, *thumb: Union[InputFile, String] = None*, *caption: String = None*, *parse_mode: Boolean = None*, ***kwargs*)

Bases: `aiogram.types.input_media.InputMedia`

Represents a photo to be sent.

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

### InputMediaAudio

**class** aiogram.types.input_media.**InputMediaAudio**(*media: InputFile, thumb: Union[InputFile, String] = None, caption: String = None, width: Integer = None, height: Integer = None, duration: Integer = None, performer: String = None, title: String = None, parse_mode: Boolean = None, **kwargs*)

Bases: *aiogram.types.input_media.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

### InputMediaPhoto

**class** aiogram.types.input_media.**InputMediaPhoto**(*media: InputFile, thumb: Union[InputFile, String] = None, caption: String = None, parse_mode: Boolean = None, **kwargs*)

Bases: *aiogram.types.input_media.InputMedia*

Represents a photo to be sent.

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

### InputMediaVideo

**class** aiogram.types.input_media.**InputMediaVideo**(*media: InputFile, thumb: Union[InputFile, String] = None, caption: String = None, width: Integer = None, height: Integer = None, duration: Integer = None, parse_mode: Boolean = None, supports_streaming: Boolean = None, **kwargs*)

Bases: *aiogram.types.input_media.InputMedia*

Represents a video to be sent.

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

### MediaGroup

**class** aiogram.types.input_media.**MediaGroup**(*medias: Optional[List[Union[aiogram.types.input_media.InputMedia, Dict[KT, VT]]]] = None*)

Bases: *aiogram.types.base.TelegramObject*

Helper for sending media group

**attach_many**(**medias*)

Attach list of media

**Parameters medias** –

**attach**(*media: Union[aiogram.types.input_media.InputMedia, Dict[KT, VT]]*)
Attach media

**Parameters** **media** –

**attach_photo**(*photo: Union[aiogram.types.input_media.InputMediaPhoto, InputFile], caption: String = None*)
Attach photo

**Parameters**

- **photo** –
- **caption** –

**attach_video**(*video: Union[aiogram.types.input_media.InputMediaVideo, InputFile], thumb: Union[InputFile, String] = None, caption: String = None, width: Integer = None, height: Integer = None, duration: Integer = None*)
Attach video

**Parameters**

- **video** –
- **caption** –
- **width** –
- **height** –
- **duration** –

**to_python**() → List[T]
Get object as JSON serializable

**Returns**

### InlineQueryResult

**class** aiogram.types.inline_query_result.**InlineQueryResult**(***kwargs*)
Bases: *aiogram.types.base.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

### InlineQueryResultArticle

**class** aiogram.types.inline_query_result.**InlineQueryResultArticle**(**, id: String, title: String, input_message_content: aiogram.types.input_message_content re-ply_markup: Optional[aiogram.types.inline_keyboard. = 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: *aiogram.types.inline_query_result.InlineQueryResult*

Represents a link to an article or web page.

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

### 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*, *reply_markup: Optional[aiogram.types.inline_keyboard.Inl = None*, *input_message_content: Optional[aiogram.types.input_message_con = None*)

Bases: *aiogram.types.inline_query_result.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

### 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[aiogram.types.inline_keyboard.Inline = None, input_message_content: Optional[aiogram.types.input_message_conten = None*)

Bases: *aiogram.types.inline_query_result.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

### 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[aiogram.types.inline_keyboar = None, input_message_content: Optional[aiogram.types.input_message = None*)

Bases: *aiogram.types.inline_query_result.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

### 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[aiogram.types.inline_keyboard.Inl = None, input_message_content: Optional[aiogram.types.input_message_con = None*)

Bases: *aiogram.types.inline_query_result.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

### 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[aiogram.types.inline_keyboard.Inl*
*= None, input_message_content: Optional[aiogram.types.input_message_con*
*= None*)

Bases: *aiogram.types.inline_query_result.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

### 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[aiogram.types.inline_keyboard.Inl = None, input_message_content: Optional[aiogram.types.input_message_con = None*)

Bases: `aiogram.types.inline_query_result.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

### InlineQueryResultDocument

**class** aiogram.types.inline_query_result.**InlineQueryResultDocument**(**, id: String, title: String, caption: Optional[String] = None, parse_mode: Optional[String] = None, document_url: Optional[String] = None, mime_type: Optional[String] = None, description: Optional[String] = None, reply_markup: Optional[aiogram.types.inline_keyboar = None, input_message_content: Optional[aiogram.types.input_message = None, thumb_url: Optional[String] = None, thumb_width: Optional[Integer] = None, thumb_height: Optional[Integer] = None*)

Bases: *aiogram.types.inline_query_result.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

### InlineQueryResultLocation

**class** aiogram.types.inline_query_result.**InlineQueryResultLocation**(**, id: String, latitude: Float, longitude: Float, title: String, live_period: Optional[Integer] = None, reply_markup: Optional[aiogram.types.inline_keyboar = None, input_message_content: Optional[aiogram.types.input_message = None, thumb_url: Optional[String] = None, thumb_width: Optional[Integer] = None, thumb_height: Optional[Integer] = None*)

Bases: *aiogram.types.inline_query_result.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.

