
aiogram Documentation

Release 2.15

Illemius / Alex Root Junior

Oct 09, 2021

CONTENTS

1	Official aiogram resources	3
2	Features	5
3	Contribute	7
4	Contents	9
4.1	Installation Guide	9
4.2	Quick start	10
4.3	Migration FAQ (1.4 -> 2.0)	12
4.4	Telegram	17
4.5	Dispatcher	161
4.6	Utils	188
4.7	Examples	203
4.8	Contribution	231
4.9	Links	231
5	Indices and tables	233
	Python Module Index	235
	Index	237

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

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](#)

FEATURES

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

CONTRIBUTE

- [Issue Tracker](#)
- [Source Code](#)

CONTENTS

4.1 Installation Guide

4.1.1 Using PIP

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

4.1.2 Using Pipenv

```
$ pipenv install aiogram
```

4.1.3 Using Pacman

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

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

4.1.4 From sources

Development versions:

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

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

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

4.1.5 Recommendations

You can speedup your bots by following next instructions:

- Use `uvloop` instead of default `asyncio` loop.

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

Installation:

```
$ pip install uvloop
```

- Use `ujson` instead of the default `json` module.

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

Installation:

```
$ pip install ujson
```

- Use `aiohttp` speedups

- Use `cchardet` instead of the `chardet` module.

cChardet is a high speed universal character encoding detector.

Installation:

```
$ pip install cchardet
```

- Use `aiodns` for speeding up DNS resolving.

aiodns provides a simple way for doing asynchronous DNS resolutions.

Installation:

```
$ pip install aiodns
```

- Installing speedups altogether.

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

Installation:

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

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

4.2 Quick start

4.2.1 Simple template

At first you have to import all necessary modules

```
import logging

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

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

```
API_TOKEN = 'BOT TOKEN HERE'

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

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

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

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

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

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

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

Last step: run long polling.

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

4.2.2 Summary

```
1  """
2  This is a echo bot.
3  It echoes any incoming text messages.
4  """
5
6  import logging
7
8  from aiogram import Bot, Dispatcher, executor, types
9
10 API_TOKEN = 'BOT TOKEN HERE'
11
12 # Configure logging
13 logging.basicConfig(level=logging.INFO)
14
15 # Initialize bot and dispatcher
16 bot = Bot(token=API_TOKEN)
```

(continues on next page)

(continued from previous page)

```

17 dp = Dispatcher(bot)
18
19
20 @dp.message_handler(commands=['start', 'help'])
21 async def send_welcome(message: types.Message):
22     """
23     This handler will be called when user sends `/start` or `/help` command
24     """
25     await message.reply("Hi!\nI'm EchoBot!\nPowered by aiogram.")
26
27
28
29 @dp.message_handler()
30 async def echo(message: types.Message):
31     # old style:
32     # await bot.send_message(message.chat.id, message.text)
33
34     await message.answer(message.text)
35
36
37 if __name__ == '__main__':
38     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 *aiosocks* instead of *aiosocksy* for Socks4/5 proxy;
- *types.ContentType* was divided to *types.ContentType* and *types.ContentTypes*;

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

4.3.2 Instructions

Contextvars

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

Example:

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

Filters

Custom filters

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

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

(func filter is still available until v2.1)

Filters factory

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

```
from aiogram.dispatcher.filters import BoundFilter

class MyFilter(BoundFilter):
    key = 'is_admin'

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

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

dp.filters_factory.bind(MyFilter)

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

Customize commands prefix

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

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

Passing data from filters as keyword arguments to the handlers

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

Before (<=v1.4)

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

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

Now (v2.0)

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

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

Other

Filters can also be used as logical expressions:

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

States group

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

Writing states group:

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

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

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

FSM storage's proxy

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

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

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

Usage:

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

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

File uploading mechanism

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

Pipe for uploading files from URL

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

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

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

Example:

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

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

I18n Middleware

You can internalize your bot by following next steps:

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

First usage

1. Extract texts

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

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

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

Updating translations

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

1. Regenerate pot file:

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

2. Update po files

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

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

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

Error handlers

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

Content types

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

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

4.4 Telegram

4.4.1 Bot object

Low level API

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

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

Bases: object

Base class for bot. It's raw bot.

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

Parameters

- **token** (str) – token from @BotFather
- **loop** (Optional Union asyncio.BaseEventLoop, asyncio.AbstractEventLoop) – event loop
- **connections_limit** (int) – connections limit for aiohttp.ClientSession
- **proxy** (str) – HTTP proxy URL
- **proxy_auth** (Optional aiohttp.BasicAuth) – Authentication information
- **validate_token** (bool) – Validate token.

- **parse_mode** (str) – You can set default parse mode
- **timeout** (typing.Optional[typing.Union[base.Integer, base.Float, aiohttp.ClientTimeout]]) – Request timeout
- **server** (TelegramAPIServer) – Telegram Bot API Server endpoint.

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

request_timeout(*timeout: Union[aiogram.types.base.Integer, aiogram.types.base.Float, aiohttp.client.ClientTimeout]*)

Context manager implements opportunity to change request timeout in current context

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

Returns

close()

Close all client sessions

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

Make an request to Telegram Bot API

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

Parameters

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

Returns result

Return type Union[List, Dict]

Raise `aiogram.exceptions.TelegramApiError`

async download_file(*file_path: aiogram.types.base.String, destination: Optional[aiogram.types.base.InputFile] = None, timeout: Optional[aiogram.types.base.Integer] = <object object>, chunk_size: Optional[aiogram.types.base.Integer] = 65536, seek: Optional[aiogram.types.base.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

async send_file(*file_type, method, file, payload*) → Union[Dict, aiogram.types.base.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: aiogram.types.base.String, loop:
    Optional[Union[asyncio.base_events.BaseEventLoop,
        asyncio.events.AbstractEventLoop]] = None, connections_limit:
    Optional[aiogram.types.base.Integer] = None, proxy:
    Optional[aiogram.types.base.String] = None, proxy_auth:
    Optional[aiohttp.helpers.BasicAuth] = None, validate_token:
    Optional[aiogram.types.base.Boolean] = True, parse_mode:
    Optional[aiogram.types.base.String] = None, timeout:
    Optional[Union[aiogram.types.base.Integer, aiogram.types.base.Float,
        aiohttp.client.ClientTimeout]] = None, server:
    aiogram.bot.api.TelegramAPIServer =
    TelegramAPIServer(base='https://api.telegram.org/bot{token}/{method}',
        file='https://api.telegram.org/file/bot{token}/{path}'))

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
- **timeout** (typing.Optional[typing.Union[base.Integer, base.Float, aiohttp.ClientTimeout]]) – Request timeout
- **server** (TelegramAPIServer) – Telegram Bot API Server endpoint.

Raise when token is invalid throw an [aiogram.utils.exceptions.ValidationError](#)

property me: `aiogram.types.user.User`

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

Returns `aiogram.types.User`

async download_file_by_id(*file_id: aiogram.types.base.String, destination=None, timeout: aiogram.types.base.Integer = 30, chunk_size: aiogram.types.base.Integer = 65536, seek: aiogram.types.base.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

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

Returns An Array of Update objects is returned

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

async set_webhook(*url: aiogram.types.base.String, certificate: Optional[aiogram.types.base.InputFile] = None, ip_address: Optional[aiogram.types.base.String] = None, max_connections: Optional[aiogram.types.base.Integer] = None, allowed_updates: Optional[List[aiogram.types.base.String]] = None, drop_pending_updates: Optional[aiogram.types.base.Boolean] = None*) → `aiogram.types.base.Boolean`

Use this method to specify a url and receive incoming updates via an outgoing webhook. Whenever there is an update for the bot, we will send an HTTPS POST request to the specified url, containing a JSON-

serialized Update. In case of an unsuccessful request, we will give up after a reasonable amount of attempts. Returns True on success.

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

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

Parameters

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

Returns Returns true

Return type `base.Boolean`

async delete_webhook(*drop_pending_updates: Optional[aiogram.types.base.Boolean] = None*) → `aiogram.types.base.Boolean`

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

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

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

Returns Returns True on success

Return type `base.Boolean`

async get_webhook_info() → `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`

async `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`

async `log_out()` → `aiogram.types.base.Boolean`

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

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

Returns Returns `True` on success

Return type `base.Boolean`

close_bot() → `aiogram.types.base.Boolean`

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

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

Returns Returns `True` on success

Return type `base.Boolean`

async `send_message(chat_id: Union[aiogram.types.base.Integer, aiogram.types.base.String], text: aiogram.types.base.String, parse_mode: Optional[aiogram.types.base.String] = None, entities: Optional[List[aiogram.types.message_entity.MessageEntity]] = None, disable_web_page_preview: Optional[aiogram.types.base.Boolean] = None, disable_notification: Optional[aiogram.types.base.Boolean] = None, reply_to_message_id: Optional[aiogram.types.base.Integer] = None, allow_sending_without_reply: Optional[aiogram.types.base.Boolean] = None, reply_markup: Optional[Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup, aiogram.types.reply_keyboard.ReplyKeyboardMarkup, aiogram.types.reply_keyboard.ReplyKeyboardRemove, aiogram.types.force_reply.ForceReply]] = 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.Optional[base.String]`) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.

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

Returns On success, the sent Message is returned

Return type `types.Message`

```
async forward_message(chat_id: Union[aiogram.types.base.Integer, aiogram.types.base.String],
                     from_chat_id: Union[aiogram.types.base.Integer, aiogram.types.base.String],
                     message_id: aiogram.types.base.Integer, disable_notification:
                     Optional[aiogram.types.base.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.Optional[base.Boolean]`) – 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`

```
async copy_message(chat_id: Union[aiogram.types.base.Integer, aiogram.types.base.String],
                    from_chat_id: Union[aiogram.types.base.Integer, aiogram.types.base.String],
                    message_id: aiogram.types.base.Integer, caption:
                    Optional[aiogram.types.base.String] = None, parse_mode:
                    Optional[aiogram.types.base.String] = None, caption_entities:
                    Optional[List[aiogram.types.message_entity.MessageEntity]] = None,
                    disable_notification: Optional[aiogram.types.base.Boolean] = None,
                    reply_to_message_id: Optional[aiogram.types.base.Integer] = None,
                    allow_sending_without_reply: Optional[aiogram.types.base.Boolean] = None,
                    reply_markup:
                    Optional[Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup,
                    aiogram.types.reply_keyboard.ReplyKeyboardMarkup,
                    aiogram.types.reply_keyboard.ReplyKeyboardRemove,
                    aiogram.types.force_reply.ForceReply]] = None) →
                    aiogram.types.message_id.MessageId
```

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

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

Parameters

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

Returns On success, the sent `Message` is returned

Return type `types.Message`

```

async send_photo(chat_id: Union[aiogram.types.base.Integer, aiogram.types.base.String], photo:
    Union[aiogram.types.base.InputFile, aiogram.types.base.String], caption:
    Optional[aiogram.types.base.String] = None, parse_mode:
    Optional[aiogram.types.base.String] = None, caption_entities:
    Optional[List[aiogram.types.message_entity.MessageEntity]] = None,
    disable_notification: Optional[aiogram.types.base.Boolean] = None,
    reply_to_message_id: Optional[aiogram.types.base.Integer] = None,
    allow_sending_without_reply: Optional[aiogram.types.base.Boolean] = None,
    reply_markup: Optional[Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup,
    aiogram.types.reply_keyboard.ReplyKeyboardMarkup,
    aiogram.types.reply_keyboard.ReplyKeyboardRemove,
    aiogram.types.force_reply.ForceReply]] = 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.Optional[base.String]) – Photo caption (may also be used when resending photos by file_id), 0-1024 characters
- **parse_mode** (typing.Optional[base.String]) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.
- **caption_entities** (typing.Optional[typing.List[types.MessageEntity]]) – List of special entities that appear in message text, which can be specified instead of parse_mode
- **disable_notification** (typing.Optional[base.Boolean]) – Sends the message silently. Users will receive a notification with no sound
- **reply_to_message_id** (typing.Optional[base.Integer]) – If the message is a reply, ID of the original message
- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user

Returns On success, the sent Message is returned

Return type types.Message

```
async send_audio(chat_id: Union[aiogram.types.base.Integer, aiogram.types.base.String], audio:
    Union[aiogram.types.base.InputFile, aiogram.types.base.String], caption:
    Optional[aiogram.types.base.String] = None, parse_mode:
    Optional[aiogram.types.base.String] = None, caption_entities:
    Optional[List[aiogram.types.message_entity.MessageEntity]] = None, duration:
    Optional[aiogram.types.base.Integer] = None, performer:
    Optional[aiogram.types.base.String] = None, title: Optional[aiogram.types.base.String]
    = None, thumb: Optional[Union[aiogram.types.base.InputFile,
    aiogram.types.base.String]] = None, disable_notification:
    Optional[aiogram.types.base.Boolean] = None, reply_to_message_id:
    Optional[aiogram.types.base.Integer] = None, allow_sending_without_reply:
    Optional[aiogram.types.base.Boolean] = None, reply_markup:
    Optional[Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup,
    aiogram.types.reply_keyboard.ReplyKeyboardMarkup,
    aiogram.types.reply_keyboard.ReplyKeyboardRemove,
    aiogram.types.force_reply.ForceReply]] = 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.Optional[base.String]) – Audio caption, 0-1024 characters
- **parse_mode** (typing.Optional[base.String]) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.
- **caption_entities** (typing.Optional[typing.List[types.MessageEntity]]) – List of special entities that appear in message text, which can be specified instead of `parse_mode`
- **duration** (typing.Optional[base.Integer]) – Duration of the audio in seconds
- **performer** (typing.Optional[base.String]) – Performer
- **title** (typing.Optional[base.String]) – Track name
- **thumb** (typing.Union[base.InputFile, base.String, None]) – Thumbnail of the file sent
- **disable_notification** (typing.Optional[base.Boolean]) – Sends the message silently. Users will receive a notification with no sound
- **reply_to_message_id** (typing.Optional[base.Integer]) – If the message is a reply, ID of the original message
- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]) – Additional interface options. A JSON-serialized object for an inline keyboard,

custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user

Returns On success, the sent Message is returned

Return type types.Message

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

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

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

Parameters

- **chat_id** (typing.Union[base.Integer, base.String]) – Unique identifier for the target chat or username of the target channel
- **document** (typing.Union[base.InputFile, base.String]) – File to send
- **thumb** (typing.Union[base.InputFile, base.String, None]) – Thumbnail of the file sent
- **caption** (typing.Optional[base.String]) – Document caption (may also be used when resending documents by file_id), 0-1024 characters
- **disable_content_type_detection** (typing.Optional[base.Boolean]) – Disables automatic server-side content type detection for files uploaded using multipart/form-data
- **parse_mode** (typing.Optional[base.String]) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.
- **caption_entities** (typing.Optional[typing.List[types.MessageEntity]]) – List of special entities that appear in message text, which can be specified instead of parse_mode
- **disable_notification** (typing.Optional[base.Boolean]) – Sends the message silently. Users will receive a notification with no sound
- **reply_to_message_id** (typing.Optional[base.Integer]) – If the message is a reply, ID of the original message
- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found

- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]`) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user

Returns On success, the sent Message is returned

Return type `types.Message`

```
async send_video(chat_id: Union[aiogram.types.base.Integer, aiogram.types.base.String], video:
    Union[aiogram.types.base.InputFile, aiogram.types.base.String], duration:
    Optional[aiogram.types.base.Integer] = None, width:
    Optional[aiogram.types.base.Integer] = None, height:
    Optional[aiogram.types.base.Integer] = None, thumb:
    Optional[Union[aiogram.types.base.InputFile, aiogram.types.base.String]] = None,
    caption: Optional[aiogram.types.base.String] = None, parse_mode:
    Optional[aiogram.types.base.String] = None, caption_entities:
    Optional[List[aiogram.types.message_entity.MessageEntity]] = None,
    supports_streaming: Optional[aiogram.types.base.Boolean] = None,
    disable_notification: Optional[aiogram.types.base.Boolean] = None,
    reply_to_message_id: Optional[aiogram.types.base.Integer] = None,
    allow_sending_without_reply: Optional[aiogram.types.base.Boolean] = None,
    reply_markup: Optional[Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup,
    aiogram.types.reply_keyboard.ReplyKeyboardMarkup,
    aiogram.types.reply_keyboard.ReplyKeyboardRemove,
    aiogram.types.force_reply.ForceReply]] = 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.Optional[base.Integer]`) – Duration of sent video in seconds
- **width** (`typing.Optional[base.Integer]`) – Video width
- **height** (`typing.Optional[base.Integer]`) – Video height
- **thumb** (`typing.Union[base.InputFile, base.String, None]`) – Thumbnail of the file sent
- **caption** (`typing.Optional[base.String]`) – Video caption (may also be used when resending videos by file_id), 0-1024 characters
- **parse_mode** (`typing.Optional[base.String]`) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.
- **caption_entities** (`typing.Optional[typing.List[types.MessageEntity]]`) – List of special entities that appear in message text, which can be specified instead of parse_mode
- **supports_streaming** (`typing.Optional[base.Boolean]`) – Pass True, if the uploaded video is suitable for streaming

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

Returns On success, the sent Message is returned

Return type `types.Message`

```
async send_animation(chat_id: Union[aiogram.types.base.Integer, aiogram.types.base.String], animation:
    Union[aiogram.types.base.InputFile, aiogram.types.base.String], duration:
    Optional[aiogram.types.base.Integer] = None, width:
    Optional[aiogram.types.base.Integer] = None, height:
    Optional[aiogram.types.base.Integer] = None, thumb:
    Optional[Union[aiogram.types.base.InputFile, aiogram.types.base.String]] =
    None, caption: Optional[aiogram.types.base.String] = None, parse_mode:
    Optional[aiogram.types.base.String] = None, caption_entities:
    Optional[List[aiogram.types.message_entity.MessageEntity]] = None,
    disable_notification: Optional[aiogram.types.base.Boolean] = None,
    reply_to_message_id: Optional[aiogram.types.base.Integer] = None,
    allow_sending_without_reply: Optional[aiogram.types.base.Boolean] = None,
    reply_markup:
    Optional[Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup,
    aiogram.types.reply_keyboard.ReplyKeyboardMarkup,
    aiogram.types.reply_keyboard.ReplyKeyboardRemove,
    aiogram.types.force_reply.ForceReply]] = 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.Optional[base.Integer]`) – Duration of sent animation in seconds
- **width** (`typing.Optional[base.Integer]`) – Animation width
- **height** (`typing.Optional[base.Integer]`) – Animation height

- **thumb** (`typing.Union[typing.Union[base.InputFile, base.String], None]`) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320.
- **caption** (`typing.Optional[base.String]`) – Animation caption (may also be used when resending animation by `file_id`), 0-1024 characters
- **parse_mode** (`typing.Optional[base.String]`) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption
- **caption_entities** (`typing.Optional[typing.List[types.MessageEntity]]`) – List of special entities that appear in message text, which can be specified instead of `parse_mode`
- **disable_notification** (`typing.Optional[base.Boolean]`) – Sends the message silently. Users will receive a notification with no sound
- **reply_to_message_id** (`typing.Optional[base.Integer]`) – If the message is a reply, ID of the original message
- **allow_sending_without_reply** (`typing.Optional[base.Boolean]`) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (`typing.Union[typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply], None]`) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user

Returns On success, the sent Message is returned

Return type `types.Message`

```
async send_voice(chat_id: Union[aiogram.types.base.Integer, aiogram.types.base.String], voice:
    Union[aiogram.types.base.InputFile, aiogram.types.base.String], caption:
    Optional[aiogram.types.base.String] = None, parse_mode:
    Optional[aiogram.types.base.String] = None, caption_entities:
    Optional[List[aiogram.types.message_entity.MessageEntity]] = None, duration:
    Optional[aiogram.types.base.Integer] = None, disable_notification:
    Optional[aiogram.types.base.Boolean] = None, reply_to_message_id:
    Optional[aiogram.types.base.Integer] = None, allow_sending_without_reply:
    Optional[aiogram.types.base.Boolean] = None, reply_markup:
    Optional[Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup,
aiogram.types.reply_keyboard.ReplyKeyboardMarkup,
aiogram.types.reply_keyboard.ReplyKeyboardRemove,
aiogram.types.force_reply.ForceReply]] = 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.Optional[base.String]`) – Voice message caption, 0-1024 characters
- **parse_mode** (`typing.Optional[base.String]`) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.
- **caption_entities** (`typing.Optional[typing.List[types.MessageEntity]]`) – List of special entities that appear in message text, which can be specified instead of `parse_mode`
- **duration** (`typing.Optional[base.Integer]`) – Duration of the voice message in seconds
- **disable_notification** (`typing.Optional[base.Boolean]`) – Sends the message silently. Users will receive a notification with no sound
- **reply_to_message_id** (`typing.Optional[base.Integer]`) – If the message is a reply, ID of the original message
- **allow_sending_without_reply** (`typing.Optional[base.Boolean]`) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]`) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user

Returns On success, the sent Message is returned

Return type `types.Message`

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

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

Returns On success, the sent Message is returned

Return type types.Message

```
async def send_media_group(chat_id: Union[aiogram.types.base.Integer, aiogram.types.base.String], media:
    Union[aiogram.types.input_media.MediaGroup, List], disable_notification:
    Optional[aiogram.types.base.Boolean] = None, reply_to_message_id:
    Optional[aiogram.types.base.Integer] = None, allow_sending_without_reply:
    Optional[aiogram.types.base.Boolean] = None) →
    List[aiogram.types.message.Message]
```

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

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

Parameters

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

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

Return type typing.List[types.Message]

```

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

Returns On success, the sent Message is returned

Return type types.Message

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

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

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

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

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

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

async send_venue(*chat_id: Union[aiogram.types.base.Integer, aiogram.types.base.String], latitude: aiogram.types.base.Float, longitude: aiogram.types.base.Float, title: aiogram.types.base.String, address: aiogram.types.base.String, foursquare_id: Optional[aiogram.types.base.String] = None, foursquare_type: Optional[aiogram.types.base.String] = None, google_place_id: Optional[aiogram.types.base.String] = None, google_place_type: Optional[aiogram.types.base.String] = None, disable_notification: Optional[aiogram.types.base.Boolean] = None, reply_to_message_id: Optional[aiogram.types.base.Integer] = None, allow_sending_without_reply: Optional[aiogram.types.base.Boolean] = None, reply_markup: Optional[Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup, aiogram.types.reply_keyboard.ReplyKeyboardMarkup, aiogram.types.reply_keyboard.ReplyKeyboardRemove, aiogram.types.force_reply.ForceReply]] = 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 (in the format `@channelusername`)
- **latitude** (`base.Float`) – Latitude of the venue
- **longitude** (`base.Float`) – Longitude of the venue
- **title** (`base.String`) – Name of the venue
- **address** (`base.String`) – Address of the venue
- **foursquare_id** (`typing.Optional[base.String]`) – Foursquare identifier of the venue
- **foursquare_type** (`typing.Optional[base.String]`) – Foursquare type of the venue, if known
- **google_place_id** (`typing.Optional[base.String]`) – Google Places identifier of the venue
- **google_place_type** (`typing.Optional[base.String]`) – Google Places type of the venue. See supported types: https://developers.google.com/places/web-service/supported_types
- **disable_notification** (`typing.Optional[base.Boolean]`) – Sends the message silently. Users will receive a notification with no sound

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

Returns On success, the sent Message is returned

Return type `types.Message`

```
async send_contact(chat_id: Union[aiogram.types.base.Integer, aiogram.types.base.String],
                  phone_number: aiogram.types.base.String, first_name: aiogram.types.base.String,
                  last_name: Optional[aiogram.types.base.String] = None, vcard:
                  Optional[aiogram.types.base.String] = None, disable_notification:
                  Optional[aiogram.types.base.Boolean] = None, reply_to_message_id:
                  Optional[aiogram.types.base.Integer] = None, allow_sending_without_reply:
                  Optional[aiogram.types.base.Boolean] = None, reply_markup:
                  Optional[Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup,
                  aiogram.types.reply_keyboard.ReplyKeyboardMarkup,
                  aiogram.types.reply_keyboard.ReplyKeyboardRemove,
                  aiogram.types.force_reply.ForceReply]] = 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.Optional[base.String]`) – Contact's last name
- **vcard** (`typing.Optional[base.String]`) – vcard
- **disable_notification** (`typing.Optional[base.Boolean]`) – Sends the message silently. Users will receive a notification with no sound
- **reply_to_message_id** (`typing.Optional[base.Integer]`) – If the message is a reply, ID of the original message
- **allow_sending_without_reply** (`typing.Optional[base.Boolean]`) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]`) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user

Returns On success, the sent Message is returned

Return type `types.Message`


```

async send_poll(chat_id: Union[aiogram.types.base.Integer, aiogram.types.base.String], question:
    aiogram.types.base.String, options: List[aiogram.types.base.String], is_anonymous:
    Optional[aiogram.types.base.Boolean] = None, type:
    Optional[aiogram.types.base.String] = None, allows_multiple_answers:
    Optional[aiogram.types.base.Boolean] = None, correct_option_id:
    Optional[aiogram.types.base.Integer] = None, explanation:
    Optional[aiogram.types.base.String] = None, explanation_parse_mode:
    Optional[aiogram.types.base.String] = None, explanation_entities:
    Optional[List[aiogram.types.message_entity.MessageEntity]] = None, open_period:
    Optional[aiogram.types.base.Integer] = None, close_date:
    Optional[Union[aiogram.types.base.Integer, datetime.datetime, datetime.timedelta]] =
    None, is_closed: Optional[aiogram.types.base.Boolean] = None, disable_notification:
    Optional[aiogram.types.base.Boolean] = None, reply_to_message_id:
    Optional[aiogram.types.base.Integer] = None, allow_sending_without_reply:
    Optional[aiogram.types.base.Boolean] = None, reply_markup:
    Optional[Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup,
    aiogram.types.reply_keyboard.ReplyKeyboardMarkup,
    aiogram.types.reply_keyboard.ReplyKeyboardRemove,
    aiogram.types.force_reply.ForceReply]] = None) → aiogram.types.message.Message