Note: This will only work in Telegram versions released after 9 April, 2016. Older clients will ignore them.

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

### InlineQueryResultVenue

**class** aiogram.types.inline_query_result.**InlineQueryResultVenue**(**, id: String, latitude: Float, longitude: Float, title: String, address: String, foursquare_id: Optional[String] = None, reply_markup: Optional[aiogram.types.inline_keyboard.Inl = None, input_message_content: Optional[aiogram.types.input_message_con = None, thumb_url: Optional[String] = None, thumb_width: Optional[Integer] = None, thumb_height: Optional[Integer] = None, foursquare_type: Optional[String] = None*)

Bases: *aiogram.types.inline_query_result.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

### InlineQueryResultContact

**class** aiogram.types.inline_query_result.**InlineQueryResultContact**(**, id: String, phone_number: String, first_name: String, last_name: Optional[String] = None, reply_markup: Optional[aiogram.types.inline_keyboard.*
*= None, input_message_content: Optional[aiogram.types.input_message_*
*= None, thumb_url: Optional[String] = None, thumb_width: Optional[Integer] = None, thumb_height: Optional[Integer] = None, foursquare_type: Optional[String] = None*)

Bases: *aiogram.types.inline_query_result.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

### InlineQueryResultGame

**class** aiogram.types.inline_query_result.**InlineQueryResultGame**(**, id: String, game_short_name: String, reply_markup: Optional[aiogram.types.inline_keyboard.Inlin*
*= None*)

Bases: *aiogram.types.inline_query_result.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

### 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*, *reply_markup: Optional[aiogram.types.inline_key* = *None*, *input_message_content: Optional[aiogram.types.input_mes* = *None*)

Bases: *aiogram.types.inline_query_result.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

### 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*, *reply_markup: Optional[aiogram.types.inline_keybo* *= None*, *input_message_content: Optional[aiogram.types.input_messag* *= None*)

Bases: `aiogram.types.inline_query_result.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

### 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*, *reply_markup: Optional[aiogram.types.inline_* = None*, *input_message_content: Optional[aiogram.types.input_* = None*)

Bases: *aiogram.types.inline_query_result.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

### InlineQueryResultCachedSticker

**class** aiogram.types.inline_query_result.**InlineQueryResultCachedSticker**(*, *id: String*, *sticker_file_id: String*, *reply_markup: Optional[aiogram.types.inline_k* = *None*, *input_message_content: Optional[aiogram.types.input_m* = *None*)

Bases: *aiogram.types.inline_query_result.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

### 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*, *reply_markup: Optional[aiogram.types.inline* *= None*, *input_message_content: Optional[aiogram.types.input* *= None*)

Bases: *aiogram.types.inline_query_result.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

### 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, reply_markup: Optional[aiogram.types.inline_key = None, input_message_content: Optional[aiogram.types.input_mes = None*)

Bases: `aiogram.types.inline_query_result.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

### 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*, *reply_markup: Optional[aiogram.types.inline_key* *= None*, *input_message_content: Optional[aiogram.types.input_mes* *= None*)

Bases: *aiogram.types.inline_query_result.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

### InlineQueryResultCachedAudio

**class** aiogram.types.inline_query_result.**InlineQueryResultCachedAudio**(**, id: String, audio_file_id: String, caption: Optional[String] = None, parse_mode: Optional[String] = None, reply_markup: Optional[aiogram.types.inline_key = None, input_message_content: Optional[aiogram.types.input_mes = None*)

Bases: *aiogram.types.inline_query_result.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

### InputFile

**class** aiogram.types.input_file.**InputFile**(*path_or_bytesio, filename=None, conf=None*)

Bases: *aiogram.types.base.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`** –

**Returns**

## PreCheckoutQuery

**`class`** `aiogram.types.pre_checkout_query.`**`PreCheckoutQuery`**`(`*conf=None*, ***kwargs*`)`

Bases: *`aiogram.types.base.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=None*, ***kwargs*)

Bases: *aiogram.types.base.TelegramObject*, *aiogram.types.mixins.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=None*, ***kwargs*)

Bases: *aiogram.types.base.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*)

Bases: *aiogram.types.input_message_content.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

### InputLocationMessageContent

**class** aiogram.types.input_message_content.**InputLocationMessageContent**(*latitude: Float*, *longitude: Float*)

Bases: *aiogram.types.input_message_content.InputMessageContent*

Represents the content of a location 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#inputlocationmessagecontent