```

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

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

Parameters

- **chat_id** (typing.Union[base.Integer, base.String]) – Unique identifier for the target chat or username of the target channel (in the format @channelusername)
- **question** (base.String) – Poll question, 1-300 characters
- **options** (typing.List[base.String]) – A list of answer options, 2-10 strings 1-100 characters each
- **is_anonymous** (typing.Optional[base.Boolean]) – True, if the poll needs to be anonymous, defaults to True
- **type** (typing.Optional[base.String]) – Poll type, “quiz” or “regular”, defaults to “regular”
- **allows_multiple_answers** (typing.Optional[base.Boolean]) – True, if the poll allows multiple answers, ignored for polls in quiz mode, defaults to False
- **correct_option_id** (typing.Optional[base.Integer]) – 0-based identifier of the correct answer option, required for polls in quiz mode
- **explanation** (typing.Optional[base.String]) – Text that is shown when a user chooses an incorrect answer or taps on the lamp icon in a quiz-style poll, 0-200 characters with at most 2 line feeds after entities parsing
- **explanation_parse_mode** (typing.Optional[base.String]) – Mode for parsing entities in the explanation. See formatting options for more details.
- **explanation_entities** (typing.Optional[typing.List[types.MessageEntity]]) – List of special entities that appear in message text, which can be specified instead of parse_mode
- **open_period** (typing.Optional[base.Integer]) – Amount of time in seconds the poll will be active after creation, 5-600. Can't be used together with close_date.
- **close_date** (typing.Union[base.Integer, datetime.datetime, datetime.timedelta, None]) – Point in time (Unix timestamp) when the poll will be automatically

closed. Must be at least 5 and no more than 600 seconds in

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

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

Returns On success, the sent Message is returned

Return type `types.Message`

```
async send_dice(chat_id: Union[aiogram.types.base.Integer, aiogram.types.base.String],
               disable_notification: Optional[aiogram.types.base.Boolean] = None, emoji:
               Optional[aiogram.types.base.String] = None, reply_to_message_id:
               Optional[aiogram.types.base.Integer] = None, allow_sending_without_reply:
               Optional[aiogram.types.base.Boolean] = None, reply_markup:
               Optional[Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup,
                             aiogram.types.reply_keyboard.ReplyKeyboardMarkup,
                             aiogram.types.reply_keyboard.ReplyKeyboardRemove,
                             aiogram.types.force_reply.ForceReply]] = None) → aiogram.types.message.Message
```

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

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

Parameters

- **chat_id** (`typing.Union[base.Integer, base.String]`) – Unique identifier for the target chat or username of the target channel (in the format `@channelusername`)
- **emoji** (`typing.Optional[base.String]`) – Emoji on which the dice throw animation is based. Currently, must be one of “”, “”, “”, “”, or “”. Dice can have values 1-6 for “” and “”, values 1-5 for “” and “”, and values 1-64 for “”. Defaults to “”
- **disable_notification** (`typing.Optional[base.Boolean]`) – Sends the message silently. Users will receive a notification with no sound
- **reply_to_message_id** (`typing.Optional[base.Integer]`) – If the message is a reply, ID of the original message
- **allow_sending_without_reply** (`typing.Optional[base.Boolean]`) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]`) – Additional interface options. A JSON-serialized object for an inline keyboard,

custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user

Returns On success, the sent Message is returned

Return type `types.Message`

async send_chat_action(*chat_id: Union[aiogram.types.base.Integer, aiogram.types.base.String], action: aiogram.types.base.String*) → `aiogram.types.base.Boolean`

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

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

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

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

Parameters

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

Returns Returns True on success

Return type `base.Boolean`

async get_user_profile_photos(*user_id: aiogram.types.base.Integer, offset: Optional[aiogram.types.base.Integer] = None, limit: Optional[aiogram.types.base.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.Optional[base.Integer]`) – Sequential number of the first photo to be returned. By default, all photos are returned
- **limit** (`typing.Optional[base.Integer]`) – Limits the number of photos to be retrieved. Values between 1—100 are accepted. Defaults to 100

Returns Returns a `UserProfilePhotos` object

Return type `types.UserProfilePhotos`

async get_file(*file_id: aiogram.types.base.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`

```
async ban_chat_member(chat_id: Union[aiogram.types.base.Integer, aiogram.types.base.String], user_id:
    aiogram.types.base.Integer, until_date:
    Optional[Union[aiogram.types.base.Integer, datetime.datetime,
    datetime.timedelta]] = None, revoke_messages:
    Optional[aiogram.types.base.Boolean] = None) → aiogram.types.base.Boolean
```

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

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

Parameters

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

the chat for the user that is being removed. If False, the user will be able to see messages in the group that were sent before the user was removed. Always True for supergroups and channels. :type revoke_messages: `typing.Optional[base.Boolean]`

Returns Returns True on success

Return type `base.Boolean`

```
async kick_chat_member(chat_id: Union[aiogram.types.base.Integer, aiogram.types.base.String], user_id:
    aiogram.types.base.Integer, until_date:
    Optional[Union[aiogram.types.base.Integer, datetime.datetime,
    datetime.timedelta]] = None, revoke_messages:
    Optional[aiogram.types.base.Boolean] = None) → aiogram.types.base.Boolean
```

Renamed to `ban_chat_member`.

```
async unban_chat_member(chat_id: Union[aiogram.types.base.Integer, aiogram.types.base.String],
    user_id: aiogram.types.base.Integer, only_if_banned:
    Optional[aiogram.types.base.Boolean] = None) →
    aiogram.types.base.Boolean
```

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

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

Parameters

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

Returns Returns True on success

Return type `base.Boolean`

```
async restrict_chat_member(chat_id: Union[aiogram.types.base.Integer, aiogram.types.base.String],
                             user_id: aiogram.types.base.Integer, permissions:
                             Optional[aiogram.types.chat_permissions.ChatPermissions] = None,
                             until_date: Optional[Union[aiogram.types.base.Integer, datetime.datetime,
                             datetime.timedelta]] = None, can_send_messages:
                             Optional[aiogram.types.base.Boolean] = None,
                             can_send_media_messages: Optional[aiogram.types.base.Boolean] =
                             None, can_send_other_messages: Optional[aiogram.types.base.Boolean] =
                             None, can_add_web_page_previews:
                             Optional[aiogram.types.base.Boolean] = None) →
                             aiogram.types.base.Boolean
```

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

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

Parameters

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

Returns Returns True on success

Return type `base.Boolean`

```
async promote_chat_member(chat_id: Union[aiogram.types.base.Integer, aiogram.types.base.String],
                           user_id: aiogram.types.base.Integer, is_anonymous:
                           Optional[aiogram.types.base.Boolean] = None, can_manage_chat:
                           Optional[aiogram.types.base.Boolean] = None, can_change_info:
                           Optional[aiogram.types.base.Boolean] = None, can_post_messages:
                           Optional[aiogram.types.base.Boolean] = None, can_edit_messages:
                           Optional[aiogram.types.base.Boolean] = None, can_delete_messages:
                           Optional[aiogram.types.base.Boolean] = None, can_manage_voice_chats:
                           Optional[aiogram.types.base.Boolean] = None, can_invite_users:
                           Optional[aiogram.types.base.Boolean] = None, can_restrict_members:
                           Optional[aiogram.types.base.Boolean] = None, can_pin_messages:
                           Optional[aiogram.types.base.Boolean] = None, can_promote_members:
                           Optional[aiogram.types.base.Boolean] = None) →
                           aiogram.types.base.Boolean
```

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

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

Parameters

- **chat_id** (typing.Union[base.Integer, base.String]) – Unique identifier for the target chat or username of the target channel
- **user_id** (base.Integer) – Unique identifier of the target user
- **is_anonymous** (typing.Optional[base.Boolean]) – Pass True, if the administrator's presence in the chat is hidden
- **can_manage_chat** (typing.Optional[base.Boolean]) – Pass True, if the administrator can access the chat event log, chat statistics, message statistics in channels, see channel members, see anonymous administrators in supergroups and ignore slow mode. Implied by any other administrator privilege
- **can_change_info** (typing.Optional[base.Boolean]) – Pass True, if the administrator can change chat title, photo and other settings
- **can_post_messages** (typing.Optional[base.Boolean]) – Pass True, if the administrator can create channel posts, channels only
- **can_edit_messages** (typing.Optional[base.Boolean]) – Pass True, if the administrator can edit messages of other users, channels only
- **can_delete_messages** (typing.Optional[base.Boolean]) – Pass True, if the administrator can delete messages of other users
- **can_manage_voice_chats** (typing.Optional[base.Boolean]) – Pass True, if the administrator can manage voice chats, supergroups only
- **can_invite_users** (typing.Optional[base.Boolean]) – Pass True, if the administrator can invite new users to the chat
- **can_restrict_members** (typing.Optional[base.Boolean]) – Pass True, if the administrator can restrict, ban or unban chat members
- **can_pin_messages** (typing.Optional[base.Boolean]) – Pass True, if the administrator can pin messages, supergroups only
- **can_promote_members** (typing.Optional[base.Boolean]) – Pass True, if the administrator can add new administrators with a subset of his own privileges or demote ad-

ministrators that he has promoted, directly or indirectly (promoted by administrators that were appointed by him)

Returns Returns True on success

Return type `base.Boolean`

```
async set_chat_administrator_custom_title(chat_id: Union[aiogram.types.base.Integer,
                                                    aiogram.types.base.String], user_id:
                                                    aiogram.types.base.Integer, custom_title:
                                                    aiogram.types.base.String) →
                                                    aiogram.types.base.Boolean
```

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

Returns True on success.

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

Parameters

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

Returns True on success.

```
async set_chat_permissions(chat_id: Union[aiogram.types.base.Integer, aiogram.types.base.String],
                           permissions: aiogram.types.chat_permissions.ChatPermissions) →
                           aiogram.types.base.Boolean
```

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

Returns True on success.

Parameters

- **chat_id** – Unique identifier for the target chat or username of the target supergroup
- **permissions** – New default chat permissions

Returns True on success.

```
async export_chat_invite_link(chat_id: Union[aiogram.types.base.Integer,
                                              aiogram.types.base.String]) → aiogram.types.base.String
```

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

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

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

Returns Returns exported invite link as String on success

Return type `base.String`


```
async create_chat_invite_link(chat_id: Union[aiogram.types.base.Integer,  
                                     aiogram.types.base.String], expire_date:  
                                     Optional[Union[aiogram.types.base.Integer, datetime.datetime,  
                                                     datetime.timedelta]] = None, member_limit:  
                                     Optional[aiogram.types.base.Integer] = None) →  
                                     aiogram.types.chat_invite_link.ChatInviteLink
```

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

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

Parameters

- **chat_id** (typing.Union[base.Integer, base.String]) – Unique identifier for the target chat or username of the target channel (in the format `@channelusername`)
- **expire_date** (typing.Union[base.Integer, datetime.datetime, datetime.timedelta, None]) – Point in time when the link will expire
- **member_limit** (typing.Optional[base.Integer]) – Maximum number of users that can be members of the chat simultaneously after joining the chat via this invite link; 1-99999

Returns the new invite link as `ChatInviteLink` object.

Return type `types.ChatInviteLink`

```
async edit_chat_invite_link(chat_id: Union[aiogram.types.base.Integer, aiogram.types.base.String],  
                             invite_link: aiogram.types.base.String, expire_date:  
                             Optional[Union[aiogram.types.base.Integer, datetime.datetime,  
                                             datetime.timedelta]] = None, member_limit:  
                             Optional[aiogram.types.base.Integer] = None) →  
                             aiogram.types.chat_invite_link.ChatInviteLink
```

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

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

Parameters

- **chat_id** (typing.Union[base.Integer, base.String]) – Unique identifier for the target chat or username of the target channel (in the format `@channelusername`)
- **invite_link** (base.String) – The invite link to edit
- **expire_date** (typing.Union[base.Integer, datetime.datetime, datetime.timedelta, None]) – Point in time (Unix timestamp) when the link will expire
- **member_limit** (typing.Optional[base.Integer]) – Maximum number of users that can be members of the chat simultaneously after joining the chat via this invite link; 1-99999

Returns edited invite link as a `ChatInviteLink` object.

```
async revoke_chat_invite_link(chat_id: Union[aiogram.types.base.Integer,  
                                     aiogram.types.base.String], invite_link: aiogram.types.base.String) →  
                                     aiogram.types.chat_invite_link.ChatInviteLink
```

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

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

Parameters

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

Returns the revoked invite link as ChatInviteLink object

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

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

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

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

Parameters

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

Returns Returns True on success

Return type base.Boolean

async delete_chat_photo(*chat_id: Union[aiogram.types.base.Integer, aiogram.types.base.String]*) → aiogram.types.base.Boolean

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

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

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

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

Returns Returns True on success

Return type base.Boolean

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

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

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

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

Parameters

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

Returns Returns True on success

Return type base.Boolean

```
async set_chat_description(chat_id: Union[aiogram.types.base.Integer, aiogram.types.base.String],
                           description: Optional[aiogram.types.base.String] = None) →
                           aiogram.types.base.Boolean
```

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

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

Parameters

- **chat_id** (typing.Union[base.Integer, base.String]) – Unique identifier for the target chat or username of the target channel
- **description** (typing.Optional[base.String]) – New chat description, 0-255 characters

Returns Returns True on success

Return type base.Boolean

```
async pin_chat_message(chat_id: Union[aiogram.types.base.Integer, aiogram.types.base.String],
                       message_id: aiogram.types.base.Integer, disable_notification:
                       Optional[aiogram.types.base.Boolean] = None) → aiogram.types.base.Boolean
```

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

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

Parameters

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

Returns Returns True on success

Return type base.Boolean

```
async unpin_chat_message(chat_id: Union[aiogram.types.base.Integer, aiogram.types.base.String],
                          message_id: Optional[aiogram.types.base.Integer] = None) →
                          aiogram.types.base.Boolean
```

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

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

Parameters

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

Returns Returns True on success

Return type `base.Boolean`

async unpin_all_chat_messages(*chat_id*: `Union[aiogram.types.base.Integer, aiogram.types.base.String]`) → `aiogram.types.base.Boolean`

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

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

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

Returns Returns True on success

Return type `base.Boolean`

async leave_chat(*chat_id*: `Union[aiogram.types.base.Integer, aiogram.types.base.String]`) → `aiogram.types.base.Boolean`

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

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

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

Returns Returns True on success

Return type `base.Boolean`

async get_chat(*chat_id*: `Union[aiogram.types.base.Integer, aiogram.types.base.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`

async get_chat_administrators(*chat_id*: `Union[aiogram.types.base.Integer, aiogram.types.base.String]`) → `List[Union[aiogram.types.chat_member.ChatMemberOwner, aiogram.types.chat_member.ChatMemberAdministrator]]`

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

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

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

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

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

async get_chat_member_count(*chat_id*: `Union[aiogram.types.base.Integer, aiogram.types.base.String]`) → `aiogram.types.base.Integer`

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

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

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

Returns Returns `Int` on success

Return type `base.Integer`

async `get_chat_members_count(chat_id: Union[aiogram.types.base.Integer, aiogram.types.base.String])`
→ `aiogram.types.base.Integer`

Renamed to `get_chat_member_count`.

async `get_chat_member(chat_id: Union[aiogram.types.base.Integer, aiogram.types.base.String], user_id: aiogram.types.base.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`

async `set_chat_sticker_set(chat_id: Union[aiogram.types.base.Integer, aiogram.types.base.String], sticker_set_name: aiogram.types.base.String)` →
`aiogram.types.base.Boolean`

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

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

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

Parameters

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

Returns Returns `True` on success

Return type `base.Boolean`

async `delete_chat_sticker_set(chat_id: Union[aiogram.types.base.Integer, aiogram.types.base.String])` → `aiogram.types.base.Boolean`

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

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

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

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

Returns Returns True on success

Return type base.Boolean

```
async answer_callback_query(callback_query_id: aiogram.types.base.String, text:
    Optional[aiogram.types.base.String] = None, show_alert:
    Optional[aiogram.types.base.Boolean] = None, url:
    Optional[aiogram.types.base.String] = None, cache_time:
    Optional[aiogram.types.base.Integer] = None) →
    aiogram.types.base.Boolean
```

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

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

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

Parameters

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

Returns On success, True is returned

Return type base.Boolean

```
async set_my_commands(commands: List[aiogram.types.bot_command.BotCommand], scope:
    Optional[aiogram.types.bot_command_scope.BotCommandScope] = None,
    language_code: Optional[aiogram.types.base.String] = None) →
    aiogram.types.base.Boolean
```

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

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

Parameters

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

Returns Returns True on success.

Return type base.Boolean

```
async delete_my_commands(scope: Optional[aiogram.types.bot_command_scope.BotCommandScope] =  
    None, language_code: Optional[aiogram.types.base.String] = None) →  
    aiogram.types.base.Boolean
```

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

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

Parameters

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

Returns Returns True on success.

Return type base.Boolean

```
async get_my_commands(scope: Optional[aiogram.types.bot_command_scope.BotCommandScope] =  
    None, language_code: Optional[aiogram.types.base.String] = None) →  
    List[aiogram.types.bot_command.BotCommand]
```

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

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

Parameters

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

Returns Returns Array of BotCommand on success or empty list.

Return type typing.List[types.BotCommand]

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

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

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

```
async edit_message_caption(chat_id: Optional[Union[aiogram.types.base.Integer,
aiogram.types.base.String]] = None, message_id:
Optional[aiogram.types.base.Integer] = None, inline_message_id:
Optional[aiogram.types.base.String] = None, caption:
Optional[aiogram.types.base.String] = None, parse_mode:
Optional[aiogram.types.base.String] = None, caption_entities:
Optional[List[aiogram.types.message_entity.MessageEntity]] = 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.Optional[base.Integer]`) – Required if `inline_message_id` is not specified. Identifier of the sent message
- **inline_message_id** (`typing.Optional[base.String]`) – Required if `chat_id` and `message_id` are not specified. Identifier of the inline message
- **caption** (`typing.Optional[base.String]`) – New caption of the message
- **parse_mode** (`typing.Optional[base.String]`) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.
- **caption_entities** (`typing.Optional[typing.List[types.MessageEntity]]`) – List of special entities that appear in message text, which can be specified instead of `parse_mode`

- **reply_markup** (`typing.Optional[types.InlineKeyboardMarkup]`) – A JSON-serialized object for an inline keyboard

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

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

```
async edit_message_media(media: aiogram.types.input_media.InputMedia, chat_id:
    Optional[Union[aiogram.types.base.Integer, aiogram.types.base.String]] =
    None, message_id: Optional[aiogram.types.base.Integer] = None,
    inline_message_id: Optional[aiogram.types.base.String] = None,
    reply_markup:
    Optional[aiogram.types.inline_keyboard.InlineKeyboardMarkup] = None) →
    Union[aiogram.types.message.Message, aiogram.types.base.Boolean]
```

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

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

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

Parameters

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

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

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

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

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

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

async stop_poll(*chat_id: Union[aiogram.types.base.String, aiogram.types.base.Integer], message_id: aiogram.types.base.Integer, reply_markup: Optional[aiogram.types.inline_keyboard.InlineKeyboardMarkup] = None*) → `aiogram.types.poll.Poll`

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

Parameters

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

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

Return type `types.Poll`

async delete_message(*chat_id: Union[aiogram.types.base.Integer, aiogram.types.base.String], message_id: aiogram.types.base.Integer*) → `aiogram.types.base.Boolean`

Use this method to delete a message, including service messages, with the following limitations: - A message can only be deleted if it was sent less than 48 hours ago. - Bots can delete outgoing messages in private chats, groups, and supergroups. - Bots can delete incoming messages in private chats. - Bots granted `can_post_messages` permissions can delete outgoing messages in channels. - If the bot is an administrator of a group, it can delete any message there. - If the bot has `can_delete_messages` permission in a supergroup or a channel, it can delete any message there.

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

Parameters

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

Returns Returns `True` on success

Return type `base.Boolean`

```
async send_sticker(chat_id: Union[aiogram.types.base.Integer, aiogram.types.base.String], sticker:
    Union[aiogram.types.base.InputFile, aiogram.types.base.String],
    disable_notification: Optional[aiogram.types.base.Boolean] = None,
    reply_to_message_id: Optional[aiogram.types.base.Integer] = None,
    allow_sending_without_reply: Optional[aiogram.types.base.Boolean] = None,
    reply_markup:
        Optional[Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup,
            aiogram.types.reply_keyboard.ReplyKeyboardMarkup,
            aiogram.types.reply_keyboard.ReplyKeyboardRemove,
            aiogram.types.force_reply.ForceReply]] = 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.Optional[base.Boolean]) – Sends the message silently. Users will receive a notification with no sound
- **reply_to_message_id** (typing.Optional[base.Integer]) – If the message is a reply, ID of the original message
- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user

Returns On success, the sent Message is returned

Return type types.Message

```
async get_sticker_set(name: aiogram.types.base.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

```
async upload_sticker_file(user_id: aiogram.types.base.Integer, png_sticker:
    aiogram.types.base.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`

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

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

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

Parameters

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

Returns Returns `True` on success

Return type `base.Boolean`

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

Use this method to add a new sticker to a set created by the bot. You must use exactly one of the fields

png_sticker or tgs_sticker. Animated stickers can be added to animated sticker sets and only to them. Animated sticker sets can have up to 50 stickers. Static sticker sets can have up to 120 stickers.

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

Parameters

- **user_id** (`base.Integer`) – User identifier of sticker set owner
- **name** (`base.String`) – Sticker set name
- **png_sticker** (`typing.Union[base.InputFile, base.String]`) – PNG image with the sticker, must be up to 512 kilobytes in size, dimensions must not exceed 512px, and either width or height must be exactly 512px. Pass a `file_id` as a `String` to send a file that already exists on the Telegram servers, pass an HTTP URL as a `String` for Telegram to get a file from the Internet, or upload a new one using multipart/form-data. More info on <https://core.telegram.org/bots/api#sending-files>
- **tgs_sticker** (`base.InputFile`) – TGS animation with the sticker, uploaded using multipart/form-data. See https://core.telegram.org/animated_stickers#technical-requirements for technical requirements
- **emojis** (`base.String`) – One or more emoji corresponding to the sticker
- **mask_position** (`typing.Optional[types.MaskPosition]`) – A JSON-serialized object for position where the mask should be placed on faces

Returns Returns True on success

Return type `base.Boolean`

async set_sticker_position_in_set (*sticker: aiogram.types.base.String, position: aiogram.types.base.Integer*) → `aiogram.types.base.Boolean`

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

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

Parameters

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

Returns Returns True on success

Return type `base.Boolean`

async delete_sticker_from_set (*sticker: aiogram.types.base.String*) → `aiogram.types.base.Boolean`

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

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

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

Returns Returns True on success

Return type `base.Boolean`

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

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

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

Parameters

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

Returns Returns True on success

Return type `base.Boolean`

```
async answer_inline_query(inline_query_id: aiogram.types.base.String, results:
    List[aiogram.types.inline_query_result.InlineQueryResult], cache_time:
    Optional[aiogram.types.base.Integer] = None, is_personal:
    Optional[aiogram.types.base.Boolean] = None, next_offset:
    Optional[aiogram.types.base.String] = None, switch_pm_text:
    Optional[aiogram.types.base.String] = None, switch_pm_parameter:
    Optional[aiogram.types.base.String] = None) →
    aiogram.types.base.Boolean
```

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

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

Parameters

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

Returns On success, True is returned

Return type `base.Boolean`

```
async send_invoice(chat_id: Union[aiogram.types.base.Integer, aiogram.types.base.String], title:
    aiogram.types.base.String, description: aiogram.types.base.String, payload:
    aiogram.types.base.String, provider_token: aiogram.types.base.String, currency:
    aiogram.types.base.String, prices: List[aiogram.types.labeled_price.LabeledPrice],
    max_tip_amount: Optional[aiogram.types.base.Integer] = None,
    suggested_tip_amounts: Optional[List[aiogram.types.base.Integer]] = None,
    start_parameter: Optional[aiogram.types.base.String] = None, provider_data:
    Optional[Dict] = None, photo_url: Optional[aiogram.types.base.String] = None,
    photo_size: Optional[aiogram.types.base.Integer] = None, photo_width:
    Optional[aiogram.types.base.Integer] = None, photo_height:
    Optional[aiogram.types.base.Integer] = None, need_name:
    Optional[aiogram.types.base.Boolean] = None, need_phone_number:
    Optional[aiogram.types.base.Boolean] = None, need_email:
    Optional[aiogram.types.base.Boolean] = None, need_shipping_address:
    Optional[aiogram.types.base.Boolean] = None, send_phone_number_to_provider:
    Optional[aiogram.types.base.Boolean] = None, send_email_to_provider:
    Optional[aiogram.types.base.Boolean] = None, is_flexible:
    Optional[aiogram.types.base.Boolean] = None, disable_notification:
    Optional[aiogram.types.base.Boolean] = None, reply_to_message_id:
    Optional[aiogram.types.base.Integer] = None, allow_sending_without_reply:
    Optional[aiogram.types.base.Boolean] = 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** (typing.Union[base.Integer, base.String]) – Unique identifier for the target chat or username of the target channel (in the format @channelusername)
- **title** (base.String) – Product name, 1-32 characters
- **description** (base.String) – Product description, 1-255 characters
- **payload** (base.String) – Bot-defined invoice payload, 1-128 bytes This will not be displayed to the user, use for your internal processes.
- **provider_token** (base.String) – Payments provider token, obtained via Botfather
- **currency** (base.String) – Three-letter ISO 4217 currency code, see more on currencies
- **prices** (typing.List[types.LabeledPrice]) – Price breakdown, a list of components (e.g. product price, tax, discount, delivery cost, delivery tax, bonus, etc.)
- **max_tip_amount** (typing.Optional[base.Integer]) – The maximum accepted amount for tips in the smallest units of the currency (integer, not float/double). For example, for a maximum tip of US\$ 1.45 pass max_tip_amount = 145. See the exp parameter in currencies.json, it shows the number of digits past the decimal point for each currency (2 for the majority of currencies). Defaults to 0
- **suggested_tip_amounts** (typing.Optional[typing.List[base.Integer]]) – A JSON-serialized array of suggested amounts of tips in the smallest units of the currency (integer, not float/double). At most 4 suggested tip amounts can be specified. The suggested tip amounts must be positive, passed in a strictly increased order and must not exceed max_tip_amount.
- **start_parameter** (typing.Optional[base.String]) – Unique deep-linking parameter. If left empty, forwarded copies of the sent message will have a Pay button, allowing

multiple users to pay directly from the forwarded message, using the same invoice. If non-empty, forwarded copies of the sent message will have a URL button with a deep link to the bot (instead of a Pay button), with the value used as the start parameter

- **provider_data** (`typing.Optional[typing.Dict]`) – JSON-encoded data about the invoice, which will be shared with the payment provider
- **photo_url** (`typing.Optional[base.String]`) – URL of the product photo for the invoice
- **photo_size** (`typing.Optional[base.Integer]`) – Photo size
- **photo_width** (`typing.Optional[base.Integer]`) – Photo width
- **photo_height** (`typing.Optional[base.Integer]`) – Photo height
- **need_name** (`typing.Optional[base.Boolean]`) – Pass True, if you require the user's full name to complete the order
- **need_phone_number** (`typing.Optional[base.Boolean]`) – Pass True, if you require the user's phone number to complete the order
- **need_email** (`typing.Optional[base.Boolean]`) – Pass True, if you require the user's email to complete the order
- **need_shipping_address** (`typing.Optional[base.Boolean]`) – Pass True, if you require the user's shipping address to complete the order
- **send_phone_number_to_provider** (`typing.Optional[base.Boolean]`) – Pass True, if user's phone number should be sent to provider
- **send_email_to_provider** (`typing.Optional[base.Boolean]`) – Pass True, if user's email address should be sent to provider
- **is_flexible** (`typing.Optional[base.Boolean]`) – Pass True, if the final price depends on the shipping method
- **disable_notification** (`typing.Optional[base.Boolean]`) – Sends the message silently. Users will receive a notification with no sound
- **reply_to_message_id** (`typing.Optional[base.Integer]`) – If the message is a reply, ID of the original message
- **allow_sending_without_reply** (`typing.Optional[base.Boolean]`) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (`typing.Optional[types.InlineKeyboardMarkup]`) – A JSON-serialized object for an inline keyboard. If empty, one 'Pay total price' button will be shown. If not empty, the first button must be a Pay button.

Returns On success, the sent Message is returned

Return type `types.Message`

```
async answer_shipping_query(shipping_query_id: aiogram.types.base.String, ok:
    aiogram.types.base.Boolean, shipping_options:
    Optional[List[aiogram.types.shipping_option.ShippingOption]] = None,
    error_message: Optional[aiogram.types.base.String] = None) →
    aiogram.types.base.Boolean
```

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

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

Parameters

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

Returns On success, True is returned

Return type `base.Boolean`

```
async answer_pre_checkout_query(pre_checkout_query_id: aiogram.types.base.String, ok:
                                aiogram.types.base.Boolean, error_message:
                                Optional[aiogram.types.base.String] = None) →
                                aiogram.types.base.Boolean
```

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

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

Parameters

- **pre_checkout_query_id** (`base.String`) – Unique identifier for the query to be answered
- **ok** (`base.Boolean`) – Specify True if everything is alright (goods are available, etc.) and the bot is ready to proceed with the order. Use False if there are any problems.
- **error_message** (`typing.Optional[base.String]`) – Required if ok is False Error message in human readable form that explains the reason for failure to proceed with the checkout (e.g. “Sorry, somebody just bought the last of our amazing black T-shirts while you were busy filling out your payment details. Please choose a different color or garment!”). Telegram will display this message to the user.

Returns On success, True is returned

Return type `base.Boolean`

```
async set_passport_data_errors(user_id: aiogram.types.base.Integer, errors:
                               List[aiogram.types.passport_element_error.PassportElementError])
                               → aiogram.types.base.Boolean
```

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

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

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

Parameters

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

Returns Returns True on success

Return type `base.Boolean`

```
async send_game(chat_id: aiogram.types.base.Integer, game_short_name: aiogram.types.base.String,
                 disable_notification: Optional[aiogram.types.base.Boolean] = None,
                 reply_to_message_id: Optional[aiogram.types.base.Integer] = None,
                 allow_sending_without_reply: Optional[aiogram.types.base.Boolean] = 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.Optional[base.Boolean]`) – Sends the message silently. Users will receive a notification with no sound
- **reply_to_message_id** (`typing.Optional[base.Integer]`) – If the message is a reply, ID of the original message
- **allow_sending_without_reply** (`typing.Optional[base.Boolean]`) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (`typing.Optional[types.InlineKeyboardMarkup]`) – A JSON-serialized object for an inline keyboard If empty, one ‘Play game_title’ button will be shown. If not empty, the first button must launch the game.

Returns On success, the sent Message is returned

Return type `types.Message`

```
async set_game_score(user_id: aiogram.types.base.Integer, score: aiogram.types.base.Integer, force:
                     Optional[aiogram.types.base.Boolean] = None, disable_edit_message:
                     Optional[aiogram.types.base.Boolean] = None, chat_id:
                     Optional[aiogram.types.base.Integer] = None, message_id:
                     Optional[aiogram.types.base.Integer] = None, inline_message_id:
                     Optional[aiogram.types.base.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.Optional[base.Boolean]`) – Pass True, if the high score is allowed to decrease This can be useful when fixing mistakes or banning cheaters
- **disable_edit_message** (`typing.Optional[base.Boolean]`) – Pass True, if the game message should not be automatically edited to include the current scoreboard

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

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

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

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

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

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

API Helpers

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

Bases: object

Base config for API Endpoints

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

Generate URL for API methods

Parameters

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

Returns URL

file_url(*token: str, path: str*) → str
Generate URL for downloading files

Parameters

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

Returns URL

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

Parameters **token** –

Returns

aiogram.bot.api.check_result(*method_name: str, content_type: str, status_code: int, body: str*)
Checks whether *result* is a valid API response. A result is considered invalid if: - The server returned an HTTP response code other than 200 - The content of the result is invalid JSON. - The method call was unsuccessful (The JSON 'ok' field equals False)

Parameters

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

Returns The result parsed to a JSON dictionary

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

aiogram.bot.api.guess_filename(*obj*)
Get file name from object

Parameters **obj** –

Returns

aiogram.bot.api.compose_data(*params=None, files=None*)
Prepare request data

Parameters

- **params** –
- **files** –

Returns

class **aiogram.bot.api.Methods**
Bases: **aiogram.utils.helper.Helper**
Helper for Telegram API Methods listed on <https://core.telegram.org/bots/api>
List is updated to Bot API 5.3

4.4.2 Telegram data types

Bases

Base TelegramObject

MetaTelegramObject

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

Bases: type

Metaclass for telegram objects

TelegramObject

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

Abstract class for telegram objects

Deserialize object

Parameters

- **conf** –
- **kwargs** –

```
property props: Dict[str, aiogram.types.fields.BaseField]  
    Get props
```

Returns dict with props

```
property props_aliases: Dict[str, str]  
    Get aliases for props
```

Returns

```
property values: Dict[str, Any]  
    Get values
```

Returns

```
classmethod to_object(data: Dict[str, Any], conf: Optional[Dict[str, Any]] = None) →  
    aiogram.types.base.T
```

Deserialize object

Parameters

- **data** –
- **conf** –

Returns

```
to_python() → Dict[str, Any]  
    Get object as JSON serializable
```

Returns

clean() → None

Remove empty values

as_json() → str

Get object as JSON string

Returns JSON

Return type str

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

Iterate over keys

Returns

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

Iterate over values

Returns

Fields

BaseField

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

Bases: object

Base field (prop)

Init prop

Parameters

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

get_value(*instance*)

Get value for the current object instance

Parameters **instance** –

Returns

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

Set prop value

Parameters

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

Returns

abstract serialize(*value*)

Serialize value to python

Parameters value –

Returns

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

Deserialize python object value to TelegramObject value

export(*instance*)

Alias for *serialize* but for current Object instance

Parameters instance –

Returns

Field

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

Bases: [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

Init prop

Parameters

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

serialize(*value*)

Serialize value to python

Parameters *value* –

Returns

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

Deserialize python object value to TelegramObject value

ListOfLists

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

Bases: [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](#)

Init prop

Parameters

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

serialize(*value*)

Serialize value to python

Parameters value –

Returns

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

Deserialize python object value to TelegramObject value

Mixins

Downloadable

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

Bases: object

Mixin for files

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

Download file

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

Parameters

- **destination** – deprecated, use :param destination_dir: or :param destination_file: instead
- **timeout** – Integer
- **chunk_size** – Integer
- **seek** – Boolean - go to start of file when downloading is finished.

- **make_dirs** – Make dirs if not exist
- **destination_dir** – directory for saving files
- **destination_file** – path to the file or instance of `io.IOBase`. For e. g. `io.BytesIO`

Returns destination

async get_file()

Get file information

Returns `aiogram.types.File`

async get_url()

Get file url.

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

Returns url

Types

StickerSet

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

Bases: `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: Optional[Dict[str, Any]] = None, **kwargs: Any*)

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: Optional[Dict[str, Any]] = None, **kwargs: Any)
```

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** –

```
async answer(text: Optional[aiogram.types.base.String] = None, show_alert: Optional[aiogram.types.base.Boolean] = None, url: Optional[aiogram.types.base.String] = None, cache_time: Optional[aiogram.types.base.Integer] = None)
```

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

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

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

Parameters

- **text** (`typing.Optional[base.String]`) – Text of the notification. If not specified, nothing will be shown to the user, 0-200 characters
- **show_alert** (`typing.Optional[base.Boolean]`) – If true, an alert will be shown by the client instead of a notification at the top of the chat screen. Defaults to false.
- **url** (`typing.Optional[base.String]`) – URL that will be opened by the user's client.
- **cache_time** (`typing.Optional[base.Integer]`) – The maximum amount of time in seconds that the result of the callback query may be cached client-side.

Returns On success, `True` is returned.

Return type `base.Boolean`

SuccessfulPayment

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

Bases: *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**(*type: aiogram.types.base.String, offset: aiogram.types.base.Integer, length: aiogram.types.base.Integer, url: Optional[aiogram.types.base.String] = None, user: Optional[aiogram.types.user.User] = None, language: Optional[aiogram.types.base.String] = 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 UNDERLINE

Key STRIKETHROUGH

Key TEXT_LINK

Key TEXT_MENTION

ShippingQuery

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

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: Optional[Dict[str, Any]] = None, **kwargs: Any*)

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, **kwargs*)

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>

Deserialize object

Parameters

- **conf** –
- **kwargs** –

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: aiogram.types.base.String, url:
    Optional[aiogram.types.base.String] =
    None, login_url: Op-
    tional[aiogram.types.login_url.LoginUrl]
    = None, callback_data:
    Optional[aiogram.types.base.String] =
    None, switch_inline_query:
    Optional[aiogram.types.base.String] =
    None, switch_inline_query_current_chat:
    Optional[aiogram.types.base.String] =
    None, callback_game: Op-
    tional[aiogram.types.callback_game.CallbackGame]
    = None, pay:
    Optional[aiogram.types.base.Boolean] =
    None, **kwargs)
```

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>

Deserialize object

Parameters

- **conf** –
- **kwargs** –

User

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

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** –

property full_name

You can get full name of user.

Returns str

property mention

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

Returns str

property locale: Optional[babel.core.Locale]

Get user's locale

Returns babel.core.Locale

Video

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

    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: Optional[Dict[str, Any]] = None, **kwargs: Any)
    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: Optional[Dict[str, Any]] = None, **kwargs: Any)
    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: aiogram.types.base.String, amount:
                                                aiogram.types.base.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>

Deserialize object

Parameters

- **conf** –
- **kwargs** –

CallbackGame

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

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.KeyboardButton]
= None, resize_keyboard: Optional[aiogram.types.base.Boolean] = None, one_time_keyboard: Optional[aiogram.types.base.Boolean] = None, input_field_placeholder: Optional[aiogram.types.base.String] = None, selective: Optional[aiogram.types.base.Boolean] = None, row_width: aiogram.types.base.Integer = 3)
```

Bases: *aiogram.types.base.TelegramObject*

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

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

Deserialize object

Parameters

- **conf** –
- **kwargs** –

add(*args)

Add buttons

Parameters **args** –

Returns self

Return type types.ReplyKeyboardMarkup

row(*args)

Add row

Parameters **args** –

Returns self

Return type types.ReplyKeyboardMarkup

insert(button)

Insert button to last row

Parameters **button** –

Returns self

Return type types.ReplyKeyboardMarkup

KeyboardButton

```
class aiogram.types.reply_keyboard.KeyboardButton(text: aiogram.types.base.String, request_contact:
Optional[aiogram.types.base.Boolean] = None,
request_location:
Optional[aiogram.types.base.Boolean] = None,
request_poll: Op-
tional[aiogram.types.reply_keyboard.KeyboardButtonPollType]
= None, **kwargs)
```

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 request_contact, request_location, and request_poll are mutually exclusive. Note: request_contact and request_location options will only work in Telegram versions released after 9 April, 2016. Older clients will ignore them. Note: request_poll option will only work in Telegram versions released after 23 January, 2020. Older clients will receive unsupported message.

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

Deserialize object

Parameters

- **conf** –
- **kwargs** –

ReplyKeyboardRemove

```
class aiogram.types.reply_keyboard.ReplyKeyboardRemove(selective:
Optional[aiogram.types.base.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>

Deserialize object

Parameters

- **conf** –
- **kwargs** –

Chat

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

Bases: `aiogram.types.base.TelegramObject`

This object represents a chat.

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

Deserialize object

Parameters

- **conf** –
- **kwargs** –

property mention: `Optional[aiogram.types.base.String]`

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

property shifted_id: `int`

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

For example: -1001122334455 -> 1122334455

async get_url() → `aiogram.types.base.String`

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

async update_chat()

Use this method to update Chat data

Returns `None`

async set_photo(*photo: aiogram.types.input_file.InputFile*) → `aiogram.types.base.Boolean`

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

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

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

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

Returns Returns True on success.

Return type `base.Boolean`

async delete_photo() → `aiogram.types.base.Boolean`

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

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

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

Returns Returns True on success.

Return type `base.Boolean`

async set_title(*title: aiogram.types.base.String*) → `aiogram.types.base.Boolean`

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

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

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

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

Returns Returns True on success.

Return type `base.Boolean`

async set_description(*description: aiogram.types.base.String*) → `aiogram.types.base.Boolean`

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

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

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

Returns Returns True on success.

Return type `base.Boolean`

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

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

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

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

Parameters

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

Returns Returns True on success

Return type `base.Boolean`

async unban(*user_id: aiogram.types.base.Integer, only_if_banned: Optional[aiogram.types.base.Boolean] = None*) → `aiogram.types.base.Boolean`

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

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

Parameters

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

Returns Returns True on success.

Return type `base.Boolean`

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

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

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

Parameters

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

Returns Returns True on success.

Return type `base.Boolean`

async promote(*user_id: aiogram.types.base.Integer, is_anonymous: Optional[aiogram.types.base.Boolean] = None, can_change_info: Optional[aiogram.types.base.Boolean] = None, can_post_messages: Optional[aiogram.types.base.Boolean] = None, can_edit_messages: Optional[aiogram.types.base.Boolean] = None, can_delete_messages: Optional[aiogram.types.base.Boolean] = None, can_invite_users: Optional[aiogram.types.base.Boolean] = None, can_restrict_members: Optional[aiogram.types.base.Boolean] = None, can_pin_messages: Optional[aiogram.types.base.Boolean] = None, can_promote_members: Optional[aiogram.types.base.Boolean] = None*) → `aiogram.types.base.Boolean`

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

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

Parameters

- **user_id** (`base.Integer`) – Unique identifier of the target user
- **is_anonymous** (`typing.Optional[base.Boolean]`) – Pass True, if the administrator's presence in the chat is hidden
- **can_change_info** (`typing.Optional[base.Boolean]`) – Pass True, if the administrator can change chat title, photo and other settings
- **can_post_messages** (`typing.Optional[base.Boolean]`) – Pass True, if the administrator can create channel posts, channels only
- **can_edit_messages** (`typing.Optional[base.Boolean]`) – Pass True, if the administrator can edit messages of other users, channels only
- **can_delete_messages** (`typing.Optional[base.Boolean]`) – Pass True, if the administrator can delete messages of other users
- **can_invite_users** (`typing.Optional[base.Boolean]`) – Pass True, if the administrator can invite new users to the chat
- **can_restrict_members** (`typing.Optional[base.Boolean]`) – Pass True, if the administrator can restrict, ban or unban chat members
- **can_pin_messages** (`typing.Optional[base.Boolean]`) – Pass True, if the administrator can pin messages, supergroups only
- **can_promote_members** (`typing.Optional[base.Boolean]`) – Pass True, if the administrator can add new administrators with a subset of his own privileges or demote administrators that he has promoted, directly or indirectly (promoted by administrators that were appointed by him)

Returns Returns True on success.

Return type `base.Boolean`

async set_permissions(*permissions: aiogram.types.chat_permissions.ChatPermissions*) → `aiogram.types.base.Boolean`

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

Returns True on success.

Parameters **permissions** – New default chat permissions

Returns True on success.

async set_administrator_custom_title(*user_id: aiogram.types.base.Integer, custom_title: aiogram.types.base.String*) → `aiogram.types.base.Boolean`

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

Returns True on success.

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

Parameters

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

Returns True on success.

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

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

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

Parameters

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

Returns Returns True on success

Return type base.Boolean

async unpin_message(*message_id*: Optional[aiogram.types.base.Integer] = None) → aiogram.types.base.Boolean

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

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

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

Returns Returns True on success

Return type base.Boolean

async unpin_all_messages()

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

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

Returns Returns True on success

Return type base.Boolean

async leave() → aiogram.types.base.Boolean

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

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

Returns Returns True on success.

Return type base.Boolean

async get_administrators() → List[Union[aiogram.types.chat_member.ChatMemberOwner, aiogram.types.chat_member.ChatMemberAdministrator]]

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

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

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

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

async get_member_count() → `aiogram.types.base.Integer`

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

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

Returns Returns `Int` on success.

Return type `base.Integer`

async get_members_count() → `aiogram.types.base.Integer`

Renamed to `get_member_count`.

async get_member(*user_id: aiogram.types.base.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 *user_id* (`base.Integer`) – Unique identifier of the target user

Returns Returns a `ChatMember` object on success.

Return type `types.ChatMember`

async set_sticker_set(*sticker_set_name: aiogram.types.base.String*) → `aiogram.types.base.Boolean`

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

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

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

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

Returns Returns `True` on success

Return type `base.Boolean`

async delete_sticker_set() → `aiogram.types.base.Boolean`

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

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

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

Returns Returns `True` on success

Return type `base.Boolean`

async do(*action: aiogram.types.base.String*) → `aiogram.types.base.Boolean`

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

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

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

Parameters *action* (`base.String`) – Type of action to broadcast.

Returns Returns `True` on success.

Return type `base.Boolean`

async export_invite_link() → `aiogram.types.base.String`

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

Source: <https://core.telegram.org/bots/api#exportchatinviolink>

Returns Returns exported invite link as `String` on success.

Return type `base.String`

async create_invite_link(*expire_date: Optional[Union[aiogram.types.base.Integer, datetime.datetime, datetime.timedelta]] = None, member_limit: Optional[aiogram.types.base.Integer] = None*) → `aiogram.types.chat_invite_link.ChatInviteLink`

Shortcut for `createChatInviteLink` method.

async edit_invite_link(*invite_link: aiogram.types.base.String, expire_date: Optional[Union[aiogram.types.base.Integer, datetime.datetime, datetime.timedelta]] = None, member_limit: Optional[aiogram.types.base.Integer] = None*) → `aiogram.types.chat_invite_link.ChatInviteLink`

Shortcut for `editChatInviteLink` method.

async revoke_invite_link(*invite_link: aiogram.types.base.String*) → `aiogram.types.chat_invite_link.ChatInviteLink`

Shortcut for `revokeChatInviteLink` method.

async delete_message(*message_id: aiogram.types.base.Integer*) → `aiogram.types.base.Boolean`

Shortcut for `deleteMessage` method.

ChatType

class `aiogram.types.chat.ChatType`

Bases: `aiogram.utils.helper.Helper`

List of chat types

Key `PRIVATE`

Key `GROUP`

Key `SUPER_GROUP`

Key `SUPERGROUP`

Key `CHANNEL`

classmethod `is_private(obj)` → `bool`

Check chat is private

Parameters `obj` –

Returns

classmethod `is_group(obj)` → `bool`

Check chat is group

Parameters `obj` –

Returns

classmethod **is_super_group**(*obj*) → bool

Check chat is super-group

Parameters *obj* –

Returns

classmethod **is_group_or_super_group**(*obj*) → bool

Check chat is group or super-group

Parameters *obj* –

Returns

classmethod **is_channel**(*obj*) → bool

Check chat is channel

Parameters *obj* –

Returns

ChatActions

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

Bases: 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

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

Do typing

Parameters *sleep* – sleep timeout

Returns

async classmethod upload_photo(*sleep=None*)

Do upload_photo

Parameters **sleep** – sleep timeout

Returns

async classmethod record_video(*sleep=None*)

Do record video

Parameters **sleep** – sleep timeout

Returns

async classmethod upload_video(*sleep=None*)

Do upload video

Parameters **sleep** – sleep timeout

Returns

async classmethod record_audio(*sleep=None*)

Do record audio

Parameters **sleep** – sleep timeout

Returns

async classmethod upload_audio(*sleep=None*)

Do upload audio

Parameters **sleep** – sleep timeout

Returns

async classmethod record_voice(*sleep=None*)

Do record voice

Parameters **sleep** – sleep timeout

Returns

async classmethod upload_voice(*sleep=None*)

Do upload voice

Parameters **sleep** – sleep timeout

Returns

async classmethod upload_document(*sleep=None*)

Do upload document

Parameters **sleep** – sleep timeout

Returns

async classmethod find_location(*sleep=None*)

Do find location

Parameters **sleep** – sleep timeout

Returns

async classmethod record_video_note(*sleep=None*)

Do record video note

Parameters **sleep** – sleep timeout

Returns**async classmethod** `upload_video_note(sleep=None)`

Do upload video note

Parameters `sleep` – sleep timeout**Returns****Document****class** `aiogram.types.document.Document`(*conf: Optional[Dict[str, Any]] = None, **kwargs: Any*)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: Optional[Dict[str, Any]] = None, **kwargs: Any*)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: Optional[Dict[str, Any]] = None, **kwargs: Any*)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.

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

Deserialize object

Parameters

- `conf` –
- `kwargs` –

```
classmethod create(input_field_placeholder: Optional[aiogram.types.base.String] = None, selective:
                    Optional[aiogram.types.base.Boolean] = None) →
                    aiogram.types.force_reply.ForceReply
```

Create new force reply

Parameters

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

Returns

PassportElementError

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

Bases: *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:
                                                                            aiogram.types.base.String,
                                                                            type:
                                                                            aiogram.types.base.String,
                                                                            field_name:
                                                                            aiogram.types.base.String,
                                                                            data_hash:
                                                                            aiogram.types.base.String,
                                                                            message:
                                                                            aiogram.types.base.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>

Deserialize object

Parameters

- **conf** –
- **kwargs** –

PassportElementErrorFile

```
class aiogram.types.passport_element_error.PassportElementErrorFile(source:
                                                                    aiogram.types.base.String,
                                                                    type:
                                                                    aiogram.types.base.String,
                                                                    file_hash:
                                                                    aiogram.types.base.String,
                                                                    message:
                                                                    aiogram.types.base.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>

Deserialize object

Parameters

- **conf** –
- **kwargs** –

PassportElementErrorFiles

```
class aiogram.types.passport_element_error.PassportElementErrorFiles(source:
                                                                    aiogram.types.base.String,
                                                                    type:
                                                                    aiogram.types.base.String,
                                                                    file_hashes:
                                                                    List[aiogram.types.base.String],
                                                                    message:
                                                                    aiogram.types.base.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>

Deserialize object

Parameters

- **conf** –
- **kwargs** –

PassportElementErrorFrontSide

```
class aiogram.types.passport_element_error.PassportElementErrorFrontSide(source:
    aiogram.types.base.String,
    type:
    aiogram.types.base.String,
    file_hash:
    aiogram.types.base.String,
    message:
    aiogram.types.base.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>

Deserialize object

Parameters

- **conf** –
- **kwargs** –

PassportElementErrorReverseSide

```
class aiogram.types.passport_element_error.PassportElementErrorReverseSide(source:
    aiogram.types.base.String,
    type:
    aiogram.types.base.String,
    file_hash:
    aiogram.types.base.String,
    message:
    aiogram.types.base.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>

Deserialize object

Parameters

- **conf** –
- **kwargs** –

PassportElementErrorSelfie

```
class aiogram.types.passport_element_error.PassportElementErrorSelfie(source:
                                                                    aiogram.types.base.String,
                                                                    type:
                                                                    aiogram.types.base.String,
                                                                    file_hash:
                                                                    aiogram.types.base.String,
                                                                    message:
                                                                    aiogram.types.base.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>

Deserialize object

Parameters

- **conf** –
- **kwargs** –

ShippingAddress

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

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: Optional[Dict[str, Any]] = None,
                                                            **kwargs: Any)
```

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: Optional[Dict[str, Any]] = None, **kwargs: Any*)
 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: Optional[Dict[str, Any]] = None, **kwargs: Any*)

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: Optional[Dict[str, Any]] = None, **kwargs: Any*)
 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** –

async set_position_in_set(*position: aiogram.types.base.Integer*) → aiogram.types.base.Boolean
 Use this method to move a sticker in a set created by the bot to a specific position.

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

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

Returns Returns True on success

Return type *base.Boolean*

async delete_from_set() → aiogram.types.base.Boolean

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

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

Returns Returns True on success

Return type base.Boolean

InlineQuery

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

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

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

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

Parameters

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

Returns On success, True is returned

Return type `base.Boolean`

Location

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

Bases: `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: Optional[Dict[str, Any]] = None, **kwargs: Any*)

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>

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InputMediaAnimation

```
class aiogram.types.input_media.InputMediaAnimation(media: aiogram.types.base.InputFile, thumb: Optional[Union[aiogram.types.base.InputFile, aiogram.types.base.String]] = None, caption: Optional[aiogram.types.base.String] = None, width: Optional[aiogram.types.base.Integer] = None, height: Optional[aiogram.types.base.Integer] = None, duration: Optional[aiogram.types.base.Integer] = None, parse_mode: Optional[aiogram.types.base.String] = None, caption_entities: Optional[List[aiogram.types.message_entity.MessageEntity]] = 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>

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InputMediaDocument

```
class aiogram.types.input_media.InputMediaDocument(media: aiogram.types.base.InputFile, thumb: Optional[Union[aiogram.types.base.InputFile, aiogram.types.base.String]] = None, caption: Optional[aiogram.types.base.String] = None, parse_mode: Optional[aiogram.types.base.String] = None, caption_entities: Optional[List[aiogram.types.message_entity.MessageEntity]] = None, disable_content_type_detection: Optional[aiogram.types.base.Boolean] = None, **kwargs)
```

Bases: `aiogram.types.input_media.InputMedia`

Represents a general file to be sent.

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

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InputMediaAudio

```
class aiogram.types.input_media.InputMediaAudio(media: aiogram.types.base.InputFile, thumb:
Optional[Union[aiogram.types.base.InputFile,
aiogram.types.base.String]] = None, caption:
Optional[aiogram.types.base.String] = None,
duration: Optional[aiogram.types.base.Integer] =
None, performer:
Optional[aiogram.types.base.String] = None, title:
Optional[aiogram.types.base.String] = None,
parse_mode: Optional[aiogram.types.base.String] =
None, caption_entities: Op-
tional[List[aiogram.types.message_entity.MessageEntity]]
= None, **kwargs)
```

Bases: *aiogram.types.input_media.InputMedia*

Represents an audio file to be treated as music to be sent.

<https://core.telegram.org/bots/api#inputmediaaudio>

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InputMediaPhoto

```
class aiogram.types.input_media.InputMediaPhoto(media: aiogram.types.base.InputFile, caption:
Optional[aiogram.types.base.String] = None,
parse_mode: Optional[aiogram.types.base.String] =
None, caption_entities: Op-
tional[List[aiogram.types.message_entity.MessageEntity]]
= None, **kwargs)
```

Bases: *aiogram.types.input_media.InputMedia*

Represents a photo to be sent.

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

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InputMediaVideo

```
class aiogram.types.input_media.InputMediaVideo(media: aiogram.types.base.InputFile, thumb:
Optional[Union[aiogram.types.base.InputFile,
aiogram.types.base.String]] = None, caption:
Optional[aiogram.types.base.String] = None, width:
Optional[aiogram.types.base.Integer] = None, height:
Optional[aiogram.types.base.Integer] = None,
duration: Optional[aiogram.types.base.Integer] =
None, parse_mode:
Optional[aiogram.types.base.String] = None,
caption_entities: Op-
tional[List[aiogram.types.message_entity.MessageEntity]]
= None, supports_streaming:
Optional[aiogram.types.base.Boolean] = None,
**kwargs)
```

Bases: *aiogram.types.input_media.InputMedia*

Represents a video to be sent.