### InputTextMessageContent

**class** aiogram.types.input_message_content.**InputTextMessageContent**(*message_text: Optional[String] = None*, *parse_mode: Optional[String] = None*, *disable_web_page_preview: Optional[Boolean] = None*)

Bases: *aiogram.types.input_message_content.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

### InputVenueMessageContent

**class** `aiogram.types.input_message_content.`**`InputVenueMessageContent`**(*latitude: Optional[Float] = None, longitude: Optional[Float] = None, title: Optional[String] = None, address: Optional[String] = None, foursquare_id: Optional[String] = None*)

Bases: `aiogram.types.input_message_content.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

### Update

**class** `aiogram.types.update.`**`Update`**(*conf=None, **kwargs*)

Bases: `aiogram.types.base.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: `aiogram.utils.helper.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=None*, ***kwargs*)

Bases: *aiogram.types.base.TelegramObject*, *aiogram.types.mixins.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=None*, ***kwargs*)

Bases: *aiogram.types.base.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=None*, ***kwargs*)

Bases: *aiogram.types.base.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=None*, ***kwargs*)

Bases: *aiogram.types.base.TelegramObject*, *aiogram.types.mixins.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=None*, ***kwargs*)

Bases: *aiogram.types.base.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=None*, ***kwargs*)

Bases: *aiogram.types.base.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=None*, ***kwargs*)

Bases: *aiogram.types.base.TelegramObject*

This object contains information about one member of a chat.

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

Deserialize object

**Parameters**

- **conf** –
- **kwargs** –

### ChatMemberStatus

**class** aiogram.types.chat_member.**ChatMemberStatus**

Bases: aiogram.utils.helper.Helper

Chat member status

### ShippingOption

**class** aiogram.types.shipping_option.**ShippingOption**(*id: String*, *title: String*, *prices: List[aiogram.types.labeled_price.LabeledPrice] = None*)

Bases: *aiogram.types.base.TelegramObject*

This object represents one shipping option.

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

**add**(*price: aiogram.types.labeled_price.LabeledPrice*)

Add price

**Parameters** **price** –

**Returns**

### ChatPhoto

**class** aiogram.types.chat_photo.**ChatPhoto**(*conf=None*, ***kwargs*)

Bases: *aiogram.types.base.TelegramObject*

This object represents a chat photo.

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

Deserialize object

**Parameters**

- **conf** –
- **kwargs** –

**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

**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

### Contact

**class** aiogram.types.contact.**Contact**(*conf=None*, ***kwargs*)
Bases: *aiogram.types.base.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=None*, ***kwargs*)
Bases: *aiogram.types.base.TelegramObject*

This object represents a message.

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

Deserialize object

**Parameters**

- **conf** –
- **kwargs** –

**is_command**()
Check message text is command

**Returns** bool

**get_full_command**()
Split command and args

**Returns** tuple of (command, args)

**get_command**(*pure=False*)
Get command from message

**Returns**

**get_args**()
Get arguments

**Returns**

**parse_entities**(*as_html=True*)
Text or caption formatted as HTML or Markdown.

**Returns** str

**md_text**
Text or caption formatted as markdown.

**Returns** str

**html_text**
Text or caption formatted as HTML

**Returns** str

**delete**()
Delete this message

**Returns** bool

**edit_live_location**(*latitude: Float*, *longitude: Float*, *reply_markup=None*) → Union[aiogram.types.message.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.Union[types.InlineKeyboardMarkup, None]`) – 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]`

**edit_text**(*text: String*, *parse_mode: Optional[String] = None*, *disable_web_page_preview: Optional[Boolean] = None*, *reply_markup=None*)
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.Union[base.String, None]`) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.
- **disable_web_page_preview** (`typing.Union[base.Boolean, None]`) – Disables link previews for links in this message

- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, None]`) – 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]`

**forward** (*chat_id*, *disable_notification=None*) → aiogram.types.message.Message
Forward this message

**Parameters**

- **chat_id** –
- **disable_notification** –

**Returns**

**pin** (*disable_notification: bool = False*)
Pin message

**Parameters** **disable_notification** –

**Returns**

**reply** (*text*, *parse_mode=None*, *disable_web_page_preview=None*, *disable_notification=None*, *reply_markup=None*, *reply=True*) → aiogram.types.message.Message
Reply to this message

**Parameters**

- **text** – str
- **parse_mode** – str
- **disable_web_page_preview** – bool
- **disable_notification** – bool
- **reply_markup** –
- **reply** – fill 'reply_to_message_id'

**Returns** `aiogram.types.Message`

**reply_audio** (*audio: Union[InputFile, String], caption: Optional[String] = None, duration: Optional[Integer] = None, performer: Optional[String] = None, title: Optional[String] = None, disable_notification: Optional[Boolean] = None, reply_markup=None, reply=True*) → aiogram.types.message.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.Union[base.String, None]`) – Audio caption, 0-200 characters
- **duration** (`typing.Union[base.Integer, None]`) – Duration of the audio in seconds

- **performer** (typing.Union[base.String, None]) – Performer
- **title** (typing.Union[base.String, None]) – Track name
- **disable_notification** (typing.Union[base.Boolean, None]) – Sends the message silently. Users will receive a notification with no sound.
- **reply_markup** (typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]) – Additional interface options.
- **reply** – fill 'reply_to_message_id'

**Returns** On success, the sent Message is returned.

**Return type** types.Message

**reply_document** (*document: Union[InputFile, String], caption: Optional[String] = None, disable_notification: Optional[Boolean] = None, reply_markup=None, reply=True*) → aiogram.types.message.Message

Use this method to send general files.

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.
- **caption** (typing.Union[base.String, None]) – Document caption (may also be used when resending documents by file_id), 0-200 characters
- **disable_notification** (typing.Union[base.Boolean, None]) – Sends the message silently. Users will receive a notification with no sound.
- **reply_markup** (typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply], None]) – Additional interface options.
- **reply** – fill 'reply_to_message_id'

**Returns** On success, the sent Message is returned.

**Return type** types.Message

**reply_location** (*latitude: Float, longitude: Float, live_period: Optional[Integer] = None, disable_notification: Optional[Boolean] = None, reply_markup=None, reply=True*) → aiogram.types.message.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
- **live_period** (typing.Union[base.Integer, None]) – Period in seconds for which the location will be updated
- **disable_notification** (typing.Union[base.Boolean, None]) – Sends the message silently. Users will receive a notification with no sound.

- **reply_markup** (typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]) – Additional interface options.
- **reply** – fill 'reply_to_message_id'

**Returns** On success, the sent Message is returned.

**Return type** types.Message

**reply_media_group**(*media: Union[aiogram.types.input_media.MediaGroup, List[T]], disable_notification: Optional[Boolean] = None, reply=True*) → List[aiogram.types.message.Message]

Use this method to send a group of photos or videos as an album.

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.Union[base.Boolean, None]) – Sends the message silently. Users will receive a notification with no sound.
- **reply** – fill 'reply_to_message_id'

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

**Return type** typing.List[types.Message]

**reply_photo**(*photo: Union[InputFile, String], caption: Optional[String] = None, disable_notification: Optional[Boolean] = None, reply_markup=None, reply=True*) → aiogram.types.message.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.Union[base.String, None]) – Photo caption (may also be used when resending photos by file_id), 0-200 characters
- **disable_notification** (typing.Union[base.Boolean, None]) – Sends the message silently. Users will receive a notification with no sound.
- **reply_markup** (typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]) – Additional interface options.
- **reply** – fill 'reply_to_message_id'

**Returns** On success, the sent Message is returned.

**Return type** types.Message

**reply_sticker**(*sticker: Union[InputFile, String], disable_notification: Optional[Boolean] = None, reply_markup=None, reply=True*) → aiogram.types.message.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.Union[base.Boolean, None]`) – Sends the message silently. Users will receive a notification with no sound.
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]`) – Additional interface options.
- **reply** – fill 'reply_to_message_id'

**Returns** On success, the sent Message is returned.

**Return type** `types.Message`

**reply_video**(*video: Union[InputFile, String], duration: Optional[Integer] = None, width: Optional[Integer] = None, height: Optional[Integer] = None, caption: Optional[String] = None, disable_notification: Optional[Boolean] = None, reply_markup=None, reply=True*) → aiogram.types.message.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.Union[base.Integer, None]`) – Duration of sent video in seconds
- **width** (`typing.Union[base.Integer, None]`) – Video width
- **height** (`typing.Union[base.Integer, None]`) – Video height
- **caption** (`typing.Union[base.String, None]`) – Video caption (may also be used when resending videos by file_id), 0-200 characters
- **disable_notification** (`typing.Union[base.Boolean, None]`) – Sends the message silently. Users will receive a notification with no sound.
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]`) – Additional interface options.
- **reply** – fill 'reply_to_message_id'

**Returns** On success, the sent Message is returned.

**Return type** `types.Message`

**reply_video_note**(*video_note: Union[InputFile, String], duration: Optional[Integer] = None, length: Optional[Integer] = None, disable_notification: Optional[Boolean] = None, reply_markup=None, reply=True*) → aiogram.types.message.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.Union[base.Integer, None]`) – Duration of sent video in seconds
- **length** (`typing.Union[base.Integer, None]`) – Video width and height
- **disable_notification** (`typing.Union[base.Boolean, None]`) – Sends the message silently. Users will receive a notification with no sound.
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]`) – Additional interface options.
- **reply** – fill 'reply_to_message_id'