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

Deserialize object

Parameters

- **conf** –
- **kwargs** –

MediaGroup

```
class aiogram.types.input_media.MediaGroup(medias: Op-
tional[List[Union[aiogram.types.input_media.InputMedia,
Dict]]] = None)
```

Bases: *aiogram.types.base.TelegramObject*

Helper for sending media group

Deserialize object

Parameters

- **conf** –
- **kwargs** –

attach_many(*medias: Union[aiogram.types.input_media.InputMedia, Dict])

Attach list of media

Parameters **medias** –

attach(media: Union[aiogram.types.input_media.InputMedia, Dict])

Attach media

Parameters **media** –

```
attach_audio(audio: Union[aiogram.types.input_media.InputMediaAudio, aiogram.types.base.InputFile],
             thumb: Optional[Union[aiogram.types.base.InputFile, aiogram.types.base.String]] = None,
             caption: Optional[aiogram.types.base.String] = None, duration:
             Optional[aiogram.types.base.Integer] = None, performer:
             Optional[aiogram.types.base.String] = None, title: Optional[aiogram.types.base.String] =
             None, parse_mode: Optional[aiogram.types.base.String] = None, caption_entities:
             Optional[List[aiogram.types.message_entity.MessageEntity]] = None)
```

Attach audio

Parameters

- **audio** –
- **thumb** –
- **caption** –
- **duration** –
- **performer** –
- **title** –
- **parse_mode** –
- **caption_entities** –

```
attach_document(document: Union[aiogram.types.input_media.InputMediaDocument,
                                aiogram.types.base.InputFile], thumb: Optional[Union[aiogram.types.base.InputFile,
                                aiogram.types.base.String]] = None, caption: Optional[aiogram.types.base.String] =
                                None, parse_mode: Optional[aiogram.types.base.String] = None, caption_entities:
                                Optional[List[aiogram.types.message_entity.MessageEntity]] = None,
                                disable_content_type_detection: Optional[aiogram.types.base.Boolean] = None)
```

Attach document

Parameters

- **document** –
- **caption** –
- **thumb** –
- **parse_mode** –
- **caption_entities** –
- **disable_content_type_detection** –

```
attach_photo(photo: Union[aiogram.types.input_media.InputMediaPhoto, aiogram.types.base.InputFile],
              caption: Optional[aiogram.types.base.String] = None, parse_mode:
              Optional[aiogram.types.base.String] = None, caption_entities:
              Optional[List[aiogram.types.message_entity.MessageEntity]] = None)
```

Attach photo

Parameters

- **photo** –
- **caption** –
- **parse_mode** –
- **caption_entities** –

```
attach_video(video: Union[aiogram.types.input_media.InputMediaVideo, aiogram.types.base.InputFile],
              thumb: Optional[Union[aiogram.types.base.InputFile, aiogram.types.base.String]] = None,
              caption: Optional[aiogram.types.base.String] = None, width:
              Optional[aiogram.types.base.Integer] = None, height: Optional[aiogram.types.base.Integer]
              = None, duration: Optional[aiogram.types.base.Integer] = None, parse_mode:
              Optional[aiogram.types.base.String] = None, caption_entities:
              Optional[List[aiogram.types.message_entity.MessageEntity]] = None, supports_streaming:
              Optional[aiogram.types.base.Boolean] = None)
```

Attach video

Parameters

- **video** –
- **thumb** –
- **caption** –
- **width** –
- **height** –
- **duration** –
- **parse_mode** –
- **caption_entities** –
- **supports_streaming** –

to_python() → List

Get object as JSON serializable

Returns

InlineQueryResult

```
class aiogram.types.inline_query_result.InlineQueryResult(**kwargs)
```

Bases: *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>

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InlineQueryResultArticle

```
class aiogram.types.inline_query_result.InlineQueryResultArticle(*, id:
                                                                    aiogram.types.base.String, title:
                                                                    aiogram.types.base.String,
                                                                    input_message_content:
                                                                    aiogram.types.input_message_content.InputMessageContent,
                                                                    reply_markup: Op-
                                                                    tional[aiogram.types.inline_keyboard.InlineKeyboardMarkup] = None, url: Op-
                                                                    tional[aiogram.types.base.String]
                                                                    = None, hide_url: Op-
                                                                    tional[aiogram.types.base.Boolean]
                                                                    = None, description: Op-
                                                                    tional[aiogram.types.base.String]
                                                                    = None, thumb_url: Op-
                                                                    tional[aiogram.types.base.String]
                                                                    = None, thumb_width: Op-
                                                                    tional[aiogram.types.base.Integer]
                                                                    = None, thumb_height: Op-
                                                                    tional[aiogram.types.base.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>

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InlineQueryResultPhoto

```
class aiogram.types.inline_query_result.InlineQueryResultPhoto(*, id: aiogram.types.base.String,
                                                                photo_url:
                                                                aiogram.types.base.String,
                                                                thumb_url:
                                                                aiogram.types.base.String,
                                                                photo_width: Op-
                                                                tional[aiogram.types.base.Integer]
                                                                = None, photo_height: Op-
                                                                tional[aiogram.types.base.Integer]
                                                                = None, title: Op-
                                                                tional[aiogram.types.base.String]
                                                                = None, description: Op-
                                                                tional[aiogram.types.base.String]
                                                                = None, caption: Op-
                                                                tional[aiogram.types.base.String]
                                                                = None, parse_mode: Op-
                                                                tional[aiogram.types.base.String]
                                                                = None, caption_entities: Op-
                                                                tional[List[aiogram.types.message_entity.MessageEn-
                                                                = None, reply_markup: Op-
                                                                tional[aiogram.types.inline_keyboard.InlineKeyboar-
                                                                = None, input_message_content:
                                                                Op-
                                                                tional[aiogram.types.input_message_content.InputM-
                                                                = 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>

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InlineQueryResultGif

```
class aiogram.types.inline_query_result.InlineQueryResultGif(*, id: aiogram.types.base.String,
    gif_url: aiogram.types.base.String,
    gif_width: Optional[aiogram.types.base.Integer] =
    None, gif_height: Optional[aiogram.types.base.Integer] =
    None, gif_duration: Optional[aiogram.types.base.Integer] =
    None, thumb_url: Optional[aiogram.types.base.String]
    = None, title: Optional[aiogram.types.base.String]
    = None, caption: Optional[aiogram.types.base.String]
    = None, parse_mode: Optional[aiogram.types.base.String]
    = None, reply_markup: Optional[aiogram.types.inline_keyboard.InlineKeyboardM
    = None, caption_entities: Optional[List[aiogram.types.message_entity.MessageEntit
    = None, input_message_content: Optional[aiogram.types.input_message_content.InputMes
    = 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>

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InlineQueryResultMpeg4Gif

```
class aiogram.types.inline_query_result.InlineQueryResultMpeg4Gif(*, id:
    aiogram.types.base.String,
    mpeg4_url:
    aiogram.types.base.String,
    thumb_url:
    aiogram.types.base.String,
    mpeg4_width: Op-
    tional[aiogram.types.base.Integer]
    = None, mpeg4_height: Op-
    tional[aiogram.types.base.Integer]
    = None, mpeg4_duration: Op-
    tional[aiogram.types.base.Integer]
    = None, title: Op-
    tional[aiogram.types.base.String]
    = None, caption: Op-
    tional[aiogram.types.base.String]
    = None, parse_mode: Op-
    tional[aiogram.types.base.String]
    = None, reply_markup: Op-
    tional[aiogram.types.inline_keyboard.InlineKeyb
    = None, caption_entities: Op-
    tional[List[aiogram.types.message_entity.Messa
    = None,
    input_message_content: Op-
    tional[aiogram.types.input_message_content.Inp
    = 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>

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InlineQueryResultVideo

```
class aiogram.types.inline_query_result.InlineQueryResultVideo(*, id: aiogram.types.base.String,
                                                                video_url:
                                                                aiogram.types.base.String,
                                                                mime_type:
                                                                aiogram.types.base.String,
                                                                thumb_url:
                                                                aiogram.types.base.String, title:
                                                                aiogram.types.base.String,
                                                                caption: Op-
                                                                tional[aiogram.types.base.String]
                                                                = None, parse_mode: Op-
                                                                tional[aiogram.types.base.String]
                                                                = None, video_width: Op-
                                                                tional[aiogram.types.base.Integer]
                                                                = None, video_height: Op-
                                                                tional[aiogram.types.base.Integer]
                                                                = None, video_duration: Op-
                                                                tional[aiogram.types.base.Integer]
                                                                = None, description: Op-
                                                                tional[aiogram.types.base.String]
                                                                = None, reply_markup: Op-
                                                                tional[aiogram.types.inline_keyboard.InlineKeyboard
                                                                = None, caption_entities: Op-
                                                                tional[List[aiogram.types.message_entity.MessageEn
                                                                = None, input_message_content:
                                                                Op-
                                                                tional[aiogram.types.input_message_content.InputM
                                                                = 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>

Deserialize object

Parameters

- `conf` –
- `kwargs` –

InlineQueryResultAudio

```
class aiogram.types.inline_query_result.InlineQueryResultAudio(*, id: aiogram.types.base.String,
                                                                audio_url:
                                                                aiogram.types.base.String, title:
                                                                aiogram.types.base.String,
                                                                caption: Op-
                                                                tional[aiogram.types.base.String]
                                                                = None, parse_mode: Op-
                                                                tional[aiogram.types.base.String]
                                                                = None, performer: Op-
                                                                tional[aiogram.types.base.String]
                                                                = None, audio_duration: Op-
                                                                tional[aiogram.types.base.Integer]
                                                                = None, reply_markup: Op-
                                                                tional[aiogram.types.inline_keyboard.InlineKeyboard
                                                                = None, caption_entities: Op-
                                                                tional[List[aiogram.types.message_entity.MessageEn
                                                                = None, input_message_content:
                                                                Op-
                                                                tional[aiogram.types.input_message_content.InputM
                                                                = 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>

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InlineQueryResultVoice

```
class aiogram.types.inline_query_result.InlineQueryResultVoice(*, id: aiogram.types.base.String,
                                                                voice_url:
                                                                aiogram.types.base.String, title:
                                                                aiogram.types.base.String,
                                                                caption: Op-
                                                                tional[aiogram.types.base.String]
                                                                = None, parse_mode: Op-
                                                                tional[aiogram.types.base.String]
                                                                = None, voice_duration: Op-
                                                                tional[aiogram.types.base.Integer]
                                                                = None, reply_markup: Op-
                                                                tional[aiogram.types.inline_keyboard.InlineKeyboard
                                                                = None, caption_entities: Op-
                                                                tional[List[aiogram.types.message_entity.MessageEn
                                                                = None, input_message_content:
                                                                Op-
                                                                tional[aiogram.types.input_message_content.InputM
                                                                = 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>

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InlineQueryResultDocument

```
class aiogram.types.inline_query_result.InlineQueryResultDocument(*, id:
    aiogram.types.base.String,
    title:
    aiogram.types.base.String,
    caption: Op-
    tional[aiogram.types.base.String]
    = None, parse_mode: Op-
    tional[aiogram.types.base.String]
    = None, caption_entities: Op-
    tional[List[aiogram.types.message_entity.MessageEntity]]
    = None, document_url: Op-
    tional[aiogram.types.base.String]
    = None, mime_type: Op-
    tional[aiogram.types.base.String]
    = None, description: Op-
    tional[aiogram.types.base.String]
    = None, reply_markup: Op-
    tional[aiogram.types.inline_keyboard.InlineKeyboardMarkup]
    = None,
    input_message_content: Op-
    tional[aiogram.types.input_message_content.InputMessageContent]
    = None, thumb_url: Op-
    tional[aiogram.types.base.String]
    = None, thumb_width: Op-
    tional[aiogram.types.base.Integer]
    = None, thumb_height: Op-
    tional[aiogram.types.base.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>

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InlineQueryResultLocation

```
class aiogram.types.inline_query_result.InlineQueryResultLocation(*, id:
    aiogram.types.base.String,
    latitude:
    aiogram.types.base.Float,
    longitude:
    aiogram.types.base.Float,
    title:
    aiogram.types.base.String,
    horizontal_accuracy: Optional[aiogram.types.base.Float]
    = None, live_period: Optional[aiogram.types.base.Integer]
    = None, heading: Optional[aiogram.types.base.Integer]
    = None,
    proximity_alert_radius: Optional[aiogram.types.base.Integer]
    = None, reply_markup: Optional[aiogram.types.inline_keyboard.InlineKeyb
    = None,
    input_message_content: Optional[aiogram.types.input_message_content.Inp
    = None, thumb_url: Optional[aiogram.types.base.String]
    = None, thumb_width: Optional[aiogram.types.base.Integer]
    = None, thumb_height: Optional[aiogram.types.base.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.

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

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InlineQueryResultVenue

```
class aiogram.types.inline_query_result.InlineQueryResultVenue(*, id: aiogram.types.base.String,
    latitude:
    aiogram.types.base.Float,
    longitude:
    aiogram.types.base.Float, title:
    aiogram.types.base.String,
    address:
    aiogram.types.base.String,
    foursquare_id: Op-
    tional[aiogram.types.base.String]
    = None, foursquare_type: Op-
    tional[aiogram.types.base.String]
    = None, google_place_id: Op-
    tional[aiogram.types.base.String]
    = None, google_place_type: Op-
    tional[aiogram.types.base.String]
    = None, reply_markup: Op-
    tional[aiogram.types.inline_keyboard.InlineKeyboard
    = None, input_message_content:
    Op-
    tional[aiogram.types.input_message_content.InputM
    = None, thumb_url: Op-
    tional[aiogram.types.base.String]
    = None, thumb_width: Op-
    tional[aiogram.types.base.Integer]
    = None, thumb_height: Op-
    tional[aiogram.types.base.Integer]
    = 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>

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InlineQueryResultContact

```
class aiogram.types.inline_query_result.InlineQueryResultContact(*, id:
    aiogram.types.base.String,
    phone_number:
    aiogram.types.base.String,
    first_name:
    aiogram.types.base.String,
    last_name: Op-
    tional[aiogram.types.base.String]
    = None, reply_markup: Op-
    tional[aiogram.types.inline_keyboard.InlineKeybo
    = None,
    input_message_content: Op-
    tional[aiogram.types.input_message_content.Inpu
    = None, thumb_url: Op-
    tional[aiogram.types.base.String]
    = None, thumb_width: Op-
    tional[aiogram.types.base.Integer]
    = None, thumb_height: Op-
    tional[aiogram.types.base.Integer]
    = None, foursquare_type: Op-
    tional[aiogram.types.base.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>

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InlineQueryResultGame

```
class aiogram.types.inline_query_result.InlineQueryResultGame(*, id: aiogram.types.base.String,
    game_short_name:
    aiogram.types.base.String,
    reply_markup: Op-
    tional[aiogram.types.inline_keyboard.InlineKeyboard
    = 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>

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InlineQueryResultCachedPhoto

```
class aiogram.types.inline_query_result.InlineQueryResultCachedPhoto(*, id:
    aiogram.types.base.String,
    photo_file_id:
    aiogram.types.base.String,
    title: Op-
    tional[aiogram.types.base.String]
    = None, description: Op-
    tional[aiogram.types.base.String]
    = None, caption: Op-
    tional[aiogram.types.base.String]
    = None, parse_mode: Op-
    tional[aiogram.types.base.String]
    = None, caption_entities:
    Op-
    tional[List[aiogram.types.message_entity.Me
    = None, reply_markup:
    Op-
    tional[aiogram.types.inline_keyboard.InlineK
    = None,
    input_message_content:
    Op-
    tional[aiogram.types.input_message_content
    = 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>

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InlineQueryResultCachedGif

```
class aiogram.types.inline_query_result.InlineQueryResultCachedGif(*, id:
    aiogram.types.base.String,
    gif_file_id:
    aiogram.types.base.String,
    title: Op-
    tional[aiogram.types.base.String]
    = None, caption: Op-
    tional[aiogram.types.base.String]
    = None, parse_mode: Op-
    tional[aiogram.types.base.String]
    = None, caption_entities:
    Op-
    tional[List[aiogram.types.message_entity.MessageEntity]]
    = None, reply_markup: Op-
    tional[aiogram.types.inline_keyboard.InlineKeyboardMarkup]
    = None,
    input_message_content: Op-
    tional[aiogram.types.input_message_content.InputMessageContent]
    = 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>

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InlineQueryResultCachedMpeg4Gif

```
class aiogram.types.inline_query_result.InlineQueryResultCachedMpeg4Gif(*, id:
    aiogram.types.base.String,
    mpeg4_file_id:
    aiogram.types.base.String,
    title: Op-
    tional[aiogram.types.base.String]
    = None, caption: Op-
    tional[aiogram.types.base.String]
    = None, parse_mode:
    Op-
    tional[aiogram.types.base.String]
    = None,
    caption_entities: Op-
    tional[List[aiogram.types.message_entity]
    = None,
    reply_markup: Op-
    tional[aiogram.types.inline_keyboard.Inlin
    = None, in-
    put_message_content:
    Op-
    tional[aiogram.types.input_message_con
    = 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>

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InlineQueryResultCachedSticker

```
class aiogram.types.inline_query_result.InlineQueryResultCachedSticker(*, id:
    aiogram.types.base.String,
    sticker_file_id:
    aiogram.types.base.String,
    reply_markup: Op-
    tional[aiogram.types.inline_keyboard.Inlin
    = None,
    input_message_content:
    Op-
    tional[aiogram.types.input_message_cont
    = 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>

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InlineQueryResultCachedDocument

```
class aiogram.types.inline_query_result.InlineQueryResultCachedDocument(*, id:
    aiogram.types.base.String,
    title:
    aiogram.types.base.String,
    document_file_id:
    aiogram.types.base.String,
    description: Op-
    tional[aiogram.types.base.String]
    = None, caption: Op-
    tional[aiogram.types.base.String]
    = None, parse_mode:
    Op-
    tional[aiogram.types.base.String]
    = None,
    caption_entities: Op-
    tional[List[aiogram.types.message_entity]
    = None,
    reply_markup: Op-
    tional[aiogram.types.inline_keyboard.Inl
    = None, in-
    put_message_content:
    Op-
    tional[aiogram.types.input_message_con
    = 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>

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InlineQueryResultCachedVideo

```
class aiogram.types.inline_query_result.InlineQueryResultCachedVideo(*, id:
    aiogram.types.base.String,
    video_file_id:
    aiogram.types.base.String,
    title:
    aiogram.types.base.String,
    description: Op-
    tional[aiogram.types.base.String]
    = None, caption: Op-
    tional[aiogram.types.base.String]
    = None, parse_mode: Op-
    tional[aiogram.types.base.String]
    = None, caption_entities:
    Op-
    tional[List[aiogram.types.message_entity.Me
    = None, reply_markup:
    Op-
    tional[aiogram.types.inline_keyboard.InlineK
    = None,
    input_message_content:
    Op-
    tional[aiogram.types.input_message_content
    = 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>

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InlineQueryResultCachedVoice

```
class aiogram.types.inline_query_result.InlineQueryResultCachedVoice(*, id:
    aiogram.types.base.String,
    voice_file_id:
    aiogram.types.base.String,
    title:
    aiogram.types.base.String,
    caption: Op-
    tional[aiogram.types.base.String]
    = None, parse_mode: Op-
    tional[aiogram.types.base.String]
    = None, caption_entities:
    Op-
    tional[List[aiogram.types.message_entity.Me
    = None, reply_markup:
    Op-
    tional[aiogram.types.inline_keyboard.InlineK
    = None,
    input_message_content:
    Op-
    tional[aiogram.types.input_message_content
    = 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>

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InlineQueryResultCachedAudio

```
class aiogram.types.inline_query_result.InlineQueryResultCachedAudio(*, id:
    aiogram.types.base.String,
    audio_file_id:
    aiogram.types.base.String,
    caption: Op-
    tional[aiogram.types.base.String]
    = None, parse_mode: Op-
    tional[aiogram.types.base.String]
    = None, caption_entities:
    Op-
    tional[List[aiogram.types.message_entity.Me
    = None, reply_markup:
    Op-
    tional[aiogram.types.inline_keyboard.InlineK
    = None,
    input_message_content:
    Op-
    tional[aiogram.types.input_message_content
    = 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>

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InputFile

```
class aiogram.types.input_file.InputFile(path_or_bytesio: Union[str, io.IOBase, pathlib.Path,
    aiogram.types.input_file._WebPipe], 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

PreCheckoutQuery

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

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: Optional[Dict[str, Any]] = None, **kwargs: Any)
    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: Optional[Dict[str, Any]] =
    None, **kwargs: Any)
```

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:
    aiogram.types.base.String,
    first_name: Optional[aiogram.types.base.String]
    = None, last_name: Optional[aiogram.types.base.String]
    = None, vcard: Optional[aiogram.types.base.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>

Deserialize object

Parameters

- **conf** –

- **kwargs** –

InputLocationMessageContent

```
class aiogram.types.input_message_content.InputLocationMessageContent(latitude:
                                                                    aiogram.types.base.Float,
                                                                    longitude:
                                                                    aiogram.types.base.Float,
                                                                    horizontal_accuracy:
                                                                    Op-
                                                                    tional[aiogram.types.base.Float]
                                                                    = None, live_period: Op-
                                                                    tional[aiogram.types.base.Integer]
                                                                    = None, heading: Op-
                                                                    tional[aiogram.types.base.Integer]
                                                                    = None,
                                                                    proximity_alert_radius:
                                                                    Op-
                                                                    tional[aiogram.types.base.Integer]
                                                                    = None)
```

Bases: *aiogram.types.input_message_content.InputMessageContent*

Represents the content of a location message to be sent as the result of an inline query.

<https://core.telegram.org/bots/api#inputlocationmessagecontent>

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InputTextMessageContent

```
class aiogram.types.input_message_content.InputTextMessageContent(message_text:
                                                                    aiogram.types.base.String,
                                                                    parse_mode: Op-
                                                                    tional[aiogram.types.base.String]
                                                                    = None, caption_entities: Op-
                                                                    tional[List[aiogram.types.message_entity.MessageEntity]]
                                                                    = None,
                                                                    disable_web_page_preview:
                                                                    Op-
                                                                    tional[aiogram.types.base.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>

Deserialize object

Parameters

- **conf** –
- **kwargs** –

InputVenueMessageContent

```
class aiogram.types.input_message_content.InputVenueMessageContent(latitude:
                                                                    aiogram.types.base.Float,
                                                                    longitude:
                                                                    aiogram.types.base.Float,
                                                                    title:
                                                                    aiogram.types.base.String,
                                                                    address:
                                                                    aiogram.types.base.String,
                                                                    foursquare_id: Op-
                                                                    tional[aiogram.types.base.String]
                                                                    = None, foursquare_type:
                                                                    Op-
                                                                    tional[aiogram.types.base.String]
                                                                    = None, google_place_id:
                                                                    Op-
                                                                    tional[aiogram.types.base.String]
                                                                    = None, google_place_type:
                                                                    Op-
                                                                    tional[aiogram.types.base.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>

Deserialize object

Parameters

- **conf** –
- **kwargs** –

Update

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

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: Optional[Dict[str, Any]] = None, **kwargs: Any*)

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: Optional[Dict[str, Any]] = None, **kwargs: Any*)

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: Optional[Dict[str, Any]] = None, **kwargs: Any*)

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: Optional[Dict[str, Any]] = None, **kwargs: Any*)
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: Optional[Dict[str, Any]] = None, **kwargs: Any*)
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: Optional[Dict[str, Any]] = None, **kwargs: Any*)

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

This object contains information about one member of a chat. Currently, the following 6 types of chat members are supported:

ChatMemberOwner ChatMemberAdministrator ChatMemberMember ChatMemberRestricted ChatMemberLeft ChatMemberBanned

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

Deserialize object

Parameters

- **conf** –
- **kwargs** –

ChatMemberStatus

class aiogram.types.chat_member.**ChatMemberStatus**
 Bases: [aiogram.utils.helper.Helper](#)

Chat member status

ShippingOption

class aiogram.types.shipping_option.**ShippingOption**(*id: aiogram.types.base.String, title: aiogram.types.base.String, prices: Optional[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>

Deserialize object

Parameters

- **conf** –
- **kwargs** –

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

Parameters **price** –

Returns

ChatPhoto

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

This object represents a chat photo.

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

Deserialize object

Parameters

- **conf** –
- **kwargs** –

async **download_small**(*destination=None, timeout=30, chunk_size=65536, seek=True, make_dirs=True*)

Download file

Parameters

- **destination** – filename or instance of `io.IOBase`. For e. g. `io.BytesIO`
- **timeout** – Integer
- **chunk_size** – Integer
- **seek** – Boolean - go to start of file when downloading is finished.
- **make_dirs** – Make dirs if not exist

Returns destination

async **download_big**(*destination=None, timeout=30, chunk_size=65536, seek=True, make_dirs=True*)

Download file

Parameters

- **destination** – filename or instance of `io.IOBase`. For e. g. `io.BytesIO`
- **timeout** – Integer
- **chunk_size** – Integer
- **seek** – Boolean - go to start of file when downloading is finished.
- **make_dirs** – Make dirs if not exist

Returns destination

Contact

class aiogram.types.contact.**Contact**(*conf: Optional[Dict[str, Any]] = None, **kwargs: Any*)
Bases: [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: Optional[Dict[str, Any]] = None, **kwargs: Any*)

Bases: [aiogram.types.base.TelegramObject](#)

This object represents a message.

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

Deserialize object

Parameters

- **conf** –
- **kwargs** –

is_forward() → bool

Check that the message is forwarded. Only *forward_date* is required to be in forwarded message.

Returns bool

is_command() → bool

Check message text is command

Returns bool

get_full_command() → Optional[Tuple[str, str]]

Split command and args

Returns tuple of (command, args)

get_command(*pure=False*) → Optional[str]

Get command from message

Returns

get_args() → Optional[str]

Get arguments

Returns

parse_entities(*as_html=True*) → str

Text or caption formatted as HTML or Markdown.

Returns str

property md_text: str

Text or caption formatted as markdown.