**Returns** On success, the sent Message is returned.

**Return type** `types.Message`

**reply_voice**(*voice: Union[InputFile, String], caption: Optional[String] = None, duration: Optional[Integer] = None, disable_notification: Optional[Boolean] = None, reply_markup=None, reply=True*) → aiogram.types.message.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.Union[base.String, None]`) – Voice message caption, 0-200 characters
- **duration** (`typing.Union[base.Integer, None]`) – Duration of the voice message in seconds
- **disable_notification** (`typing.Union[base.Boolean, None]`) – Sends the message silently. Users will receive a notification with no sound.
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]`) – Additional interface options.
- **reply** – fill 'reply_to_message_id'

**Returns** On success, the sent Message is returned.

**Return type** `types.Message`

**send_animation**(*animation: Union[InputFile, String], duration: Optional[Integer] = None, width: Optional[Integer] = None, height: Optional[Integer] = None, thumb: Union[InputFile, String, None] = None, caption: Optional[String] = None, parse_mode: Optional[String] = None, disable_notification: Optional[Boolean] = None, reply_markup=None, reply=True*) → aiogram.types.message.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.Union[base.Integer, None]`) – Duration of sent animation in seconds
- **width** (`typing.Union[base.Integer, None]`) – Animation width
- **height** (`typing.Union[base.Integer, None]`) – 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 90.
- **caption** (`typing.Union[base.String, None]`) – Animation caption (may also be used when resending animation by file_id), 0-1024 characters
- **parse_mode** (`typing.Union[base.String, None]`) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption
- **disable_notification** (`typing.Union[base.Boolean, None]`) – Sends the message silently. Users will receive a notification with no sound
- **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** – fill 'reply_to_message_id'

**Returns** On success, the sent Message is returned

**Return type** `types.Message`

**send_contact** (*phone_number: String*, *first_name: String*, *last_name: Optional[String] = None*, *disable_notification: Optional[Boolean] = None*, *reply_markup=None*, *reply=True*) → aiogram.types.message.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.Union[base.String, None]`) – Contact's last name
- **disable_notification** (`typing.Union[base.Boolean, None]`) – Sends the message silently. Users will receive a notification with no sound.
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]`) – Additional interface options.
- **reply** – fill 'reply_to_message_id'

**Returns** On success, the sent Message is returned.

**Return type** `types.Message`

**send_venue**(*latitude: Float*, *longitude: Float*, *title: String*, *address: String*, *foursquare_id: Optional[String] = None*, *disable_notification: Optional[Boolean] = None*, *reply_markup=None*, *reply=True*) → aiogram.types.message.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.Union[base.String, None]`) – Foursquare identifier of the venue
- **disable_notification** (`typing.Union[base.Boolean, None]`) – Sends the message silently. Users will receive a notification with no sound.
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]`) – Additional interface options.
- **reply** – fill 'reply_to_message_id'

**Returns** On success, the sent Message is returned.

**Return type** `types.Message`

**stop_live_location**(*reply_markup=None*) → Union[aiogram.types.message.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.Union[types.InlineKeyboardMarkup, None]`) – 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]`

## ContentType

**class** `aiogram.types.message.`**ContentType**

Bases: `aiogram.utils.helper.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** 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: `aiogram.utils.helper.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** 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: `aiogram.utils.helper.Helper`

Parse modes

**Key** MARKDOWN

**Key** HTML

### MaskPosition

**class** `aiogram.types.mask_position.`**`MaskPosition`**(*conf=None*, ***kwargs*)

Bases: *`aiogram.types.base.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=None*, ***kwargs*)

Bases: *`aiogram.types.base.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=None*, ***kwargs*)

Bases: *aiogram.types.base.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=None*, ***kwargs*)

Bases: *aiogram.types.base.TelegramObject*

Deserialize object

**Parameters**

- **conf** –
- **kwargs** –

**classmethod parse**(*request: aiohttp.web_request.Request*) → aiogram.types.auth_widget_data.AuthWidgetData

Parse request as Telegram auth widget data.

**Parameters request** –

**Returns** *AuthWidgetData*

**Raise** aiohttp.web.HTTPBadRequest

## 4.5 Dispatcher

### 4.5.1 Filters

#### Basics

Coming soon…

#### Filters factory

Coming soon…

#### Builtin filters

Coming soon…

#### Making own filters (Custom filters)

Coming soon. . .

### 4.5.2 Finite state machine

#### Storage

Coming soon. . .

#### Available storage's

Coming soon. . .

#### Memory storage

Coming soon. . .

#### Redis storage

Coming soon. . .

#### Rethink DB storage

Coming soon. . .

#### 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[aiogram.dispatcher.storage.BaseStorage] = None*, *run_tasks_by_default: bool = False*, *throttling_rate_limit=0.1*, *no_throttle_error=False*, *filters_factory=None*)

Bases: `aiogram.utils.mixins.DataMixin`, `aiogram.utils.mixins.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.

**stop_polling**()

Break long-polling process.

**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(rexexp='^[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_query: 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_query: True)
async def some_chosen_inline_handler(chosen_inline_query: 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_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: Union[str, int, None] = None*, *user: Union[str, int, None] = None*) → aiogram.dispatcher.storage.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**

**check_key**(*key*, *chat=None*, *user=None*)
Get information about key in bucket

**Parameters**

- **key** –
- **chat** –
- **user** –

**Returns**

**process_update**(*update: aiogram.types.update.Update*)
Process single update object

**Parameters** **update** –

**Returns**

**process_updates**(*updates*, *fast: Optional[bool] = True*)
Process list of updates

**Parameters**

- **updates** –
- **fast** –

**Returns**

**release_key**(*key*, *chat=None*, *user=None*)
Release blocked key

**Parameters**

- **key** –
- **chat** –
- **user** –

**Returns**

**reset_webhook**(*check=True*) → bool
Reset webhook

**Parameters** **check** – check before deleting

**Returns**

**skip_updates**()
You can skip old incoming updates from queue. This method is not recommended to use if you use payments or you bot has high-load.

**Returns** None

**start_polling**(*timeout=20*, *relax=0.1*, *limit=None*, *reset_webhook=None*, *fast: Optional[bool] = True*)
Start long-polling

**Parameters**

- **timeout** –
- **relax** –
- **limit** –
- **reset_webhook** –
- **fast** –

**Returns**

**throttle**(*key*, *, *rate=None*, *user=None*, *chat=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** – user id
- **chat** – chat id
- **no_error** – return boolean value instead of raising error

**Returns** bool

**wait_closed**()
Wait for the long-polling to close

**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**

## 4.6 Utils

### 4.6.1 Executor

Coming soon…

### 4.6.2 Exceptions

Coming soon…

### 4.6.3 Context

Coming soon…

### 4.6.4 Markdown

Coming soon…

### 4.6.5 Helper

Coming soon. . .

### 4.6.6 Auth Widget

Coming soon. . .

### 4.6.7 Payload

Coming soon. . .

### 4.6.8 Parts

Coming soon. . .

### 4.6.9 JSON

Coming soon. . .

### 4.6.10 Emoji

Coming soon. . .

### 4.6.11 Deprecated

Coming soon. . .

## 4.7 Examples

### 4.7.1 Echo bot

Very simple example of the bot which will sent text of the received messages to the sender

Listing 1: echo_bot.py

```python
"""
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'

```

(continues on next page)

(continued from previous page)

```python
# 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 client send `/start` or `/help` commands.
    """
    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:
        await bot.send_photo(message.chat.id, photo, caption='Cats is here ',
                             reply_to_message_id=message.message_id)


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


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

### 4.7.2 Inline bot

Listing 2: inline_bot.py

```python
import asyncio
import logging

from aiogram import Bot, types, Dispatcher, executor

API_TOKEN = 'BOT TOKEN HERE'

logging.basicConfig(level=logging.DEBUG)

loop = asyncio.get_event_loop()
bot = Bot(token=API_TOKEN, loop=loop)
dp = Dispatcher(bot)


@dp.inline_handler()
async def inline_echo(inline_query: types.InlineQuery):
    input_content = types.InputTextMessageContent(inline_query.query or 'echo')
    item = types.InlineQueryResultArticle(id='1', title='echo',
                                          input_message_content=input_content)
    await bot.answer_inline_query(inline_query.id, results=[item], cache_time=1)
```

(continues on next page)

(continued from previous page)

```python


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

### 4.7.3 Adwanced executor example

!/usr/bin/env python3 **This example is outdated** In this example used ArgumentParser for configuring Your bot. Provided to start bot with webhook: python adwanced_executor_example.py –token TOKEN_HERE –host 0.0.0.0 –port 8084 –host-name example.com –webhook-port 443 Or long polling: python adwanced_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.