Returns str

property html_text: str

Text or caption formatted as HTML

Returns str

property url: str

Get URL for the message

Returns str

link(*text, as_html=True*) → str

Generate URL for using in text messages with HTML or MD parse mode

Parameters

- **text** – link label
- **as_html** – generate as HTML

Returns str

```
async answer(text: aiogram.types.base.String, parse_mode: Optional[aiogram.types.base.String] = None,  
             entities: Optional[List[aiogram.types.message_entity.MessageEntity]] = None,  
             disable_web_page_preview: Optional[aiogram.types.base.Boolean] = None,  
             disable_notification: Optional[aiogram.types.base.Boolean] = None,  
             allow_sending_without_reply: Optional[aiogram.types.base.Boolean] = None,  
             reply_markup: Optional[Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup,  
aiogram.types.reply_keyboard.ReplyKeyboardMarkup,  
aiogram.types.reply_keyboard.ReplyKeyboardRemove,  
aiogram.types.force_reply.ForceReply]] = None, reply: aiogram.types.base.Boolean = False)  
    → aiogram.types.message.Message
```

Answer to this message

Parameters

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

Returns On success, the sent Message is returned

Return type types.Message

```

async answer_photo(photo: Union[aiogram.types.base.InputFile, aiogram.types.base.String], caption:
    Optional[aiogram.types.base.String] = None, parse_mode:
    Optional[aiogram.types.base.String] = None, caption_entities:
    Optional[List[aiogram.types.message_entity.MessageEntity]] = None,
    disable_notification: Optional[aiogram.types.base.Boolean] = None,
    allow_sending_without_reply: Optional[aiogram.types.base.Boolean] = None,
    reply_markup:
    Optional[Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup,
    aiogram.types.reply_keyboard.ReplyKeyboardMarkup,
    aiogram.types.reply_keyboard.ReplyKeyboardRemove,
    aiogram.types.force_reply.ForceReply]] = None, reply: aiogram.types.base.Boolean
    = False) → 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.Optional[base.String]) – Photo caption (may also be used when resending photos by file_id), 0-1024 characters
- **parse_mode** (typing.Optional[base.String]) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.
- **caption_entities** (typing.Optional[typing.List[MessageEntity]]) – List of special entities that appear in message text, which can be specified instead of parse_mode
- **disable_notification** (typing.Optional[base.Boolean]) – Sends the message silently. Users will receive a notification with no sound
- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **reply** (base.Boolean) – fill 'reply_to_message_id'

Returns On success, the sent Message is returned

Return type types.Message

```
async answer_audio(audio: Union[aiogram.types.base.InputFile, aiogram.types.base.String], caption:
    Optional[aiogram.types.base.String] = None, parse_mode:
    Optional[aiogram.types.base.String] = None, caption_entities:
    Optional[List[aiogram.types.message_entity.MessageEntity]] = None, duration:
    Optional[aiogram.types.base.Integer] = None, performer:
    Optional[aiogram.types.base.String] = None, title:
    Optional[aiogram.types.base.String] = None, thumb:
    Optional[Union[aiogram.types.base.InputFile, aiogram.types.base.String]] = None,
    disable_notification: Optional[aiogram.types.base.Boolean] = None,
    allow_sending_without_reply: Optional[aiogram.types.base.Boolean] = None,
    reply_markup:
    Optional[Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup,
    aiogram.types.reply_keyboard.ReplyKeyboardMarkup,
    aiogram.types.reply_keyboard.ReplyKeyboardRemove,
    aiogram.types.force_reply.ForceReply]] = None, reply: aiogram.types.base.Boolean
    = False) → 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.Optional[base.String]) – Audio caption, 0-1024 characters after entities parsing
- **parse_mode** (typing.Optional[base.String]) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.
- **caption_entities** (typing.Optional[typing.List[MessageEntity]]) – List of special entities that appear in message text, which can be specified instead of `parse_mode`
- **duration** (typing.Optional[base.Integer]) – Duration of the audio in seconds
- **performer** (typing.Optional[base.String]) – Performer
- **title** (typing.Optional[base.String]) – Track name
- **thumb** (typing.Union[typing.Union[base.InputFile, base.String], None]) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320.
- **disable_notification** (typing.Optional[base.Boolean]) – Sends the message silently. Users will receive a notification with no sound.
- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **reply** (base.Boolean) – fill 'reply_to_message_id'

Returns On success, the sent Message is returned.

Return type `types.Message`

```
async answer_animation(animation: Union[aiogram.types.base.InputFile, aiogram.types.base.String],
                        duration: Optional[aiogram.types.base.Integer] = None, width:
                        Optional[aiogram.types.base.Integer] = None, height:
                        Optional[aiogram.types.base.Integer] = None, thumb:
                        Optional[Union[aiogram.types.base.InputFile, aiogram.types.base.String]] =
                        None, caption: Optional[aiogram.types.base.String] = None, parse_mode:
                        Optional[aiogram.types.base.String] = None, caption_entities:
                        Optional[List[aiogram.types.message_entity.MessageEntity]] = None,
                        disable_notification: Optional[aiogram.types.base.Boolean] = None,
                        allow_sending_without_reply: Optional[aiogram.types.base.Boolean] = None,
                        reply_markup:
                        Optional[Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup,
                        aiogram.types.reply_keyboard.ReplyKeyboardMarkup,
                        aiogram.types.reply_keyboard.ReplyKeyboardRemove,
                        aiogram.types.force_reply.ForceReply]] = None, reply:
                        aiogram.types.base.Boolean = False) → 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.Optional[base.Integer]) – Duration of sent animation in seconds
- **width** (typing.Optional[base.Integer]) – Animation width
- **height** (typing.Optional[base.Integer]) – Animation height
- **thumb** (typing.Union[typing.Union[base.InputFile, base.String], None]) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320.
- **caption** (typing.Optional[base.String]) – Animation caption (may also be used when resending animation by `file_id`), 0-1024 characters
- **parse_mode** (typing.Optional[base.String]) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption
- **caption_entities** (typing.Optional[typing.List[MessageEntity]]) – List of special entities that appear in message text, which can be specified instead of `parse_mode`
- **disable_notification** (typing.Optional[base.Boolean]) – Sends the message silently. Users will receive a notification with no sound
- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (typing.Union[typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.

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

- **reply** (base.Boolean) – fill 'reply_to_message_id'

Returns On success, the sent Message is returned

Return type types.Message

```
async answer_document(document: Union[aiogram.types.base.InputFile, aiogram.types.base.String],
                      thumb: Optional[Union[aiogram.types.base.InputFile,
                      aiogram.types.base.String]] = None, caption:
                      Optional[aiogram.types.base.String] = None, parse_mode:
                      Optional[aiogram.types.base.String] = None, caption_entities:
                      Optional[List[aiogram.types.message_entity.MessageEntity]] = None,
                      disable_content_type_detection: Optional[aiogram.types.base.Boolean] = None,
                      disable_notification: Optional[aiogram.types.base.Boolean] = None,
                      allow_sending_without_reply: Optional[aiogram.types.base.Boolean] = None,
                      reply_markup:
                      Optional[Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup,
                      aiogram.types.reply_keyboard.ReplyKeyboardMarkup,
                      aiogram.types.reply_keyboard.ReplyKeyboardRemove,
                      aiogram.types.force_reply.ForceReply]] = None, reply:
                      aiogram.types.base.Boolean = False) → aiogram.types.message.Message
```

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

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

Parameters

- **document** (typing.Union[base.InputFile, base.String]) – File to send
- **thumb** (typing.Union[base.InputFile, base.String, None]) – Thumbnail of the file sent
- **caption** (typing.Optional[base.String]) – Document caption (may also be used when resending documents by file_id), 0-1024 characters
- **disable_content_type_detection** (typing.Optional[base.Boolean]) – Disables automatic server-side content type detection for files uploaded using multipart/form-data
- **parse_mode** (typing.Optional[base.String]) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.
- **caption_entities** (typing.Optional[typing.List[MessageEntity]]) – List of special entities that appear in message text, which can be specified instead of parse_mode
- **disable_notification** (typing.Optional[base.Boolean]) – Sends the message silently. Users will receive a notification with no sound
- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]) – Additional interface options. A JSON-serialized object for an inline keyboard,

custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user

- **reply** (`typing.Optional[base.Boolean]`) – True if the message is a reply

Returns On success, the sent Message is returned

Return type `types.Message`

```
async answer_video(video: Union[aiogram.types.base.InputFile, aiogram.types.base.String], duration:
    Optional[aiogram.types.base.Integer] = None, width:
    Optional[aiogram.types.base.Integer] = None, height:
    Optional[aiogram.types.base.Integer] = None, thumb:
    Optional[Union[aiogram.types.base.InputFile, aiogram.types.base.String]] = None,
    caption: Optional[aiogram.types.base.String] = None, parse_mode:
    Optional[aiogram.types.base.String] = None, caption_entities:
    Optional[List[aiogram.types.message_entity.MessageEntity]] = None,
    supports_streaming: Optional[aiogram.types.base.Boolean] = None,
    disable_notification: Optional[aiogram.types.base.Boolean] = None,
    allow_sending_without_reply: Optional[aiogram.types.base.Boolean] = None,
    reply_markup:
    Optional[Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup,
aiogram.types.reply_keyboard.ReplyKeyboardMarkup,
aiogram.types.reply_keyboard.ReplyKeyboardRemove,
aiogram.types.force_reply.ForceReply]] = None, reply: aiogram.types.base.Boolean
    = False) → 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.Optional[base.Integer]`) – Duration of sent video in seconds
- **width** (`typing.Optional[base.Integer]`) – Video width
- **height** (`typing.Optional[base.Integer]`) – Video height
- **thumb** (`typing.Union[base.InputFile, base.String, None]`) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320.
- **caption** (`typing.Optional[base.String]`) – Video caption (may also be used when resending videos by `file_id`), 0-1024 characters after entities parsing
- **parse_mode** (`typing.Optional[base.String]`) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption
- **caption_entities** (`typing.Optional[typing.List[MessageEntity]]`) – List of special entities that appear in message text, which can be specified instead of `parse_mode`
- **supports_streaming** (`typing.Optional[base.Boolean]`) – Pass True, if the uploaded video is suitable for streaming
- **disable_notification** (`typing.Optional[base.Boolean]`) – Sends the message silently. Users will receive a notification with no sound.

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

Returns On success, the sent Message is returned.

Return type `types.Message`

```
async answer_voice(voice: Union[aiogram.types.base.InputFile, aiogram.types.base.String], caption:
    Optional[aiogram.types.base.String] = None, parse_mode:
    Optional[aiogram.types.base.String] = None, caption_entities:
    Optional[List[aiogram.types.message_entity.MessageEntity]] = None, duration:
    Optional[aiogram.types.base.Integer] = None, disable_notification:
    Optional[aiogram.types.base.Boolean] = None, allow_sending_without_reply:
    Optional[aiogram.types.base.Boolean] = None, reply_markup:
    Optional[Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup,
aiogram.types.reply_keyboard.ReplyKeyboardMarkup,
aiogram.types.reply_keyboard.ReplyKeyboardRemove,
aiogram.types.force_reply.ForceReply]] = None, reply: aiogram.types.base.Boolean
    = False) → 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.Optional[base.String]`) – Voice message caption, 0-1024 characters after entities parsing
- **parse_mode** (`typing.Optional[base.String]`) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption
- **caption_entities** (`typing.Optional[typing.List[MessageEntity]]`) – List of special entities that appear in message text, which can be specified instead of parse_mode
- **duration** (`typing.Optional[base.Integer]`) – Duration of the voice message in seconds
- **disable_notification** (`typing.Optional[base.Boolean]`) – Sends the message silently. Users will receive a notification with no sound.
- **allow_sending_without_reply** (`typing.Optional[base.Boolean]`) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]`) – Additional interface options. A JSON-serialized object for an inline keyboard,

custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user

- **reply** (`base.Boolean`) – fill 'reply_to_message_id'

Returns On success, the sent Message is returned.

Return type `types.Message`

```
async answer_video_note(video_note: Union[aiogram.types.base.InputFile, aiogram.types.base.String],
                        duration: Optional[aiogram.types.base.Integer] = None, length:
                        Optional[aiogram.types.base.Integer] = None, thumb:
                        Optional[Union[aiogram.types.base.InputFile, aiogram.types.base.String]] =
                        None, disable_notification: Optional[aiogram.types.base.Boolean] = None,
                        allow_sending_without_reply: Optional[aiogram.types.base.Boolean] = None,
                        reply_markup:
                        Optional[Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup,
                        aiogram.types.reply_keyboard.ReplyKeyboardMarkup,
                        aiogram.types.reply_keyboard.ReplyKeyboardRemove,
                        aiogram.types.force_reply.ForceReply]] = None, reply:
                        aiogram.types.base.Boolean = False) → 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.Optional[base.Integer]`) – Duration of sent video in seconds
- **length** (`typing.Optional[base.Integer]`) – Video width and height
- **thumb** (`typing.Union[typing.Union[base.InputFile, base.String], None]`) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320.
- **disable_notification** (`typing.Optional[base.Boolean]`) – Sends the message silently. Users will receive a notification with no sound.
- **allow_sending_without_reply** (`typing.Optional[base.Boolean]`) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]`) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **reply** (`base.Boolean`) – fill 'reply_to_message_id'

Returns On success, the sent Message is returned.

Return type `types.Message`

```
async answer_media_group(media: Union[aiogram.types.input_media.MediaGroup, List],
                          disable_notification: Optional[aiogram.types.base.Boolean] = None,
                          allow_sending_without_reply: Optional[aiogram.types.base.Boolean] =
                          None, reply: aiogram.types.base.Boolean = False) →
                          List[aiogram.types.message.Message]
```

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

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

Parameters

- **media** (typing.Union[types.MediaGroup, typing.List]) – A JSON-serialized array describing photos and videos to be sent
- **disable_notification** (typing.Optional[base.Boolean]) – Sends the message silently. Users will receive a notification with no sound.
- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply** (base.Boolean) – fill 'reply_to_message_id'

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

Return type typing.List[types.Message]

```
async answer_location(latitude: aiogram.types.base.Float, longitude: aiogram.types.base.Float,
                      live_period: Optional[aiogram.types.base.Integer] = None, disable_notification:
                      Optional[aiogram.types.base.Boolean] = None, allow_sending_without_reply:
                      Optional[aiogram.types.base.Boolean] = None, horizontal_accuracy:
                      Optional[aiogram.types.base.Float] = None, heading:
                      Optional[aiogram.types.base.Integer] = None, proximity_alert_radius:
                      Optional[aiogram.types.base.Integer] = None, reply_markup:
                      Optional[Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup,
                      aiogram.types.reply_keyboard.ReplyKeyboardMarkup,
                      aiogram.types.reply_keyboard.ReplyKeyboardRemove,
                      aiogram.types.force_reply.ForceReply]] = None, reply:
                      aiogram.types.base.Boolean = False) → 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
- **horizontal_accuracy** (typing.Optional[base.Float]) – The radius of uncertainty for the location, measured in meters; 0-1500
- **live_period** (typing.Optional[base.Integer]) – Period in seconds for which the location will be updated
- **heading** (typing.Optional[base.Integer]) – For live locations, a direction in which the user is moving, in degrees. Must be between 1 and 360 if specified.
- **proximity_alert_radius** (typing.Optional[base.Integer]) – For live locations, a maximum distance for proximity alerts about approaching another chat member, in meters. Must be between 1 and 100000 if specified.

- **disable_notification** (`typing.Optional[base.Boolean]`) – Sends the message silently. Users will receive a notification with no sound.
- **allow_sending_without_reply** (`typing.Optional[base.Boolean]`) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]`) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **reply** (`base.Boolean`) – fill 'reply_to_message_id'

Returns On success, the sent Message is returned.

Return type `types.Message`

```
async answer_venue(latitude: aiogram.types.base.Float, longitude: aiogram.types.base.Float, title:
    aiogram.types.base.String, address: aiogram.types.base.String, foursquare_id:
    Optional[aiogram.types.base.String] = None, foursquare_type:
    Optional[aiogram.types.base.String] = None, google_place_id:
    Optional[aiogram.types.base.String] = None, google_place_type:
    Optional[aiogram.types.base.String] = None, disable_notification:
    Optional[aiogram.types.base.Boolean] = None, allow_sending_without_reply:
    Optional[aiogram.types.base.Boolean] = None, reply_markup:
    Optional[Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup,
    aiogram.types.reply_keyboard.ReplyKeyboardMarkup,
    aiogram.types.reply_keyboard.ReplyKeyboardRemove,
    aiogram.types.force_reply.ForceReply]] = None, reply: aiogram.types.base.Boolean
    = False) → 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.Optional[base.String]`) – Foursquare identifier of the venue
- **foursquare_type** (`typing.Optional[base.String]`) – Foursquare type of the venue, if known
- **google_place_id** (`typing.Optional[base.String]`) – Google Places identifier of the venue
- **google_place_type** (`typing.Optional[base.String]`) – Google Places type of the venue. See supported types: https://developers.google.com/places/web-service/supported_types
- **disable_notification** (`typing.Optional[base.Boolean]`) – Sends the message silently. Users will receive a notification with no sound
- **allow_sending_without_reply** (`typing.Optional[base.Boolean]`) – Pass True, if the message should be sent even if the specified replied-to message is not found

- **reply_markup** (typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **reply** (base.Boolean) – fill 'reply_to_message_id'

Returns On success, the sent Message is returned.

Return type types.Message

```
async answer_contact(phone_number: aiogram.types.base.String, first_name: aiogram.types.base.String,
                    last_name: Optional[aiogram.types.base.String] = None, disable_notification:
                    Optional[aiogram.types.base.Boolean] = None, allow_sending_without_reply:
                    Optional[aiogram.types.base.Boolean] = None, reply_markup:
                    Optional[Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup,
                    aiogram.types.reply_keyboard.ReplyKeyboardMarkup,
                    aiogram.types.reply_keyboard.ReplyKeyboardRemove,
                    aiogram.types.force_reply.ForceReply]] = None, reply:
                    aiogram.types.base.Boolean = False) → 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.Optional[base.String]) – Contact's last name
- **disable_notification** (typing.Optional[base.Boolean]) – Sends the message silently. Users will receive a notification with no sound.
- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **reply** (base.Boolean) – fill 'reply_to_message_id'

Returns On success, the sent Message is returned.

Return type types.Message

```
async answer_sticker(sticker: Union[aiogram.types.base.InputFile, aiogram.types.base.String],
                   disable_notification: Optional[aiogram.types.base.Boolean] = None,
                   allow_sending_without_reply: Optional[aiogram.types.base.Boolean] = None,
                   reply_markup:
                   Optional[Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup,
                   aiogram.types.reply_keyboard.ReplyKeyboardMarkup,
                   aiogram.types.reply_keyboard.ReplyKeyboardRemove,
                   aiogram.types.force_reply.ForceReply]] = None, reply:
                   aiogram.types.base.Boolean = False) → 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.Optional[base.Boolean]`) – Sends the message silently. Users will receive a notification with no sound.
- **allow_sending_without_reply** (`typing.Optional[base.Boolean]`) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]`) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **reply** (`base.Boolean`) – fill 'reply_to_message_id'

Returns On success, the sent Message is returned.

Return type `types.Message`

```
async answer_poll(question: aiogram.types.base.String, options: List[aiogram.types.base.String],
                  is_anonymous: Optional[aiogram.types.base.Boolean] = None, type:
                  Optional[aiogram.types.base.String] = None, allows_multiple_answers:
                  Optional[aiogram.types.base.Boolean] = None, correct_option_id:
                  Optional[aiogram.types.base.Integer] = None, explanation:
                  Optional[aiogram.types.base.String] = None, explanation_parse_mode:
                  Optional[aiogram.types.base.String] = None, explanation_entities:
                  Optional[List[aiogram.types.message_entity.MessageEntity]] = None, open_period:
                  Optional[aiogram.types.base.Integer] = None, close_date:
                  Optional[Union[aiogram.types.base.Integer, datetime.datetime, datetime.timedelta]] =
                  None, is_closed: Optional[aiogram.types.base.Boolean] = None, disable_notification:
                  Optional[aiogram.types.base.Boolean] = None, allow_sending_without_reply:
                  Optional[aiogram.types.base.Boolean] = None, reply_markup:
                  Optional[Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup,
                  aiogram.types.reply_keyboard.ReplyKeyboardMarkup,
                  aiogram.types.reply_keyboard.ReplyKeyboardRemove,
                  aiogram.types.force_reply.ForceReply]] = None, reply: aiogram.types.base.Boolean =
                  False) → aiogram.types.message.Message
```

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

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

Parameters

- **question** (`base.String`) – Poll question, 1-255 characters
- **options** (`typing.List[base.String]`) – List of answer options, 2-10 strings 1-100 characters each
- **is_anonymous** (`typing.Optional[base.Boolean]`) – True, if the poll needs to be anonymous, defaults to True
- **type** (`typing.Optional[base.String]`) – Poll type, “quiz” or “regular”, defaults to “regular”
- **allows_multiple_answers** (`typing.Optional[base.Boolean]`) – True, if the poll allows multiple answers, ignored for polls in quiz mode, defaults to False

- **correct_option_id** (`typing.Optional[base.Integer]`) – 0-based identifier of the correct answer option, required for polls in quiz mode
- **explanation** (`typing.Optional[base.String]`) – Text that is shown when a user chooses an incorrect answer or taps on the lamp icon in a quiz-style poll, 0-200 characters with at most 2 line feeds after entities parsing
- **explanation_parse_mode** (`typing.Optional[base.String]`) – Mode for parsing entities in the explanation. See formatting options for more details.
- **explanation_entities** (`typing.Optional[typing.List[MessageEntity]]`) – List of special entities that appear in message text, which can be specified instead of `parse_mode`
- **open_period** (`typing.Optional[base.Integer]`) – Amount of time in seconds the poll will be active after creation, 5-600. Can't be used together with `close_date`.
- **close_date** (`typing.Union[base.Integer, datetime.datetime, datetime.timedelta, None]`) – Point in time (Unix timestamp) when the poll will be automatically closed. Must be at least 5 and no more than 600 seconds in the future. Can't be used together with `open_period`.
- **is_closed** (`typing.Optional[base.Boolean]`) – Pass True, if the poll needs to be immediately closed
- **disable_notification** (`typing.Optional[base.Boolean]`) – Sends the message silently. Users will receive a notification with no sound.
- **allow_sending_without_reply** (`typing.Optional[base.Boolean]`) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]`) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **reply** (`base.Boolean`) – fill 'reply_to_message_id'

Returns On success, the sent Message is returned

Return type `types.Message`

```
async def answer_dice(emoji: Optional[aiogram.types.base.String] = None, disable_notification:
    Optional[aiogram.types.base.Boolean] = None, allow_sending_without_reply:
    Optional[aiogram.types.base.Boolean] = None, reply_markup:
    Optional[Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup,
aiogram.types.reply_keyboard.ReplyKeyboardMarkup,
aiogram.types.reply_keyboard.ReplyKeyboardRemove,
aiogram.types.force_reply.ForceReply]] = None, reply: aiogram.types.base.Boolean =
    False) → aiogram.types.message.Message
```

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

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

Parameters

- **emoji** (`typing.Optional[base.String]`) – Emoji on which the dice throw animation is based. Currently, must be one of “🎲”, “🎯”, “🎰”, “🎱”, or “🎳”. Dice can have values 1-6 for “🎲” and “🎱”, values 1-5 for “🎯” and “🎳”, and values 1-64 for “🎰”. Defaults to “🎲”

- **disable_notification** (`typing.Optional[base.Boolean]`) – Sends the message silently. Users will receive a notification with no sound
- **allow_sending_without_reply** (`typing.Optional[base.Boolean]`) – Pass `True`, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]`) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **reply** (`base.Boolean`) – fill `'reply_to_message_id'`

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

Return type `types.Message`

async answer_chat_action(*action: aiogram.types.base.String*) → `aiogram.types.base.Boolean`

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

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

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

Parameters **action** (`base.String`) – Type of action to broadcast

Returns Returns `True` on success

Return type `base.Boolean`

async reply(*text: aiogram.types.base.String, parse_mode: Optional[aiogram.types.base.String] = None, entities: Optional[List[aiogram.types.message_entity.MessageEntity]] = None, disable_web_page_preview: Optional[aiogram.types.base.Boolean] = None, disable_notification: Optional[aiogram.types.base.Boolean] = None, allow_sending_without_reply: Optional[aiogram.types.base.Boolean] = None, reply_markup: Optional[Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup, aiogram.types.reply_keyboard.ReplyKeyboardMarkup, aiogram.types.reply_keyboard.ReplyKeyboardRemove, aiogram.types.force_reply.ForceReply]] = None, reply: aiogram.types.base.Boolean = True*) → `aiogram.types.message.Message`

Reply to this message

Parameters

- **text** (`base.String`) – Text of the message to be sent
- **parse_mode** (`typing.Optional[base.String]`) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.
- **entities** (`typing.Optional[typing.List[MessageEntity]]`) – List of special entities that appear in message text, which can be specified instead of `parse_mode`
- **disable_web_page_preview** (`typing.Optional[base.Boolean]`) – Disables link previews for links in this message
- **disable_notification** (`typing.Optional[base.Boolean]`) – Sends the message silently. Users will receive a notification with no sound

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

Returns On success, the sent `Message` is returned

Return type `types.Message`

```
async reply_photo(photo: Union[aiogram.types.base.InputFile, aiogram.types.base.String], caption:
    Optional[aiogram.types.base.String] = None, parse_mode:
    Optional[aiogram.types.base.String] = None, caption_entities:
    Optional[List[aiogram.types.message_entity.MessageEntity]] = None,
    disable_notification: Optional[aiogram.types.base.Boolean] = None,
    allow_sending_without_reply: Optional[aiogram.types.base.Boolean] = None,
    reply_markup:
    Optional[Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup,
aiogram.types.reply_keyboard.ReplyKeyboardMarkup,
aiogram.types.reply_keyboard.ReplyKeyboardRemove,
aiogram.types.force_reply.ForceReply]] = None, reply: aiogram.types.base.Boolean =
    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.Optional[base.String]`) – Photo caption (may also be used when resending photos by `file_id`), 0-1024 characters
- **parse_mode** (`typing.Optional[base.String]`) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot’s message.
- **caption_entities** (`typing.Optional[typing.List[MessageEntity]]`) – List of special entities that appear in message text, which can be specified instead of `parse_mode`
- **disable_notification** (`typing.Optional[base.Boolean]`) – Sends the message silently. Users will receive a notification with no sound
- **allow_sending_without_reply** (`typing.Optional[base.Boolean]`) – Pass `True`, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]`) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **reply** (`base.Boolean`) – fill ‘reply_to_message_id’

Returns On success, the sent `Message` is returned

Return type `types.Message`

```

async reply_audio(audio: Union[aiogram.types.base.InputFile, aiogram.types.base.String], caption:
    Optional[aiogram.types.base.String] = None, parse_mode:
    Optional[aiogram.types.base.String] = None, caption_entities:
    Optional[List[aiogram.types.message_entity.MessageEntity]] = None, duration:
    Optional[aiogram.types.base.Integer] = None, performer:
    Optional[aiogram.types.base.String] = None, title:
    Optional[aiogram.types.base.String] = None, thumb:
    Optional[Union[aiogram.types.base.InputFile, aiogram.types.base.String]] = None,
    disable_notification: Optional[aiogram.types.base.Boolean] = None,
    allow_sending_without_reply: Optional[aiogram.types.base.Boolean] = None,
    reply_markup:
    Optional[Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup,
    aiogram.types.reply_keyboard.ReplyKeyboardMarkup,
    aiogram.types.reply_keyboard.ReplyKeyboardRemove,
    aiogram.types.force_reply.ForceReply]] = None, reply: aiogram.types.base.Boolean =
    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.Optional[base.String]) – Audio caption, 0-1024 characters after entities parsing
- **parse_mode** (typing.Optional[base.String]) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.
- **caption_entities** (typing.Optional[typing.List[MessageEntity]]) – List of special entities that appear in message text, which can be specified instead of `parse_mode`
- **duration** (typing.Optional[base.Integer]) – Duration of the audio in seconds
- **performer** (typing.Optional[base.String]) – Performer
- **title** (typing.Optional[base.String]) – Track name
- **thumb** (typing.Union[typing.Union[base.InputFile, base.String], None]) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320.
- **disable_notification** (typing.Optional[base.Boolean]) – Sends the message silently. Users will receive a notification with no sound.
- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **reply** (base.Boolean) – fill 'reply_to_message_id'

Returns On success, the sent Message is returned.