Listing 3: adwanced_executor_example.py

```python
import argparse
import logging
import ssl
import sys

from aiogram import Bot
from aiogram.dispatcher import Dispatcher
from aiogram.dispatcher.webhook import *
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):
```

(continues on next page)

(continued from previous page)

```
    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()


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:
```

(continues on next page)

(continued from previous page)

```
            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)

    main(argv)
```

### 4.7.4 Proxy and emojize

Listing 4: proxy_and_emojize.py

```
import asyncio
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://...'

# 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 25, like this:
# >>> bot = Bot(token=API_TOKEN, loop=loop, 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)

loop = asyncio.get_event_loop()
bot = Bot(token=API_TOKEN, loop=loop, proxy=PROXY_URL)
dp = Dispatcher(bot)
```

(continues on next page)

(continued from previous page)

```


async def fetch(url, proxy=None, proxy_auth=None):
    async with aiohttp.ClientSession() as session:
        async with session.get(url, proxy=proxy, proxy_auth=proxy_auth) as response:
            return await response.text()


@dp.message_handler(commands=['start'])
async def cmd_start(message: types.Message):
    content = []

    # Make request (without proxy)
    ip = await fetch(GET_IP_URL)
    content.append(text(':globe_showing_Americas:', bold('IP:'), code(ip)))
    # This line is formatted to ' *IP:* `YOUR IP`'

    # Make request through proxy
    ip = await fetch(GET_IP_URL, bot.proxy, bot.proxy_auth)
    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, loop=loop, skip_updates=True)
```

## 4.7.5 Finite state machine example

Listing 5: finite_state_machine_example.py

```
import asyncio
from typing import Optional

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.state import State, StatesGroup
from aiogram.types import ParseMode
from aiogram.utils import executor
```

(continues on next page)

(continued from previous page)

```

API_TOKEN = 'BOT TOKEN HERE'

loop = asyncio.get_event_loop()

bot = Bot(token=API_TOKEN, loop=loop)

# 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(lambda message: message.text.lower() == 'cancel', state='*')
async def cancel_handler(message: types.Message, state: FSMContext, raw_state:␣
↪Optional[str] = None):
    """
    Allow user to cancel any action
    """
    if raw_state is None:
        return

    # Cancel state and inform user about it
    await state.finish()
    # And remove keyboard (just in case)
    await message.reply('Canceled.', 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()
    await message.reply("How old are you?")
```

(continues on next page)

(continued from previous page)

```python


# Check age. Age gotta be digit
@dp.message_handler(lambda message: not message.text.isdigit(), state=Form.age)
async def failed_process_age(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 failed_process_gender(message: types.Message):
    """
    In this example gender has to be one of: Male, Female, Other.
    """
    return await message.reply("Bad gender name. Choose you gender from 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:', data['age']),
            md.text('Gender:', data['gender']),
            sep='\n'), reply_markup=markup, parse_mode=ParseMode.MARKDOWN)

        # Finish conversation
        data.state = None


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

### 4.7.6 Throtling example

Example for throttling manager. You can use that for flood controlling.

Listing 6: throtling_example.py

```python
import asyncio
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)

loop = asyncio.get_event_loop()
bot = Bot(token=API_TOKEN, loop=loop)

# Throttling manager does not work without Leaky Bucket.
# Then need to use storages. For example use simple in-memory storage.
storage = MemoryStorage()
dp = Dispatcher(bot, storage=storage)


@dp.message_handler(commands=['start', 'help'])
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!')
    else:
        # Otherwise do something
        await message.reply("Hi!\nI'm EchoBot!\nPowered by aiogram.")


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

### 4.7.7 I18n example

Internalize your bot Step 1: extract texts # pybabel extract i18n_example.py -o locales/mybot.pot Step 2: create *.po files. For e.g. create en, ru, uk locales. # echo {en,ru,uk} | xargs -n1 pybabel init -i locales/mybot.pot -d locales -D mybot -l Step 3: translate texts 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 Step 5.4: compile mo files command from step 4

Listing 7: i18n_example.py

```python
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)

# 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):
    await message.reply(_('Your current language: <i>{language}</i>').
↪format(language=locale))


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

### 4.7.8 Regexp commands filter example

Listing 8: regexp_commands_filter_example.py

```python
from aiogram import Bot, types
from aiogram.dispatcher import Dispatcher, filters
from aiogram.utils import executor

bot = Bot(token='TOKEN')
dp = Dispatcher(bot)


@dp.message_handler(filters.RegexpCommandsFilter(regexp_commands=['item_([0-9]*)']))
async def send_welcome(message: types.Message, regexp_command):
```

(continues on next page)

(continued from previous page)

```python
    await message.reply("You have requested an item with number: {}".format(regexp_
↪command.group(1)))


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

### 4.7.9 Check user language

Babel is required.

Listing 9: check_user_language.py

```python
import asyncio
import logging

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

API_TOKEN = 'BOT TOKEN HERE'

logging.basicConfig(level=logging.INFO)

loop = asyncio.get_event_loop()
bot = Bot(token=API_TOKEN, loop=loop, parse_mode=types.ParseMode.MARKDOWN)
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.italic(locale.locale)),
        md.text(' ', md.bold('Territory:'), md.italic(locale.territory or 'Unknown')),
        md.text(' ', md.bold('Language name:'), md.italic(locale.language_name)),
        md.text(' ', md.bold('English language name:'), md.italic(locale.english_
↪name)),
        sep='\n'))


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

### 4.7.10 Middleware and antiflood

Listing 10: middleware_and_antiflood.py

```python
import asyncio

from aiogram import Bot, Dispatcher, executor, types
from aiogram.contrib.fsm_storage.redis import RedisStorage2
from aiogram.dispatcher import DEFAULT_RATE_LIMIT
```

(continues on next page)

(continued from previous page)

```python
from aiogram.dispatcher.handler import CancelHandler, current_handler
from aiogram.dispatcher.middlewares import BaseMiddleware
from aiogram.utils.exceptions import Throttled

TOKEN = 'BOT TOKEN HERE'

loop = asyncio.get_event_loop()

# In this example Redis storage is used
storage = RedisStorage2(db=5)

bot = Bot(token=TOKEN, loop=loop)
dp = Dispatcher(bot, storage=storage)


def rate_limit(limit: int, key=None):
    """
    Decorator for configuring rate limit and key in different functions.

    :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)
```

(continues on next page)

(continued from previous page)

```
            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()

    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.')

```

(continues on next page)

(continued from previous page)

```python

if __name__ == '__main__':
    # Setup middleware
    dp.middleware.setup(ThrottlingMiddleware())

    # Start long-polling
    executor.start_polling(dp, loop=loop)
```

### 4.7.11 Webhook example

Example outdated

Listing 11: webhook_example.py

```python
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
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

loop = asyncio.get_event_loop()
bot = Bot(TOKEN, loop=loop)
storage = MemoryStorage()
```

(continues on next page)

(continued from previous page)

```python
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.
    """
    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}"]
```

(continues on next page)

(continued from previous page)

```
    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
    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):
```

(continues on next page)

(continued from previous page)

```
    """
    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:
    #   If you start your bot using nginx or Apache web server, SSL context is not␣
↪required.
    #   Otherwise you need to set `ssl_context` parameter.
```

### 4.7.12 Webhook example 2

Listing 12: webhook_example_2.py

```
import asyncio
import logging

from aiogram import Bot, types
from aiogram.dispatcher import Dispatcher
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)

loop = asyncio.get_event_loop()
```

(continues on next page)

(continued from previous page)

```python
bot = Bot(token=API_TOKEN, loop=loop)
dp = Dispatcher(bot)


@dp.message_handler()
async def echo(message: types.Message):
    await bot.send_message(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):
    # insert code here to run it before shutdown
    pass


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)
```

### 4.7.13 Payments

Listing 13: payments.py

```python
import asyncio

from aiogram import Bot
from aiogram import types
from aiogram.utils import executor
from aiogram.dispatcher import Dispatcher
from aiogram.types.message import ContentTypes


BOT_TOKEN = 'BOT TOKEN HERE'
PAYMENTS_PROVIDER_TOKEN = '123456789:TEST:1234567890abcdef1234567890abcdef'

loop = asyncio.get_event_loop()
bot = Bot(BOT_TOKEN)
dp = Dispatcher(bot, loop=loop)

# 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))
```

(continues on next page)

(continued from previous page)

```
]


@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 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://images.fineartamerica.com/images-medium-large/2-the-time-machine-dmitriy-khristenko.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')

```

(continues on next page)

(continued from previous page)

```

@dp.shipping_query_handler(func=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(func=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 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, loop=loop)
```

### 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')

loop = asyncio.get_event_loop()
bot = Bot(token=API_TOKEN, loop=loop, parse_mode=types.ParseMode.HTML)
dp = Dispatcher(bot, loop=loop)
```

(continues on next page)

(continued from previous page)

```


def get_users():
    """
    Return users list

    In this example returns some random ID's
    """
    yield from (61043901, 78238238, 78378343, 98765431, 12345678)


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:
```

(continues on next page)

(continued from previous page)

```python
        log.info(f"{count} messages successful sent.")

    return count


if __name__ == '__main__':
    # Execute broadcaster
    executor.start(dp, broadcaster())
```

### 4.7.15 Media group

Listing 15: media_group.py

```python
import asyncio

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

API_TOKEN = 'BOT TOKEN HERE'

loop = asyncio.get_event_loop()
bot = Bot(token=API_TOKEN, loop=loop)
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?")

    # And wait few seconds...
    await asyncio.sleep(1)

    # Good bots should send chat actions. Or not.
    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__':
```

(continues on next page)

(continued from previous page)

```
    executor.start_polling(dp, loop=loop, skip_updates=True)
```

## 4.8 Contribution

TODO

## 4.9 Links

TODO

# CHAPTER 5

# Indices and tables

- genindex
- modindex
- search

# Python Module Index

# Index