Return type `types.Message`

```
async reply_animation(animation: Union[aiogram.types.base.InputFile, aiogram.types.base.String],
    duration: Optional[aiogram.types.base.Integer] = None, width:
    Optional[aiogram.types.base.Integer] = None, height:
    Optional[aiogram.types.base.Integer] = None, thumb:
    Optional[Union[aiogram.types.base.InputFile, aiogram.types.base.String]] =
    None, caption: Optional[aiogram.types.base.String] = None, parse_mode:
    Optional[aiogram.types.base.String] = None, caption_entities:
    Optional[List[aiogram.types.message_entity.MessageEntity]] = None,
    disable_notification: Optional[aiogram.types.base.Boolean] = None,
    allow_sending_without_reply: Optional[aiogram.types.base.Boolean] = None,
    reply_markup:
    Optional[Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup,
    aiogram.types.reply_keyboard.ReplyKeyboardMarkup,
    aiogram.types.reply_keyboard.ReplyKeyboardRemove,
    aiogram.types.force_reply.ForceReply]] = None, reply:
    aiogram.types.base.Boolean = 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.Optional[base.Integer]`) – Duration of sent animation in seconds
- **width** (`typing.Optional[base.Integer]`) – Animation width
- **height** (`typing.Optional[base.Integer]`) – Animation height
- **thumb** (`typing.Union[typing.Union[base.InputFile, base.String], None]`) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320.
- **caption** (`typing.Optional[base.String]`) – Animation caption (may also be used when resending animation by `file_id`), 0-1024 characters
- **parse_mode** (`typing.Optional[base.String]`) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption
- **caption_entities** (`typing.Optional[typing.List[MessageEntity]]`) – List of special entities that appear in message text, which can be specified instead of `parse_mode`
- **disable_notification** (`typing.Optional[base.Boolean]`) – Sends the message silently. Users will receive a notification with no sound
- **allow_sending_without_reply** (`typing.Optional[base.Boolean]`) – Pass `True`, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (`typing.Union[typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.`

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

- **reply** (`base.Boolean`) – fill `'reply_to_message_id'`

Returns On success, the sent `Message` is returned

Return type `types.Message`

```
async reply_document(document: Union[aiogram.types.base.InputFile, aiogram.types.base.String],
    thumb: Optional[Union[aiogram.types.base.InputFile, aiogram.types.base.String]] = None,
    caption: Optional[aiogram.types.base.String] = None, parse_mode: Optional[aiogram.types.base.String] = None,
    caption_entities: Optional[List[aiogram.types.message_entity.MessageEntity]] = None,
    disable_content_type_detection: Optional[aiogram.types.base.Boolean] = None,
    disable_notification: Optional[aiogram.types.base.Boolean] = None,
    allow_sending_without_reply: Optional[aiogram.types.base.Boolean] = None,
    reply_markup: Optional[Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup,
    aiogram.types.reply_keyboard.ReplyKeyboardMarkup,
    aiogram.types.reply_keyboard.ReplyKeyboardRemove,
    aiogram.types.force_reply.ForceReply]] = None, reply: aiogram.types.base.Boolean = True) → aiogram.types.message.Message
```

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

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

Parameters

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

- **reply** (`typing.Optional[base.Boolean]`) – True if the message is a reply

Returns On success, the sent Message is returned

Return type `types.Message`

```
async reply_video(video: Union[aiogram.types.base.InputFile, aiogram.types.base.String], duration:
    Optional[aiogram.types.base.Integer] = None, width:
    Optional[aiogram.types.base.Integer] = None, height:
    Optional[aiogram.types.base.Integer] = None, thumb:
    Optional[Union[aiogram.types.base.InputFile, aiogram.types.base.String]] = None,
    caption: Optional[aiogram.types.base.String] = None, parse_mode:
    Optional[aiogram.types.base.String] = None, caption_entities:
    Optional[List[aiogram.types.message_entity.MessageEntity]] = None,
    supports_streaming: Optional[aiogram.types.base.Boolean] = None,
    disable_notification: Optional[aiogram.types.base.Boolean] = None,
    allow_sending_without_reply: Optional[aiogram.types.base.Boolean] = None,
    reply_markup:
    Optional[Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup,
aiogram.types.reply_keyboard.ReplyKeyboardMarkup,
aiogram.types.reply_keyboard.ReplyKeyboardRemove,
aiogram.types.force_reply.ForceReply]] = None, reply: aiogram.types.base.Boolean =
    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.Optional[base.Integer]`) – Duration of sent video in seconds
- **width** (`typing.Optional[base.Integer]`) – Video width
- **height** (`typing.Optional[base.Integer]`) – Video height
- **thumb** (`typing.Union[base.InputFile, base.String, None]`) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320.
- **caption** (`typing.Optional[base.String]`) – Video caption (may also be used when resending videos by file_id), 0-1024 characters after entities parsing
- **parse_mode** (`typing.Optional[base.String]`) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption
- **caption_entities** (`typing.Optional[typing.List[MessageEntity]]`) – List of special entities that appear in message text, which can be specified instead of parse_mode
- **supports_streaming** (`typing.Optional[base.Boolean]`) – Pass True, if the uploaded video is suitable for streaming
- **disable_notification** (`typing.Optional[base.Boolean]`) – Sends the message silently. Users will receive a notification with no sound.
- **allow_sending_without_reply** (`typing.Optional[base.Boolean]`) – Pass True, if the message should be sent even if the specified replied-to message is not found

- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]`) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **reply** (`base.Boolean`) – fill 'reply_to_message_id'

Returns On success, the sent Message is returned.

Return type `types.Message`

async reply_voice(*voice: Union[aiogram.types.base.InputFile, aiogram.types.base.String], caption: Optional[aiogram.types.base.String] = None, parse_mode: Optional[aiogram.types.base.String] = None, caption_entities: Optional[List[aiogram.types.message_entity.MessageEntity]] = None, duration: Optional[aiogram.types.base.Integer] = None, disable_notification: Optional[aiogram.types.base.Boolean] = None, allow_sending_without_reply: Optional[aiogram.types.base.Boolean] = None, reply_markup: Optional[Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup, aiogram.types.reply_keyboard.ReplyKeyboardMarkup, aiogram.types.reply_keyboard.ReplyKeyboardRemove, aiogram.types.force_reply.ForceReply]] = None, reply: aiogram.types.base.Boolean = 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.Optional[base.String]`) – Voice message caption, 0-1024 characters after entities parsing
- **parse_mode** (`typing.Optional[base.String]`) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption
- **caption_entities** (`typing.Optional[typing.List[MessageEntity]]`) – List of special entities that appear in message text, which can be specified instead of parse_mode
- **duration** (`typing.Optional[base.Integer]`) – Duration of the voice message in seconds
- **disable_notification** (`typing.Optional[base.Boolean]`) – Sends the message silently. Users will receive a notification with no sound.
- **allow_sending_without_reply** (`typing.Optional[base.Boolean]`) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]`) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **reply** (`base.Boolean`) – fill 'reply_to_message_id'

Returns On success, the sent Message is returned.

Return type `types.Message`

```
async reply_video_note(video_note: Union[aiogram.types.base.InputFile, aiogram.types.base.String],  
                        duration: Optional[aiogram.types.base.Integer] = None, length:  
                        Optional[aiogram.types.base.Integer] = None, thumb:  
                        Optional[Union[aiogram.types.base.InputFile, aiogram.types.base.String]] =  
                        None, disable_notification: Optional[aiogram.types.base.Boolean] = None,  
                        allow_sending_without_reply: Optional[aiogram.types.base.Boolean] = None,  
                        reply_markup:  
                        Optional[Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup,  
aiogram.types.reply_keyboard.ReplyKeyboardMarkup,  
aiogram.types.reply_keyboard.ReplyKeyboardRemove,  
aiogram.types.force_reply.ForceReply]] = None, reply:  
                        aiogram.types.base.Boolean = 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.Optional[base.Integer]`) – Duration of sent video in seconds
- **length** (`typing.Optional[base.Integer]`) – Video width and height
- **thumb** (`typing.Union[typing.Union[base.InputFile, base.String], None]`) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320.
- **disable_notification** (`typing.Optional[base.Boolean]`) – Sends the message silently. Users will receive a notification with no sound.
- **allow_sending_without_reply** (`typing.Optional[base.Boolean]`) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (`(:obj: typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]`)`) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **reply** (`base.Boolean`) – fill 'reply_to_message_id'

Returns On success, the sent Message is returned.

Return type `types.Message`

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

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

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

Parameters

- **media** (`typing.Union[types.MediaGroup, typing.List]`) – A JSON-serialized array describing photos and videos to be sent
- **disable_notification** (`typing.Optional[base.Boolean]`) – Sends the message silently. Users will receive a notification with no sound.
- **allow_sending_without_reply** (`typing.Optional[base.Boolean]`) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply** (`base.Boolean`) – fill 'reply_to_message_id'

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

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

```
async reply_location(latitude: aiogram.types.base.Float, longitude: aiogram.types.base.Float,
    live_period: Optional[aiogram.types.base.Integer] = None, disable_notification:
    Optional[aiogram.types.base.Boolean] = None, horizontal_accuracy:
    Optional[aiogram.types.base.Float] = None, heading:
    Optional[aiogram.types.base.Integer] = None, proximity_alert_radius:
    Optional[aiogram.types.base.Integer] = None, reply_markup:
    Optional[Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup,
    aiogram.types.reply_keyboard.ReplyKeyboardMarkup,
    aiogram.types.reply_keyboard.ReplyKeyboardRemove,
    aiogram.types.force_reply.ForceReply]] = None, reply:
    aiogram.types.base.Boolean = 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
- **horizontal_accuracy** (`typing.Optional[base.Float]`) – The radius of uncertainty for the location, measured in meters; 0-1500
- **live_period** (`typing.Optional[base.Integer]`) – Period in seconds for which the location will be updated
- **heading** (`typing.Optional[base.Integer]`) – For live locations, a direction in which the user is moving, in degrees. Must be between 1 and 360 if specified.
- **proximity_alert_radius** (`typing.Optional[base.Integer]`) – For live locations, a maximum distance for proximity alerts about approaching another chat member, in meters. Must be between 1 and 100000 if specified.
- **disable_notification** (`typing.Optional[base.Boolean]`) – Sends the message silently. Users will receive a notification with no sound.
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]`) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **reply** (`base.Boolean`) – fill 'reply_to_message_id'

Returns On success, the sent Message is returned.

Return type `types.Message`

```
async reply_venue(latitude: aiogram.types.base.Float, longitude: aiogram.types.base.Float, title:
    aiogram.types.base.String, address: aiogram.types.base.String, foursquare_id:
    Optional[aiogram.types.base.String] = None, foursquare_type:
    Optional[aiogram.types.base.String] = None, google_place_id:
    Optional[aiogram.types.base.String] = None, google_place_type:
    Optional[aiogram.types.base.String] = None, disable_notification:
    Optional[aiogram.types.base.Boolean] = None, allow_sending_without_reply:
    Optional[aiogram.types.base.Boolean] = None, reply_markup:
    Optional[Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup,
    aiogram.types.reply_keyboard.ReplyKeyboardMarkup,
    aiogram.types.reply_keyboard.ReplyKeyboardRemove,
    aiogram.types.force_reply.ForceReply]] = None, reply: aiogram.types.base.Boolean =
    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.Optional[base.String]`) – Foursquare identifier of the venue
- **foursquare_type** (`typing.Optional[base.String]`) – Foursquare type of the venue, if known
- **google_place_id** (`typing.Optional[base.String]`) – Google Places identifier of the venue
- **google_place_type** (`typing.Optional[base.String]`) – Google Places type of the venue. See supported types: https://developers.google.com/places/web-service/supported_types
- **disable_notification** (`typing.Optional[base.Boolean]`) – Sends the message silently. Users will receive a notification with no sound
- **allow_sending_without_reply** (`typing.Optional[base.Boolean]`) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]`) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **reply** (`base.Boolean`) – fill 'reply_to_message_id'

Returns On success, the sent Message is returned.

Return type `types.Message`

```
async reply_contact(phone_number: aiogram.types.base.String, first_name: aiogram.types.base.String,
                    last_name: Optional[aiogram.types.base.String] = None, disable_notification:
                    Optional[aiogram.types.base.Boolean] = None, allow_sending_without_reply:
                    Optional[aiogram.types.base.Boolean] = None, reply_markup:
                    Optional[Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup,
                    aiogram.types.reply_keyboard.ReplyKeyboardMarkup,
                    aiogram.types.reply_keyboard.ReplyKeyboardRemove,
                    aiogram.types.force_reply.ForceReply]] = None, reply: aiogram.types.base.Boolean =
                    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.Optional[base.String]`) – Contact’s last name
- **disable_notification** (`typing.Optional[base.Boolean]`) – Sends the message silently. Users will receive a notification with no sound.
- **allow_sending_without_reply** (`typing.Optional[base.Boolean]`) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]`) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **reply** (`base.Boolean`) – fill ‘reply_to_message_id’

Returns On success, the sent Message is returned.

Return type `types.Message`

```
async reply_poll(question: aiogram.types.base.String, options: List[aiogram.types.base.String],
                 is_anonymous: Optional[aiogram.types.base.Boolean] = None, type:
                 Optional[aiogram.types.base.String] = None, allows_multiple_answers:
                 Optional[aiogram.types.base.Boolean] = None, correct_option_id:
                 Optional[aiogram.types.base.Integer] = None, explanation:
                 Optional[aiogram.types.base.String] = None, explanation_parse_mode:
                 Optional[aiogram.types.base.String] = None, explanation_entities:
                 Optional[List[aiogram.types.message_entity.MessageEntity]] = None, open_period:
                 Optional[aiogram.types.base.Integer] = None, close_date:
                 Optional[Union[aiogram.types.base.Integer, datetime.datetime, datetime.timedelta]] =
                 None, is_closed: Optional[aiogram.types.base.Boolean] = None, disable_notification:
                 Optional[aiogram.types.base.Boolean] = None, allow_sending_without_reply:
                 Optional[aiogram.types.base.Boolean] = None, reply_markup:
                 Optional[Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup,
                 aiogram.types.reply_keyboard.ReplyKeyboardMarkup,
                 aiogram.types.reply_keyboard.ReplyKeyboardRemove,
                 aiogram.types.force_reply.ForceReply]] = None, reply: aiogram.types.base.Boolean =
                 True) → aiogram.types.message.Message
```

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

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

Parameters

- **question** (`base.String`) – Poll question, 1-255 characters
- **options** (`typing.List[base.String]`) – List of answer options, 2-10 strings 1-100 characters each
- **is_anonymous** (`typing.Optional[base.Boolean]`) – True, if the poll needs to be anonymous, defaults to True
- **type** (`typing.Optional[base.String]`) – Poll type, “quiz” or “regular”, defaults to “regular”
- **allows_multiple_answers** (`typing.Optional[base.Boolean]`) – True, if the poll allows multiple answers, ignored for polls in quiz mode, defaults to False
- **correct_option_id** (`typing.Optional[base.Integer]`) – 0-based identifier of the correct answer option, required for polls in quiz mode
- **explanation** (`typing.Optional[base.String]`) – Text that is shown when a user chooses an incorrect answer or taps on the lamp icon in a quiz-style poll, 0-200 characters with at most 2 line feeds after entities parsing
- **explanation_parse_mode** (`typing.Optional[base.String]`) – Mode for parsing entities in the explanation. See formatting options for more details.
- **explanation_entities** (`typing.Optional[typing.List[MessageEntity]]`) – List of special entities that appear in message text, which can be specified instead of `parse_mode`
- **open_period** (`typing.Optional[base.Integer]`) – Amount of time in seconds the poll will be active after creation, 5-600. Can't be used together with `close_date`.
- **close_date** (`typing.Union[base.Integer, datetime.datetime, datetime.timedelta, None]`) – Point in time (Unix timestamp) when the poll will be automatically closed. Must be at least 5 and no more than 600 seconds in the future. Can't be used together with `open_period`.
- **is_closed** (`typing.Optional[base.Boolean]`) – Pass True, if the poll needs to be immediately closed
- **disable_notification** (`typing.Optional[base.Boolean]`) – Sends the message silently. Users will receive a notification with no sound.
- **allow_sending_without_reply** (`typing.Optional[base.Boolean]`) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (`typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]`) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **reply** (`base.Boolean`) – fill 'reply_to_message_id'

Returns On success, the sent Message is returned

Return type `types.Message`

```
async reply_sticker(sticker: Union[aiogram.types.base.InputFile, aiogram.types.base.String],
                    disable_notification: Optional[aiogram.types.base.Boolean] = None,
                    allow_sending_without_reply: Optional[aiogram.types.base.Boolean] = None,
                    reply_markup:
                    Optional[Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup,
                    aiogram.types.reply_keyboard.ReplyKeyboardMarkup,
                    aiogram.types.reply_keyboard.ReplyKeyboardRemove,
                    aiogram.types.force_reply.ForceReply]] = None,
                    reply: aiogram.types.base.Boolean = 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.Optional[base.Boolean]) – Sends the message silently. Users will receive a notification with no sound.
- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply, None]) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **reply** (base.Boolean) – fill 'reply_to_message_id'

Returns On success, the sent Message is returned.

Return type types.Message

```
async reply_dice(emoji: Optional[aiogram.types.base.String] = None,
                 disable_notification:
                 Optional[aiogram.types.base.Boolean] = None,
                 allow_sending_without_reply:
                 Optional[aiogram.types.base.Boolean] = None,
                 reply_markup:
                 Optional[Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup,
                 aiogram.types.reply_keyboard.ReplyKeyboardMarkup,
                 aiogram.types.reply_keyboard.ReplyKeyboardRemove,
                 aiogram.types.force_reply.ForceReply]] = None,
                 reply: aiogram.types.base.Boolean = True) → aiogram.types.message.Message
```

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

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

Parameters

- **emoji** (typing.Optional[base.String]) – Emoji on which the dice throw animation is based. Currently, must be one of “🎲”, “🎯”, “🎰”, “🎱”, or “🎮”. Dice can have values 1-6 for “🎲” and “🎱”, values 1-5 for “🎰” and “🎯”, and values 1-64 for “🎮”. Defaults to “🎲”
- **disable_notification** (typing.Optional[base.Boolean]) – Sends the message silently. Users will receive a notification with no sound
- **allow_sending_without_reply** (typing.Optional[base.Boolean]) – Pass True, if the message should be sent even if the specified replied-to message is not found
- **reply_markup** (typing.Union[types.InlineKeyboardMarkup, types.ReplyKeyboardMarkup, types.ReplyKeyboardRemove, types.ForceReply,

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

- **reply** (`base.Boolean`) – fill 'reply_to_message_id'

Returns On success, the sent Message is returned.

Return type `types.Message`

```
async forward(chat_id: Union[aiogram.types.base.Integer, aiogram.types.base.String],
               disable_notification: Optional[aiogram.types.base.Boolean] = None) →
               aiogram.types.message.Message
```

Forward this message

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

Parameters

- **chat_id** (`typing.Union[base.Integer, base.String]`) – Unique identifier for the target chat or username of the target channel
- **disable_notification** (`typing.Optional[base.Boolean]`) – Sends the message silently. Users will receive a notification with no sound

Returns On success, the sent Message is returned

Return type `types.Message`

```
async edit_text(text: aiogram.types.base.String, parse_mode: Optional[aiogram.types.base.String] =
                 None, entities: Optional[List[aiogram.types.message_entity.MessageEntity]] = None,
                 disable_web_page_preview: Optional[aiogram.types.base.Boolean] = None,
                 reply_markup: Optional[aiogram.types.inline_keyboard.InlineKeyboardMarkup] =
                 None) → Union[aiogram.types.message.Message, aiogram.types.base.Boolean]
```

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

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

Parameters

- **text** (`base.String`) – New text of the message
- **parse_mode** (`typing.Optional[base.String]`) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.
- **entities** (`typing.Optional[typing.List[MessageEntity]]`) – List of special entities that appear in message text, which can be specified instead of `parse_mode`
- **disable_web_page_preview** (`typing.Optional[base.Boolean]`) – Disables link previews for links in this message
- **reply_markup** (`typing.Optional[types.InlineKeyboardMarkup]`) – A JSON-serialized object for an inline keyboard.

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

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

```

async edit_caption(caption: aiogram.types.base.String, parse_mode:
    Optional[aiogram.types.base.String] = None, caption_entities:
    Optional[List[aiogram.types.message_entity.MessageEntity]] = None, reply_markup:
    Optional[aiogram.types.inline_keyboard.InlineKeyboardMarkup] = None) →
    Union[aiogram.types.message.Message, aiogram.types.base.Boolean]

```

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

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

Parameters

- **caption** (typing.Optional[base.String]) – New caption of the message
- **parse_mode** (typing.Optional[base.String]) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.
- **caption_entities** (typing.Optional[typing.List[MessageEntity]]) – List of special entities that appear in message text, which can be specified instead of parse_mode
- **reply_markup** (typing.Optional[types.InlineKeyboardMarkup]) – A JSON-serialized object for an inline keyboard

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

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

```

async edit_media(media: aiogram.types.input_media.InputMedia, reply_markup:
    Optional[aiogram.types.inline_keyboard.InlineKeyboardMarkup] = None) →
    Union[aiogram.types.message.Message, aiogram.types.base.Boolean]

```

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

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

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

Parameters

- **media** (types.InputMedia) – A JSON-serialized object for a new media content of the message
- **reply_markup** (typing.Optional[types.InlineKeyboardMarkup]) – A JSON-serialized object for a new inline keyboard

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

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

```

async edit_reply_markup(reply_markup:
    Optional[aiogram.types.inline_keyboard.InlineKeyboardMarkup] = None) →
    Union[aiogram.types.message.Message, aiogram.types.base.Boolean]

```

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

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

Parameters **reply_markup** (typing.Optional[types.InlineKeyboardMarkup]) – A JSON-serialized object for an inline keyboard

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

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

async delete_reply_markup() → Union[*aiogram.types.message.Message*, aiogram.types.base.Boolean]
Use this method to delete reply markup of messages sent by the bot or via the bot (for inline bots).

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

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

async edit_live_location(*latitude: aiogram.types.base.Float, longitude: aiogram.types.base.Float, reply_markup: Optional[aiogram.types.inline_keyboard.InlineKeyboardMarkup] = None*) → Union[*aiogram.types.message.Message*, aiogram.types.base.Boolean]

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

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

Parameters

- **latitude** (base.Float) – Latitude of new location
- **longitude** (base.Float) – Longitude of new location
- **reply_markup** (typing.Optional[types.InlineKeyboardMarkup]) – A JSON-serialized object for a new inline keyboard.

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

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

async stop_live_location(*reply_markup: Optional[aiogram.types.inline_keyboard.InlineKeyboardMarkup] = None*) → Union[*aiogram.types.message.Message*, aiogram.types.base.Boolean]

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

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

Parameters **reply_markup** (typing.Optional[types.InlineKeyboardMarkup]) – A JSON-serialized object for a new inline keyboard.

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

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

async delete() → aiogram.types.base.Boolean

Use this method to delete a message, including service messages, with the following limitations: - A message can only be deleted if it was sent less than 48 hours ago. - Bots can delete outgoing messages in private chats, groups, and supergroups. - Bots can delete incoming messages in private chats. - Bots granted can_post_messages permissions can delete outgoing messages in channels. - If the bot is an administrator of a group, it can delete any message there. - If the bot has can_delete_messages permission in a supergroup or a channel, it can delete any message there.

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

Returns Returns True on success

Return type `base.Boolean`

async pin(*disable_notification: Optional[aiogram.types.base.Boolean] = None*) → `aiogram.types.base.Boolean`

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

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

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

Returns Returns True on success

Return type `base.Boolean`

async unpin() → `aiogram.types.base.Boolean`

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

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

Returns Returns True on success

Return type `base.Boolean`

async send_copy(*chat_id: Union[str, int], disable_notification: Optional[bool] = None, disable_web_page_preview: Optional[bool] = None, reply_to_message_id: Optional[int] = None, allow_sending_without_reply: Optional[aiogram.types.base.Boolean] = None, reply_markup: Optional[Union[aiogram.types.inline_keyboard.InlineKeyboardMarkup, aiogram.types.reply_keyboard.ReplyKeyboardMarkup]] = None*) → `aiogram.types.message.Message`

Send copy of current message

Parameters

- **chat_id** –
- **disable_notification** –
- **disable_web_page_preview** – for text messages only
- **reply_to_message_id** –
- **allow_sending_without_reply** –
- **reply_markup** –

Returns

ContentType

class `aiogram.types.message.ContentType`
Bases: `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 POLL
Key DICE
Key NEW_CHAT_MEMBERS
Key LEFT_CHAT_MEMBER
Key INVOICE
Key SUCCESSFUL_PAYMENT
Key CONNECTED_WEBSITE
Key MIGRATE_TO_CHAT_ID
Key MIGRATE_FROM_CHAT_ID
Key UNKNOWN
Key ANY

ContentTypes

class aiogram.types.message.**ContentTypes**

Bases: 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 POLL
Key DICE
Key NEW_CHAT_MEMBERS
Key LEFT_CHAT_MEMBER
Key INVOICE
Key SUCCESSFUL_PAYMENT
Key CONNECTED_WEBSITE
Key MIGRATE_TO_CHAT_ID
Key MIGRATE_FROM_CHAT_ID
Key UNKNOWN
Key ANY

ParseMode

class aiogram.types.message.**ParseMode**
Bases: aiogram.utils.helper.Helper
Parse modes
Key MARKDOWN
Key HTML

MaskPosition

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

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: Optional[Dict[str, Any]] = None,  
                                                         **kwargs: Any)
```

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: Optional[Dict[str, Any]] = None, **kwargs: Any)
```

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: Optional[Dict[str, Any]] = None,  
                                                         **kwargs: Any)
```

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

Filter factory greatly simplifies the reuse of filters when registering handlers.

Filters factory

```
class aiogram.dispatcher.filters.FiltersFactory(dispatcher)
    Bases: object

    Filters factory

    bind(callback: Union[Callable, aiogram.dispatcher.filters.filters.AbstractFilter], validator:
        Optional[Callable] = None, event_handlers: Optional[List[aiogram.dispatcher.handler.Handler]] =
        None, exclude_event_handlers: Optional[Iterable[aiogram.dispatcher.handler.Handler]] = None)
        Register filter
```

Parameters

- **callback** – callable or subclass of `AbstractFilter`
- **validator** – custom validator.
- **event_handlers** – list of instances of `Handler`
- **exclude_event_handlers** – list of excluded event handlers (`Handler`)

```
unbind(callback: Union[Callable, aiogram.dispatcher.filters.filters.AbstractFilter])
    Unregister filter
```

Parameters **callback** – callable of subclass of `AbstractFilter`

```
resolve(event_handler, *custom_filters, **full_config) → List[Union[Callable,
    aiogram.dispatcher.filters.filters.AbstractFilter]]
    Resolve filters to filters-set
```

Parameters

- **event_handler** –
- **custom_filters** –
- **full_config** –

Returns

Builtin filters

aiogram has some builtin filters. Here you can see all of them:

Command

```
class aiogram.dispatcher.filters.Command(commands: Union[Iterable, str], prefixes: Union[Iterable, str]
                                         = '/', ignore_case: bool = True, ignore_mention: bool = False,
                                         ignore_caption: bool = True)
```

Bases: [aiogram.dispatcher.filters.filters.Filter](#)

You can handle commands by using this filter.

If filter is successful processed the [Command.CommandObj](#) will be passed to the handler arguments.

By default this filter is registered for messages and edited messages handlers.

Filter can be initialized from filters factory or by simply creating instance of this class.

Examples:

```
@dp.message_handler(commands=['myCommand'])
@dp.message_handler(Command(['myCommand']))
@dp.message_handler(commands=['myCommand'], commands_prefix='!/')
```

Parameters

- **commands** – Command or list of commands always without leading slashes (prefix)
- **prefixes** – Allowed commands prefix. By default is slash. If you change the default behavior pass the list of prefixes to this argument.
- **ignore_case** – Ignore case of the command
- **ignore_mention** – Ignore mention in command (By default this filter pass only the commands addressed to current bot)
- **ignore_caption** – Ignore caption from message (in message types like photo, video, audio, etc) By default is True. If you want check commands in captions, you also should set required content_types.

Examples:

```
@dp.message_handler(commands=['myCommand'], commands_ignore_
↳ caption=False, content_types=ContentType.ANY)
@dp.message_handler(Command(['myCommand'], ignore_caption=False),
↳ content_types=[ContentType.TEXT, ContentType.DOCUMENT])
```

```
classmethod validate(full_config: Dict[str, Any]) → Optional[Dict[str, Any]]
```

Validator for filters factory

From filters factory this filter can be registered with arguments:

- **command**
- **commands_prefix** (will be passed as **prefixes**)
- **commands_ignore_mention** (will be passed as **ignore_mention**)
- **commands_ignore_caption** (will be passed as **ignore_caption**)

Parameters **full_config** –

Returns config or empty dict

async check(*message*: [aiogram.types.message.Message](#))

Will be called when filters checks.

This method must be overridden.

Parameters *args* –

Returns

class CommandObj(*prefix*: *str* = '/', *command*: *str* = '', *mention*: *Optional[str]* = *None*, *args*: *Optional[str]* = *None*)

Bases: `object`

Instance of this object is always has command and it prefix.

Can be passed as keyword argument `command` to the handler

prefix: *str* = '/'

Command without prefix and mention

command: *str* = ''

Mention (if available)

mention: *str* = *None*

Command argument

property mentioned: *bool*

This command has mention?

Returns

property text: *str*

Generate original text from object

Returns

CommandStart

class `aiogram.dispatcher.filters.CommandStart`(*deep_link*: *Optional[Union[str, Pattern[str]]]* = *None*, *encoded*: *bool* = *False*)

Bases: `aiogram.dispatcher.filters.builtin.Command`

This filter based on `Command` filter but can handle only `/start` command.

Also this filter can handle `deep-linking` arguments.

Example:

```
@dp.message_handler(CommandStart(re.compile(r'ref-([\d]+)')))
```

Parameters

- **deep_link** – string or compiled regular expression (by `re.compile(...)`).
- **encoded** – set `True` if you're waiting for encoded payload (default - `False`).

async check(*message*: [aiogram.types.message.Message](#))

If deep-linking is passed to the filter result of the matching will be passed as `deep_link` to the handler

Parameters *message* –

Returns

CommandHelp

class aiogram.dispatcher.filters.CommandHelp

Bases: *aiogram.dispatcher.filters.builtin.Command*

This filter based on *Command* filter but can handle only `/help` command.

Filter can be initialized from filters factory or by simply creating instance of this class.

Examples:

```
@dp.message_handler(commands=['myCommand'])
@dp.message_handler(Command(['myCommand']))
@dp.message_handler(commands=['myCommand'], commands_prefix='!/')
```

Parameters

- **commands** – Command or list of commands always without leading slashes (prefix)
- **prefixes** – Allowed commands prefix. By default is slash. If you change the default behavior pass the list of prefixes to this argument.
- **ignore_case** – Ignore case of the command
- **ignore_mention** – Ignore mention in command (By default this filter pass only the commands addressed to current bot)
- **ignore_caption** – Ignore caption from message (in message types like photo, video, audio, etc) By default is True. If you want check commands in captions, you also should set required `content_types`.

Examples:

```
@dp.message_handler(commands=['myCommand'], commands_ignore_
↳ caption=False, content_types=ContentType.ANY)
@dp.message_handler(Command(['myCommand'], ignore_caption=False),
↳ content_types=[ContentType.TEXT, ContentType.DOCUMENT])
```

CommandSettings

class aiogram.dispatcher.filters.CommandSettings

Bases: *aiogram.dispatcher.filters.builtin.Command*

This filter based on *Command* filter but can handle only `/settings` command.

Filter can be initialized from filters factory or by simply creating instance of this class.

Examples:

```
@dp.message_handler(commands=['myCommand'])
@dp.message_handler(Command(['myCommand']))
@dp.message_handler(commands=['myCommand'], commands_prefix='!/')
```

Parameters

- **commands** – Command or list of commands always without leading slashes (prefix)
- **prefixes** – Allowed commands prefix. By default is slash. If you change the default behavior pass the list of prefixes to this argument.

- **ignore_case** – Ignore case of the command
- **ignore_mention** – Ignore mention in command (By default this filter pass only the commands addressed to current bot)
- **ignore_caption** – Ignore caption from message (in message types like photo, video, audio, etc) By default is True. If you want check commands in captions, you also should set required content_types.

Examples:

```
@dp.message_handler(commands=['myCommand'], commands_ignore_
    ↪caption=False, content_types=ContentType.ANY)
@dp.message_handler(Command(['myCommand'], ignore_caption=False),
    ↪content_types=[ContentType.TEXT, ContentType.DOCUMENT])
```

CommandPrivacy

class aiogram.dispatcher.filters.CommandPrivacy

Bases: *aiogram.dispatcher.filters.builtin.Command*

This filter based on *Command* filter but can handle only /privacy command.

Filter can be initialized from filters factory or by simply creating instance of this class.

Examples:

```
@dp.message_handler(commands=['myCommand'])
@dp.message_handler(Command(['myCommand']))
@dp.message_handler(commands=['myCommand'], commands_prefix='!/')
```

Parameters

- **commands** – Command or list of commands always without leading slashes (prefix)
- **prefixes** – Allowed commands prefix. By default is slash. If you change the default behavior pass the list of prefixes to this argument.
- **ignore_case** – Ignore case of the command
- **ignore_mention** – Ignore mention in command (By default this filter pass only the commands addressed to current bot)
- **ignore_caption** – Ignore caption from message (in message types like photo, video, audio, etc) By default is True. If you want check commands in captions, you also should set required content_types.

Examples:

```
@dp.message_handler(commands=['myCommand'], commands_ignore_
    ↪caption=False, content_types=ContentType.ANY)
@dp.message_handler(Command(['myCommand'], ignore_caption=False),
    ↪content_types=[ContentType.TEXT, ContentType.DOCUMENT])
```

Text

```
class aiogram.dispatcher.filters.Text(equals: Optional[Union[str, babel.support.LazyProxy,  
    Iterable[Union[str, babel.support.LazyProxy]]]] = None, contains:  
    Optional[Union[str, babel.support.LazyProxy, Iterable[Union[str,  
        babel.support.LazyProxy]]]] = None, startswith:  
    Optional[Union[str, babel.support.LazyProxy, Iterable[Union[str,  
        babel.support.LazyProxy]]]] = None, endswith:  
    Optional[Union[str, babel.support.LazyProxy, Iterable[Union[str,  
        babel.support.LazyProxy]]]] = None, ignore_case=False)
```

Bases: [aiogram.dispatcher.filters.filters.Filter](#)

Simple text filter

Check text for one of pattern. Only one mode can be used in one filter. In every pattern, a single string is treated as a list with 1 element.

Parameters

- **equals** – True if object's text in the list
- **contains** – True if object's text contains all strings from the list
- **startswith** – True if object's text starts with any of strings from the list
- **endswith** – True if object's text ends with any of strings from the list
- **ignore_case** – case insensitive

```
async check(obj: Union[aiogram.types.message.Message, aiogram.types.callback_query.CallbackQuery,  
    aiogram.types.inline_query.InlineQuery, aiogram.types.poll.Poll])
```

Will be called when filters checks.

This method must be overridden.

Parameters **args** –

Returns

HashTag

```
class aiogram.dispatcher.filters.HashTag(hashtags=None, cashtags=None)
```

Bases: [aiogram.dispatcher.filters.filters.Filter](#)

Filter for hashtag's and cashtag's

```
async check(message: aiogram.types.message.Message)
```

Will be called when filters checks.

This method must be overridden.

Parameters **args** –

Returns

Regexp

```
class aiogram.dispatcher.filters.Regexp(regexp)
    Bases: aiogram.dispatcher.filters.filters.Filter
    Regexp filter for messages and callback query
    async check(obj: Union[aiogram.types.message.Message, aiogram.types.callback_query.CallbackQuery,
        aiogram.types.inline_query.InlineQuery, aiogram.types.poll.Poll])
        Will be called when filters checks.
        This method must be overridden.
        Parameters args –
        Returns
```

RegexpCommandsFilter

```
class aiogram.dispatcher.filters.RegexpCommandsFilter(regexp_commands)
    Bases: aiogram.dispatcher.filters.filters.BoundFilter
    Check commands by regexp in message
    async check(message)
        Will be called when filters checks.
        This method must be overridden.
        Parameters args –
        Returns
```

ContentTypeFilter

```
class aiogram.dispatcher.filters.ContentTypeFilter(content_types)
    Bases: aiogram.dispatcher.filters.filters.BoundFilter
    Check message content type
    async check(message)
        Will be called when filters checks.
        This method must be overridden.
        Parameters args –
        Returns
```

IsSenderContact

```
class aiogram.dispatcher.filters.IsSenderContact(is_sender_contact: bool)
    Bases: aiogram.dispatcher.filters.filters.BoundFilter

    Filter check that the contact matches the sender

    is_sender_contact=True - contact matches the sender is_sender_contact=False - result will be inverted

    async check(message: aiogram.types.message.Message) → bool
        Will be called when filters checks.

        This method must be overridden.

        Parameters args –
        Returns
```

StateFilter

```
class aiogram.dispatcher.filters.StateFilter(dispatcher, state)
    Bases: aiogram.dispatcher.filters.filters.BoundFilter

    Check user state

    async check(obj)
        Will be called when filters checks.

        This method must be overridden.

        Parameters args –
        Returns
```

ExceptionsFilter

```
class aiogram.dispatcher.filters.ExceptionsFilter(exception)
    Bases: aiogram.dispatcher.filters.filters.BoundFilter

    Filter for exceptions

    async check(update, exception)
        Will be called when filters checks.

        This method must be overridden.

        Parameters args –
        Returns
```

IDFilter

```
class aiogram.dispatcher.filters.builtin.IDFilter(user_id: Optional[Union[Iterable[Union[int, str]],
                                                                    str, int]] = None, chat_id:
                                                                    Optional[Union[Iterable[Union[int, str]], str, int]]
                                                                    = None)
```

Bases: *aiogram.dispatcher.filters.filters.Filter*

Parameters

- **user_id** –
- **chat_id** –

```
async check(obj: Union[aiogram.types.message.Message, aiogram.types.callback_query.CallbackQuery,
                        aiogram.types.inline_query.InlineQuery,
                        aiogram.types.chat_member_updated.ChatMemberUpdated])
```

Will be called when filters checks.

This method must be overridden.

Parameters args –

Returns

AdminFilter

```
class aiogram.dispatcher.filters.AdminFilter(is_chat_admin: Optional[Union[Iterable[Union[int, str]],
                                                                    str, int, bool]] = None)
```

Bases: *aiogram.dispatcher.filters.filters.Filter*

Checks if user is admin in a chat. If `is_chat_admin` is not set, the filter will check in the current chat (correct only for messages). `is_chat_admin` is required for `InlineQuery`.

```
async check(obj: Union[aiogram.types.message.Message, aiogram.types.callback_query.CallbackQuery,
                        aiogram.types.inline_query.InlineQuery,
                        aiogram.types.chat_member_updated.ChatMemberUpdated]) → bool
```

Will be called when filters checks.

This method must be overridden.

Parameters args –

Returns

IsReplyFilter

```
class aiogram.dispatcher.filters.IsReplyFilter(is_reply)
```

Bases: *aiogram.dispatcher.filters.filters.BoundFilter*

Check if message is replied and send reply message to handler

```
async check(msg: aiogram.types.message.Message)
```

Will be called when filters checks.

This method must be overridden.

Parameters args –

Returns

ForwardedMessageFilter

class aiogram.dispatcher.filters.**ForwardedMessageFilter**(*is_forwarded: bool*)

Bases: *aiogram.dispatcher.filters.filters.BoundFilter*

async check(*message: aiogram.types.message.Message*)

Will be called when filters checks.

This method must be overridden.

Parameters **args** –

Returns

ChatTypeFilter

class aiogram.dispatcher.filters.**ChatTypeFilter**(*chat_type: Container[aiogram.types.chat.ChatType]*)

Bases: *aiogram.dispatcher.filters.filters.BoundFilter*

async check(*obj: Union[aiogram.types.message.Message, aiogram.types.callback_query.CallbackQuery, aiogram.types.chat_member_updated.ChatMemberUpdated]*)

Will be called when filters checks.

This method must be overridden.

Parameters **args** –

Returns

MediaGroupFilter

class aiogram.dispatcher.filters.**MediaGroupFilter**(*is_media_group: bool*)

Bases: *aiogram.dispatcher.filters.filters.BoundFilter*

Check if message is part of a media group.

is_media_group=True - the message is part of a media group *is_media_group=False* - the message is NOT part of a media group

async check(*message: aiogram.types.message.Message*) → bool

Will be called when filters checks.

This method must be overridden.

Parameters **args** –

Returns

Making own filters (Custom filters)

Own filter can be:

- any callable object
- any async function
- any anonymous function (Example: `lambda msg: msg.text == 'spam'`)
- Subclass of `AbstractFilter`, `Filter` or `BoundFilter`

AbstractFilter

class aiogram.dispatcher.filters.**AbstractFilter**

Bases: abc.ABC

Abstract class for custom filters.

abstract classmethod **validate**(*full_config: Dict[str, Any]*) → Optional[Dict[str, Any]]

Validate and parse config.

This method will be called by the filters factory when you bind this filter. Must be overridden.

Parameters **full_config** – dict with arguments passed to handler registrar

Returns Current filter config

abstract async **check**(*args) → bool

Will be called when filters checks.

This method must be overridden.

Parameters **args** –

Returns

Filter

class aiogram.dispatcher.filters.**Filter**

Bases: [aiogram.dispatcher.filters.filters.AbstractFilter](#)

You can make subclasses of that class for custom filters.

Method check must be overridden

classmethod **validate**(*full_config: Dict[str, Any]*) → Optional[Dict[str, Any]]

Here method validate is optional. If you need to use filter from filters factory you need to override this method.

Parameters **full_config** – dict with arguments passed to handler registrar

Returns Current filter config

BoundFilter

class aiogram.dispatcher.filters.**BoundFilter**

Bases: [aiogram.dispatcher.filters.filters.Filter](#)

To easily create your own filters with one parameter, you can inherit from this filter.

You need to implement `__init__` method with single argument related with key attribute and `check` method where you need to implement filter logic.

key = None

Unique name of the filter argument. You need to override this attribute.

required = False

If True this filter will be added to the all of the registered handlers

default = None

Default value for configure required filters

classmethod `validate`(*full_config*: Dict[str, Any]) → Dict[str, Any]

If `cls.key` is not None and that is in config returns config with that argument.

Parameters `full_config` –

Returns

```
class ChatIdFilter(BoundFilter):
    key = 'chat_id'

    def __init__(self, chat_id: typing.Union[typing.Iterable, int]):
        if isinstance(chat_id, int):
            chat_id = [chat_id]
        self.chat_id = chat_id

    def check(self, message: types.Message) -> bool:
        return message.chat.id in self.chat_id

dp.filters_factory.bind(ChatIdFilter, event_handlers=[dp.message_handlers])
```

4.5.2 Finite state machine

Storage

Coming soon...

Available storage's

Coming soon...

Memory storage

class `aiogram.contrib.fsm_storage.memory.MemoryStorage`

Bases: `aiogram.dispatcher.storage.BaseStorage`

In-memory based states storage.

This type of storage is not recommended for usage in bots, because you will lost all states after restarting.

Redis storage

```
class aiogram.contrib.fsm_storage.redis.RedisStorage2(host: str = 'localhost', port: int = 6379, db:
    Optional[int] = None, password:
    Optional[str] = None, ssl: Optional[bool] =
    None, pool_size: int = 10, loop:
    Optional[asyncio.events.AbstractEventLoop]
    = None, prefix: str = 'fsm', state_ttl:
    Optional[int] = None, data_ttl: Optional[int]
    = None, bucket_ttl: Optional[int] = None,
    **kwargs)
```

Bases: `aiogram.dispatcher.storage.BaseStorage`

Busted Redis-base storage for FSM. Works with Redis connection pool and customizable keys prefix.

Usage:

```
storage = RedisStorage2('localhost', 6379, db=5, pool_size=10, prefix='my_fsm_key')
dp = Dispatcher(bot, storage=storage)
```

And need to close Redis connection when shutdown

```
await dp.storage.close()
await dp.storage.wait_closed()
```

Mongo storage

```
class aiogram.contrib.fsm_storage.mongo.MongoStorage(host='localhost', port=27017,
                                                    db_name='aiogram_fsm', uri=None,
                                                    username=None, password=None, index=True,
                                                    **kwargs)
```

Bases: aiogram.dispatcher.storage.BaseStorage

Mongo-based storage for FSM.

Usage:

```
storage = MongoStorage(host='localhost', port=27017, db_name='aiogram_fsm')
dp = Dispatcher(bot, storage=storage)
```

And need to close Mongo client connections when shutdown

```
await dp.storage.close()
await dp.storage.wait_closed()
```

Rethink DB storage

```
class aiogram.contrib.fsm_storage.rethinkdb.RethinkDBStorage(host: str = 'localhost', port: int =
                                                            28015, db: str = 'aiogram', table: str
                                                            = 'aiogram', auth_key: Optional[str]
                                                            = None, user: Optional[str] = None,
                                                            password: Optional[str] = None,
                                                            timeout: int = 20, ssl: Optional[dict]
                                                            = None, loop: Op-
                                                            tional[asyncio.events.AbstractEventLoop]
                                                            = None)
```

Bases: aiogram.dispatcher.storage.BaseStorage

RethinkDB-based storage for FSM.

Usage:

```
storage = RethinkDBStorage(db='aiogram', table='aiogram', user='aiogram', password=
↪ 'aiogram_secret')
dispatcher = Dispatcher(bot, storage=storage)
```

And need to close connection when shutdown

```
await storage.close()
await storage.wait_closed()
```

Making own storage's

Coming soon...

States

Coming soon...

State utils

Coming soon...

State

Coming soon...

States group

Coming soon...

4.5.3 Middleware

Bases

Coming soon...

Making own middleware's

Coming soon...

Available middleware's

Coming soon...

4.5.4 Webhook

Coming soon...

Bases

Coming soon...

Security

Coming soon...

Making requests when getting updates

Coming soon...

4.5.5 Basics

Coming soon...

4.5.6 Available handlers

Coming soon...

Handler class

Coming soon...

4.5.7 Features

Coming soon...

4.5.8 Dispatcher class

```
class aiogram.Dispatcher(bot, loop=None, storage: Optional[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.

async 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

async process_updates(*updates*, *fast*: *Optional[bool]* = *True*)

Process list of updates

Parameters

- **updates** –
- **fast** –

Returns

async process_update(*update*: *aiogram.types.update.Update*)

Process single update object

Parameters **update** –

Returns

async reset_webhook(*check*=*True*) → *bool*

Reset webhook

Parameters **check** – check before deleting

Returns

async start_polling(*timeout*=*20*, *relax*=*0.1*, *limit*=*None*, *reset_webhook*=*None*, *fast*: *Optional[bool]* = *True*, *error_sleep*: *int* = *5*, *allowed_updates*: *Optional[List[str]]* = *None*)

Start long-polling

Parameters

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

Returns

stop_polling()

Break long-polling process.

Returns

async wait_closed()

Wait for the long-polling to close

Returns

is_polling()

Check if polling is enabled

Returns

register_message_handler(*callback*, **custom_filters*, *commands*=*None*, *regexp*=*None*, *content_types*=*None*, *state*=*None*, *run_task*=*None*, ***kwargs*)

Register handler for message

```
# This handler works only if state is None (by default).
dp.register_message_handler(cmd_start, commands=['start', 'about'])
dp.register_message_handler(entry_point, commands=['setup'])

# This handler works only if current state is "first_step"
dp.register_message_handler(step_handler_1, state="first_step")

# If you want to handle all states by one handler, use `state="*"`.
dp.register_message_handler(cancel_handler, commands=['cancel'], state="*")
dp.register_message_handler(cancel_handler, lambda msg: msg.text.lower() ==
    ↪ 'cancel', state="*")
```

Parameters

- **callback** –
- **commands** – list of commands
- **regex** – REGEXP
- **content_types** – List of content types.
- **custom_filters** – list of custom filters
- **kwargs** –
- **state** –

Returns decorated function

message_handler(*custom_filters, commands=None, regex=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(regex='^[a-z]+-[0-9]+')
async def msg_handler(message: types.Message):
```

Filter messages by command regular expression:

```
@dp.message_handler(filters.RegexpCommandsFilter(regex_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_poll_handler(*callback, *custom_filters, run_task=None, **kwargs*)

Register handler for poll

Example:

```
dp.register_poll_handler(some_poll_handler)
```

Parameters

- **callback** –
- **custom_filters** –
- **run_task** – run callback in task (no wait results)
- **kwargs** –

poll_handler(**custom_filters*, *run_task=None*, ***kwargs*)

Decorator for poll handler

Example:

```
@dp.poll_handler()
async def some_poll_handler(poll: types.Poll)
```

Parameters

- **custom_filters** –
- **run_task** – run callback in task (no wait results)
- **kwargs** –

register_poll_answer_handler(*callback*, **custom_filters*, *run_task=None*, ***kwargs*)

Register handler for poll_answer

Example:

```
dp.register_poll_answer_handler(some_poll_answer_handler)
```

Parameters

- **callback** –
- **custom_filters** –
- **run_task** – run callback in task (no wait results)
- **kwargs** –

poll_answer_handler(**custom_filters*, *run_task=None*, ***kwargs*)

Decorator for poll_answer handler

Example:

```
@dp.poll_answer_handler()
async def some_poll_answer_handler(poll_answer: types.PollAnswer)
```

Parameters

- **custom_filters** –
- **run_task** – run callback in task (no wait results)
- **kwargs** –

register_my_chat_member_handler(*callback*: *Callable*, **custom_filters*, *run_task*: *Optional[bool]* = *None*, ***kwargs*) → *None*

Register handler for my_chat_member

Example:

```
dp.register_my_chat_member_handler(some_my_chat_member_handler)
```

Parameters

- **callback** –
- **custom_filters** –
- **run_task** – run callback in task (no wait results)
- **kwargs** –

my_chat_member_handler(**custom_filters*, *run_task*=None, ***kwargs*)

Decorator for my_chat_member handler

Example:

```
@dp.my_chat_member_handler()
async def some_handler(my_chat_member: types.ChatMemberUpdated)
```

Parameters

- **custom_filters** –
- **run_task** – run callback in task (no wait results)
- **kwargs** –

register_chat_member_handler(*callback*: Callable, **custom_filters*, *run_task*: Optional[bool] = None, ***kwargs*) → None

Register handler for chat_member

Example:

```
dp.register_chat_member_handler(some_chat_member_handler)
```

Parameters

- **callback** –
- **custom_filters** –
- **run_task** – run callback in task (no wait results)
- **kwargs** –

chat_member_handler(**custom_filters*, *run_task*=None, ***kwargs*)

Decorator for chat_member handler

Example:

```
@dp.chat_member_handler()
async def some_handler(chat_member: types.ChatMemberUpdated)
```

Parameters

- **custom_filters** –
- **run_task** – run callback in task (no wait results)

- **kwargs** –

register_errors_handler(*callback, *custom_filters, exception=None, run_task=None, **kwargs*)

Register handler for errors

Parameters

- **callback** –
- **exception** – you can make handler for specific errors type
- **run_task** – run callback in task (no wait results)

errors_handler(**custom_filters, exception=None, run_task=None, **kwargs*)

Decorator for errors handler

Parameters

- **exception** – you can make handler for specific errors type
- **run_task** – run callback in task (no wait results)

Returns

current_state(**, chat: Optional[Union[str, int]] = None, user: Optional[Union[str, int]] = None*) →
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

async throttle(*key, *, rate=None, user_id=None, chat_id=None, no_error=None*) → bool

Execute throttling manager. Returns True if limit has not exceeded otherwise raises ThrottleError or returns False

Parameters

- **key** – key in storage
- **rate** – limit (by default is equal to default rate limit)
- **user_id** – user id
- **chat_id** – chat id
- **no_error** – return boolean value instead of raising error

Returns bool

async check_key(*key, chat_id=None, user_id=None*)

Get information about key in bucket

Parameters

- **key** –
- **chat_id** –
- **user_id** –

Returns

async release_key(*key*, *chat_id=None*, *user_id=None*)

Release blocked key

Parameters

- **key** –
- **chat_id** –
- **user_id** –

Returns

async_task(*func*)

Execute handler as task and return None. Use this decorator for slow handlers (with timeouts)

```
@dp.message_handler(commands=['command'])
@dp.async_task
async def cmd_with_timeout(message: types.Message):
    await asyncio.sleep(120)
    return SendMessage(message.chat.id, 'KABOOM').reply(message)
```

Parameters **func** –**Returns**

throttled(*on_throttled: Optional[Callable] = None*, *key=None*, *rate=None*, *user_id=None*, *chat_id=None*)

Meta-decorator for throttling. Invokes *on_throttled* if the handler was throttled.

Example:

```
async def handler_throttled(message: types.Message, **kwargs):
    await message.answer("Throttled!")

@dp.throttled(handler_throttled)
async def some_handler(message: types.Message):
    await message.answer("Didn't throttled!")
```

Parameters

- **on_throttled** – the callable object that should be either a function or return a coroutine
- **key** – key in storage
- **rate** – limit (by default is equal to default rate limit)
- **user_id** – user id
- **chat_id** – chat id

Returns decorator

bind_filter(*callback*: Union[Callable, aiogram.dispatcher.filters.filters.AbstractFilter], *validator*: Optional[Callable] = None, *event_handlers*: Optional[List[aiogram.dispatcher.handler.Handler]] = None, *exclude_event_handlers*: Optional[Iterable[aiogram.dispatcher.handler.Handler]] = None)

Register filter

Parameters

- **callback** – callable or subclass of AbstractFilter
- **validator** – custom validator.
- **event_handlers** – list of instances of Handler
- **exclude_event_handlers** – list of excluded event handlers (Handler)

unbind_filter(*callback*: Union[Callable, aiogram.dispatcher.filters.filters.AbstractFilter/])

Unregister filter

Parameters **callback** – callable of subclass of AbstractFilter

setup_middleware(*middleware*)

Setup middleware

Parameters **middleware** –

Returns

4.6 Utils

4.6.1 Auth Widget

Implementation of Telegram site authorization checking mechanism for more information <https://core.telegram.org/widgets/login#checking-authorization>

Source: <https://gist.github.com/JrooTJunior/887791de7273c9df5277d2b1ecadc839>

aiogram.utils.auth_widget.**generate_hash**(*data*: dict, *token*: str) → str

Generate secret hash

Parameters

- **data** –
- **token** –

Returns

aiogram.utils.auth_widget.**check_token**(*data*: dict, *token*: str) → bool

Validate auth token

Parameters

- **data** –
- **token** –

Returns

aiogram.utils.auth_widget.**check_signature**(*token*: str, *hash*: str, ***kwargs*) → bool

Generate hexadecimal representation of the HMAC-SHA-256 signature of the data-check-string with the SHA256 hash of the bot's token used as a secret key

Parameters

- **token** –
- **hash** –
- **kwargs** – all params received on auth

Returns

`aiogram.utils.auth_widget.check_integrity(token: str, data: dict) → bool`
 Verify the authentication and the integrity of the data received on user's auth

Parameters

- **token** – Bot's token
- **data** – all data that came on auth

Returns

4.6.2 Executor

`aiogram.utils.executor.start_polling(dispatcher: *, loop=None, skip_updates=False, reset_webhook=True, on_startup=None, on_shutdown=None, timeout=20, relax=0.1, fast=True, allowed_updates: Optional[List[str]] = None)`

Start bot in long-polling mode

Parameters

- **dispatcher** –
- **loop** –
- **skip_updates** –
- **reset_webhook** –
- **on_startup** –
- **on_shutdown** –
- **timeout** –
- **relax** –
- **fast** –
- **allowed_updates** –

`aiogram.utils.executor.set_webhook(dispatcher: aiogram.dispatcher.dispatcher.Dispatcher, webhook_path: str, *, loop: Optional[asyncio.events.AbstractEventLoop] = None, skip_updates: Optional[bool] = None, on_startup: Optional[Callable] = None, on_shutdown: Optional[Callable] = None, check_ip: bool = False, retry_after: Optional[Union[str, int]] = None, route_name: str = 'webhook_handler', web_app: Optional[aiohttp.web.Application] = None)`

Set webhook for bot

Parameters

- **dispatcher** – Dispatcher
- **webhook_path** – str

- **loop** – Optional[asyncio.AbstractEventLoop] (default: None)
- **skip_updates** – bool (default: None)
- **on_startup** – Optional[Callable] (default: None)
- **on_shutdown** – Optional[Callable] (default: None)
- **check_ip** – bool (default: False)
- **retry_after** – Optional[Union[str, int]] See <https://tools.ietf.org/html/rfc7231#section-7.1.3> (default: None)
- **route_name** – str (default: 'webhook_handler')
- **web_app** – Optional[Application] (default: None)

Returns

`aiogram.utils.executor.start_webhook(dispatcher, webhook_path, *, loop=None, skip_updates=None, on_startup=None, on_shutdown=None, check_ip=False, retry_after=None, route_name='webhook_handler', **kwargs)`

Start bot in webhook mode

Parameters

- **dispatcher** –
- **webhook_path** –
- **loop** –
- **skip_updates** –
- **on_startup** –
- **on_shutdown** –
- **check_ip** –
- **route_name** –
- **kwargs** –

Returns

`aiogram.utils.executor.start(dispatcher, future, *, loop=None, skip_updates=None, on_startup=None, on_shutdown=None)`

Execute Future.

Parameters

- **dispatcher** – instance of Dispatcher
- **future** – future
- **loop** – instance of AbstractEventLoop
- **skip_updates** –
- **on_startup** –
- **on_shutdown** –

Returns

`class aiogram.utils.executor.Executor(dispatcher, skip_updates=None, check_ip=False, retry_after=None, loop=None)`

Main executor class

set_web_app(*application: aiohttp.web_app.Application*)

Change instance of aiohttp.web.Application

Parameters *application* –

on_startup(*callback: callable, polling=True, webhook=True*)

Register a callback for the startup process

Parameters

- **callback** –
- **polling** – use with polling
- **webhook** – use with webhook

on_shutdown(*callback: callable, polling=True, webhook=True*)

Register a callback for the shutdown process

Parameters

- **callback** –
- **polling** – use with polling
- **webhook** – use with webhook

set_webhook(*webhook_path: Optional[str] = None, request_handler: Any = <class 'aiogram.dispatcher.webhook.WebhookRequestHandler'>, route_name: str = 'webhook_handler', web_app: Optional[aiohttp.web_app.Application] = None*)

Set webhook for bot

Parameters

- **webhook_path** – Optional[str] (default: None)
- **request_handler** – Any (default: WebhookRequestHandler)
- **route_name** – str Name of webhook handler route (default: 'webhook_handler')
- **web_app** – Optional[Application] (default: None)

Returns

start_webhook(*webhook_path=None, request_handler=<class 'aiogram.dispatcher.webhook.WebhookRequestHandler'>, route_name='webhook_handler', **kwargs*)

Start bot in webhook mode

Parameters

- **webhook_path** –
- **request_handler** –
- **route_name** – Name of webhook handler route
- **kwargs** –

Returns

start_polling(*reset_webhook=None, timeout=20, relax=0.1, fast=True, allowed_updates: Optional[List[str]] = None*)

Start bot in long-polling mode

Parameters

- **reset_webhook** –

- **timeout** –

start(*future*)

Execute Future.

Return the Future's result, or raise its exception.

Parameters **future** –

Returns

4.6.3 Exceptions

- **TelegramAPIError**

- **ValidationError**

- **Throttled**

- **BadRequest**

- * **MessageError**

- **MessageNotModified**

- **MessageToForwardNotFound**

- **MessageIdInvalid**

- **MessageToDeleteNotFound**

- **MessageToPinNotFound**

- **MessageIdentifierNotSpecified**

- **MessageTextIsEmpty**

- **MessageCantBeEdited**

- **MessageCantBeDeleted**

- **MessageCantBeForwarded**

- **MessageToEditNotFound**

- **MessageToReplyNotFound**

- **ToMuchMessages**

- * **PollError**

- **PollCantBeStopped**

- **PollHasAlreadyClosed**

- **PollsCantBeSentToPrivateChats**

- **PollSizeError**

- PollMustHaveMoreOptions**

- PollCantHaveMoreOptions**

- PollsOptionsLengthTooLong**

- PollOptionsMustBeNonEmpty**

- PollQuestionMustBeNonEmpty**

- MessageWithPollNotFound (with MessageError)
- MessageIsNotAPoll (with MessageError)
- * ObjectExpectedAsReplyMarkup
- * InlineKeyboardExpected
- * ChatNotFound
- * ChatDescriptionIsNotModified
- * InvalidQueryID
- * InvalidPeerID
- * InvalidHttpUrlContent
- * ButtonURLInvalid
- * URLHostIsEmpty
- * StartParamInvalid
- * ButtonDataInvalid
- * FileIsTooBig
- * WrongFileIdentifier
- * GroupDeactivated
- * **BadWebhook**
 - WebhookRequireHTTPS
 - BadWebhookPort
 - BadWebhookAddrInfo
 - BadWebhookNoAddressAssociatedWithHostname
- * **NotFound**
 - MethodNotKnown
- * PhotoAsInputFileRequired
- * InvalidStickersSet
- * NoStickerInRequest
- * ChatAdminRequired
- * NeedAdministratorRightsInTheChannel
- * MethodNotAvailableInPrivateChats
- * CantDemoteChatCreator
- * CantRestrictSelf
- * NotEnoughRightsToRestrict
- * PhotoDimensions
- * UnavailableMembers
- * TypeOfFileMismatch
- * WrongRemoteFileIdSpecified

- * `PaymentProviderInvalid`
- * `CurrencyTotalAmountInvalid`
- * `CantParseUrl`
- * `UnsupportedUrlProtocol`
- * `CantParseEntities`
- * `ResultIdDuplicate`
- * `MethodIsNotAvailable`

– **ConflictError**

- * `TerminatedByOtherGetUpdates`
- * `CantGetUpdates`

– **Unauthorized**

- * `BotKicked`
- * `BotBlocked`
- * `UserDeactivated`
- * `CantInitiateConversation`
- * `CantTalkWithBots`

– `NetworkError`

– `RetryAfter`

– `MigrateToChat`

– `RestartingTelegram`

• **AIOGramWarning**

– `TimeoutWarning`

exception `aiogram.utils.exceptions.TelegramAPIError`(*message=None*)

exception `aiogram.utils.exceptions.AIOGramWarning`

exception `aiogram.utils.exceptions.TimeoutWarning`

exception `aiogram.utils.exceptions.FSMStorageWarning`

exception `aiogram.utils.exceptions.ValidationError`(*message=None*)

exception `aiogram.utils.exceptions.BadRequest`(*message=None*)

exception `aiogram.utils.exceptions.MessageError`(*message=None*)

exception `aiogram.utils.exceptions.MessageNotModified`(*message=None*)

Will be raised when you try to set new text is equals to current text.

exception `aiogram.utils.exceptions.MessageToForwardNotFound`(*message=None*)

Will be raised when you try to forward very old or deleted or unknown message.

exception `aiogram.utils.exceptions.MessageIdInvalid`(*message=None*)

exception `aiogram.utils.exceptions.MessageToDeleteNotFound`(*message=None*)

Will be raised when you try to delete very old or deleted or unknown message.

exception aiogram.utils.exceptions.**MessageToPinNotFound**(*message=None*)
 Will be raised when you try to pin deleted or unknown message.

exception aiogram.utils.exceptions.**MessageToReplyNotFound**(*message=None*)
 Will be raised when you try to reply to very old or deleted or unknown message.

exception aiogram.utils.exceptions.**MessageIdentifierNotSpecified**(*message=None*)

exception aiogram.utils.exceptions.**MessageTextIsEmpty**(*message=None*)

exception aiogram.utils.exceptions.**MessageCantBeEdited**(*message=None*)

exception aiogram.utils.exceptions.**MessageCantBeDeleted**(*message=None*)

exception aiogram.utils.exceptions.**MessageCantBeForwarded**(*message=None*)

exception aiogram.utils.exceptions.**MessageToEditNotFound**(*message=None*)

exception aiogram.utils.exceptions.**MessageIsTooLong**(*message=None*)

exception aiogram.utils.exceptions.**ToMuchMessages**(*message=None*)
 Will be raised when you try to send media group with more than 10 items.

exception aiogram.utils.exceptions.**ObjectExpectedAsReplyMarkup**(*message=None*)

exception aiogram.utils.exceptions.**InlineKeyboardExpected**(*message=None*)

exception aiogram.utils.exceptions.**PollError**(*message=None*)

exception aiogram.utils.exceptions.**PollCantBeStopped**(*message=None*)

exception aiogram.utils.exceptions.**PollHasAlreadyBeenClosed**(*message=None*)

exception aiogram.utils.exceptions.**PollsCantBeSentToPrivateChats**(*message=None*)

exception aiogram.utils.exceptions.**PollSizeError**(*message=None*)

exception aiogram.utils.exceptions.**PollMustHaveMoreOptions**(*message=None*)

exception aiogram.utils.exceptions.**PollCantHaveMoreOptions**(*message=None*)

exception aiogram.utils.exceptions.**PollOptionsMustBeNonEmpty**(*message=None*)

exception aiogram.utils.exceptions.**PollQuestionMustBeNonEmpty**(*message=None*)

exception aiogram.utils.exceptions.**PollOptionsLengthTooLong**(*message=None*)

exception aiogram.utils.exceptions.**PollQuestionLengthTooLong**(*message=None*)

exception aiogram.utils.exceptions.**PollCanBeRequestedInPrivateChatsOnly**(*message=None*)

exception aiogram.utils.exceptions.**MessageWithPollNotFound**(*message=None*)
 Will be raised when you try to stop poll with message without poll

exception aiogram.utils.exceptions.**MessageIsNotAPoll**(*message=None*)
 Will be raised when you try to stop poll with message without poll

exception aiogram.utils.exceptions.**ChatNotFound**(*message=None*)

exception aiogram.utils.exceptions.**ChatIdIsEmpty**(*message=None*)

exception aiogram.utils.exceptions.**InvalidUserId**(*message=None*)

exception aiogram.utils.exceptions.**ChatDescriptionIsNotModified**(*message=None*)

exception aiogram.utils.exceptions.**InvalidQueryID**(*message=None*)

exception aiogram.utils.exceptions.**InvalidPeerID**(*message=None*)

exception aiogram.utils.exceptions.**InvalidHttpRequestContent**(*message=None*)

exception aiogram.utils.exceptions.ButtonURLInvalid(message=None)

exception aiogram.utils.exceptions.URLHostIsEmpty(message=None)

exception aiogram.utils.exceptions.StartParamInvalid(message=None)

exception aiogram.utils.exceptions.ButtonDataInvalid(message=None)

exception aiogram.utils.exceptions.FileIsTooBig(message=None)

exception aiogram.utils.exceptions.WrongFileIdentifier(message=None)

exception aiogram.utils.exceptions.GroupDeactivated(message=None)

exception aiogram.utils.exceptions.PhotoAsInputFileRequired(message=None)
Will be raised when you try to set chat photo from file ID.

exception aiogram.utils.exceptions.InvalidStickersSet(message=None)

exception aiogram.utils.exceptions.NoStickerInRequest(message=None)

exception aiogram.utils.exceptions.ChatAdminRequired(message=None)

exception aiogram.utils.exceptions.NeedAdministratorRightsInTheChannel(message=None)

exception aiogram.utils.exceptions.NotEnoughRightsToPinMessage(message=None)

exception aiogram.utils.exceptions.MethodNotAvailableInPrivateChats(message=None)

exception aiogram.utils.exceptions.CantDemoteChatCreator(message=None)

exception aiogram.utils.exceptions.CantRestrictSelf(message=None)

exception aiogram.utils.exceptions.NotEnoughRightsToRestrict(message=None)

exception aiogram.utils.exceptions.PhotoDimensions(message=None)

exception aiogram.utils.exceptions.UnavailableMembers(message=None)

exception aiogram.utils.exceptions.TypeOfFileMismatch(message=None)

exception aiogram.utils.exceptions.WrongRemoteFileIdSpecified(message=None)

exception aiogram.utils.exceptions.PaymentProviderInvalid(message=None)

exception aiogram.utils.exceptions.CurrencyTotalAmountInvalid(message=None)

exception aiogram.utils.exceptions.BadWebhook(message=None)

exception aiogram.utils.exceptions.WebhookRequireHTTPS(message=None)

exception aiogram.utils.exceptions.BadWebhookPort(message=None)

exception aiogram.utils.exceptions.BadWebhookAddrInfo(message=None)

exception aiogram.utils.exceptions.BadWebhookNoAddressAssociatedWithHostname(message=None)

exception aiogram.utils.exceptions.CantParseUrl(message=None)

exception aiogram.utils.exceptions.UnsupportedUrlProtocol(message=None)

exception aiogram.utils.exceptions.CantParseEntities(message=None)

exception aiogram.utils.exceptions.ResultIdDuplicate(message=None)

exception aiogram.utils.exceptions.BotDomainInvalid(message=None)

exception aiogram.utils.exceptions.MethodIsNotAvailable(message=None)

exception aiogram.utils.exceptions.CantRestrictChatOwner(message=None)
Raises when bot restricts the chat owner

exception aiogram.utils.exceptions.**UserIsAnAdministratorOfTheChat**(*message=None*)
 Raises when bot restricts the chat admin

exception aiogram.utils.exceptions.**NotFound**(*message=None*)

exception aiogram.utils.exceptions.**MethodNotKnown**(*message=None*)

exception aiogram.utils.exceptions.**ConflictError**(*message=None*)

exception aiogram.utils.exceptions.**TerminatedByOtherGetUpdates**(*message=None*)

exception aiogram.utils.exceptions.**CantGetUpdates**(*message=None*)

exception aiogram.utils.exceptions.**Unauthorized**(*message=None*)

exception aiogram.utils.exceptions.**BotKicked**(*message=None*)

exception aiogram.utils.exceptions.**BotBlocked**(*message=None*)

exception aiogram.utils.exceptions.**UserDeactivated**(*message=None*)

exception aiogram.utils.exceptions.**CantInitiateConversation**(*message=None*)

exception aiogram.utils.exceptions.**CantTalkWithBots**(*message=None*)

exception aiogram.utils.exceptions.**NetworkError**(*message=None*)

exception aiogram.utils.exceptions.**RestartingTelegram**

exception aiogram.utils.exceptions.**RetryAfter**(*retry_after*)

exception aiogram.utils.exceptions.**MigrateToChat**(*chat_id*)

exception aiogram.utils.exceptions.**Throttled**(***kwargs*)

4.6.4 Markdown

aiogram.utils.markdown.**quote_html**(**content, sep=' '*) → str
 Quote HTML symbols

All <, >, & and " symbols that are not a part of a tag or an HTML entity must be replaced with the corresponding HTML entities (< with < > with > & with & and " with ").

Parameters

- **content** –
- **sep** –

Returns

aiogram.utils.markdown.**escape_md**(**content, sep=' '*) → str
 Escape markdown text

E.g. for usernames

Parameters

- **content** –
- **sep** –

Returns

aiogram.utils.markdown.**text**(**content, sep=' '*)
 Join all elements with a separator

Parameters

- **content** –

- **sep** –

Returns

`aiogram.utils.markdown.bold(*content, sep=' ') → str`

Make bold text (Markdown)

Parameters

- **content** –
- **sep** –

Returns

`aiogram.utils.markdown.hbold(*content, sep=' ') → str`

Make bold text (HTML)

Parameters

- **content** –
- **sep** –

Returns

`aiogram.utils.markdown.italic(*content, sep=' ') → str`

Make italic text (Markdown)

Parameters

- **content** –
- **sep** –

Returns

`aiogram.utils.markdown.hitalic(*content, sep=' ') → str`

Make italic text (HTML)

Parameters

- **content** –
- **sep** –

Returns

`aiogram.utils.markdown.code(*content, sep=' ') → str`

Make mono-width text (Markdown)

Parameters

- **content** –
- **sep** –

Returns

`aiogram.utils.markdown.hcode(*content, sep=' ') → str`

Make mono-width text (HTML)

Parameters

- **content** –
- **sep** –

Returns

`aiogram.utils.markdown.pre(*content, sep='\n') → str`

Make mono-width text block (Markdown)

Parameters

- **content** –
- **sep** –

Returns

`aiogram.utils.markdown.hpre(*content, sep='\n') → str`
Make mono-width text block (HTML)

Parameters

- **content** –
- **sep** –

Returns

`aiogram.utils.markdown.underline(*content, sep=' ') → str`
Make underlined text (Markdown)

Parameters

- **content** –
- **sep** –

Returns

`aiogram.utils.markdown.hunderline(*content, sep=' ') → str`
Make underlined text (HTML)

Parameters

- **content** –
- **sep** –

Returns

`aiogram.utils.markdown.strikethrough(*content, sep=' ') → str`
Make strikethrough text (Markdown)

Parameters

- **content** –
- **sep** –

Returns

`aiogram.utils.markdown.hstrikethrough(*content, sep=' ') → str`
Make strikethrough text (HTML)

Parameters

- **content** –
- **sep** –

Returns

`aiogram.utils.markdown.link(title: str, url: str) → str`
Format URL (Markdown)

Parameters

- **title** –
- **url** –

Returns

`aiogram.utils.markdown.hlink(title: str, url: str) → str`
Format URL (HTML)

Parameters

- **title** –
- **url** –

Returns

`aiogram.utils.markdown.hide_link(url: str) → str`
Hide URL (HTML only) Can be used for adding an image to a text message
Parameters `url` –
Returns

4.6.5 Helper

Example:

```
>>> from aiogram.utils.helper import Helper, ListItem, HelperMode, Item
>>> class MyHelper(Helper):
...     mode = HelperMode.lowerCamelCase
...     FOO_ITEM = ListItem()
...     BAR_ITEM = ListItem()
...     BAZ_ITEM = ListItem()
...     LOREM = Item()
...
>>> print(MyHelper.FOO_ITEM & MyHelper.BAR_ITEM)
<<< ['fooItem', 'barItem']
>>> print(MyHelper.all())
<<< ['barItem', 'bazItem', 'fooItem', 'lorem']
```

class `aiogram.utils.helper.Item(value=None)`
Helper item

If a value is not provided, it will be automatically generated based on a variable's name

class `aiogram.utils.helper.ListItem(value=None)`
This item is always a list

You can use `&`, `|` and `+` operators for that.

class `aiogram.utils.helper.ItemsList(*seq)`
Patch for default list

This class provides `+`, `&`, `|`, `+=`, `&=`, `|=` operators for extending the list

4.6.6 Deprecated

`aiogram.utils.deprecated.deprecated(reason, stacklevel=2) → Callable`
This is a decorator which can be used to mark functions as deprecated. It will result in a warning being emitted when the function is used.

Source: <https://stackoverflow.com/questions/2536307/decorators-in-the-python-standard-lib-deprecated-specifically>

`aiogram.utils.deprecated.renamed_argument(old_name: str, new_name: str, until_version: str, stacklevel: int = 3)`

A meta-decorator to mark an argument as deprecated.

```
@renamed_argument("chat", "chat_id", "3.0") # stacklevel=3 by default
@renamed_argument("user", "user_id", "3.0", stacklevel=4)
def some_function(user_id, chat_id=None):
    print(f"user_id={user_id}, chat_id={chat_id}")

some_function(user=123) # prints 'user_id=123, chat_id=None' with warning
```

(continues on next page)

(continued from previous page)

```
some_function(123) # prints 'user_id=123, chat_id=None' without warning
some_function(user_id=123) # prints 'user_id=123, chat_id=None' without warning
```

Parameters

- **old_name** –
- **new_name** –
- **until_version** – the version in which the argument is scheduled to be removed
- **stacklevel** – leave it to default if it's the first decorator used.

Increment with any new decorator used. :return: decorator

```
class aiogram.utils.deprecated.DeprecatedReadOnlyClassVar(warning_message: str,
                                                         new_value_getter:
                                                         Callable[[aiogram.utils.deprecated._OwnerCls],
                                                         aiogram.utils.deprecated._VT])
```

DeprecatedReadOnlyClassVar[Owner, ValueType]

Parameters

- **warning_message** – Warning message when getter gets called
- **new_value_getter** – Any callable with (owner_class: Type[Owner]) -> ValueType signature that will be executed

Usage example:

```
>>> class MyClass:
...     some_attribute: DeprecatedReadOnlyClassVar[MyClass, int] = ...
...     DeprecatedReadOnlyClassVar(
...         "Warning message.", lambda owner: 15)
...
>>> MyClass.some_attribute # does warning.warn with `Warning message` and returns
... 15 in the current case
```

4.6.7 Payload

aiogram.utils.payload.generate_payload(exclude=None, **kwargs)

Generate payload

Usage: payload = generate_payload(**locals(), exclude=['foo'])

Parameters

- **exclude** –
- **kwargs** –

Returns dict

aiogram.utils.payload.prepare_arg(value)

Stringify dicts/lists and convert datetime/timedelta to unix-time

Parameters value –

Returns

4.6.8 Parts

`aiogram.utils.parts.split_text(text: str, length: int = 4096) → List[str]`

Split long text

Parameters

- **text** –
- **length** –

Returns list of parts

Return type `typing.List[str]`

`aiogram.utils.parts.safe_split_text(text: str, length: int = 4096, split_separator: str = ' ') → List[str]`

Split long text

Parameters

- **text** –
- **length** –

:param split_separator :return:

`aiogram.utils.parts.paginate(data: Iterable, page: int = 0, limit: int = 10) → Iterable`

Slice data over pages

Parameters

- **data** (`typing.Iterable`) – any iterable object
- **page** (`int`) – number of page
- **limit** (`int`) – items per page

Returns sliced object

Return type `typing.Iterable`

4.6.9 JSON

4.6.10 Emoji

4.6.11 Deep linking

Deep linking

Telegram bots have a deep linking mechanism, that allows for passing additional parameters to the bot on startup. It could be a command that launches the bot — or an auth token to connect the user's Telegram account to their account on some external service.

You can read detailed description in the source: <https://core.telegram.org/bots#deep-linking>

We have add some utils to get deep links more handy.

Basic link example:

```
from aiogram.utils.deep_linking import get_start_link
link = await get_start_link('foo')

# result: 'https://t.me/MyBot?start=foo'
```

Encoded link example:

```
from aiogram.utils.deep_linking import get_start_link

link = await get_start_link('foo', encode=True)
# result: 'https://t.me/MyBot?start=Zm9v'
```

Decode it back example:

```
from aiogram.utils.deep_linking import decode_payload
from aiogram.types import Message

@dp.message_handler(commands=["start"])
async def handler(message: Message):
    args = message.get_args()
    payload = decode_payload(args)
    await message.answer(f"Your payload: {payload}")
```

async aiogram.utils.deep_linking.get_start_link(payload: str, encode=False) → str

Get ‘start’ deep link with your payload.

If you need to encode payload or pass special characters - set encode as True

Parameters

- **payload** – args passed with /start
- **encode** – encode payload with base64url

Returns link

async aiogram.utils.deep_linking.get_startgroup_link(payload: str, encode=False) → str

Get ‘startgroup’ deep link with your payload.

If you need to encode payload or pass special characters - set encode as True

Parameters

- **payload** – args passed with /start
- **encode** – encode payload with base64url

Returns link

aiogram.utils.deep_linking.encode_payload(payload: str) → str

Encode payload with URL-safe base64url.

aiogram.utils.deep_linking.decode_payload(payload: str) → str

Decode payload with URL-safe base64url.

4.7 Examples

4.7.1 Echo bot

Listing 1: echo_bot.py

```
1 """
2 This is a echo bot.
3 It echoes any incoming text messages.
4 """
5
```

(continues on next page)

```

6  import logging
7
8  from aiogram import Bot, Dispatcher, executor, types
9
10 API_TOKEN = 'BOT TOKEN HERE'
11
12 # Configure logging
13 logging.basicConfig(level=logging.INFO)
14
15 # Initialize bot and dispatcher
16 bot = Bot(token=API_TOKEN)
17 dp = Dispatcher(bot)
18
19
20 @dp.message_handler(commands=['start', 'help'])
21 async def send_welcome(message: types.Message):
22     """
23     This handler will be called when user sends `/start` or `/help` command
24     """
25     await message.reply("Hi!\nI'm EchoBot!\nPowered by aiogram.")
26
27
28 @dp.message_handler(regexp='(^cat[s]?$|puss)')
29 async def cats(message: types.Message):
30     with open('data/cats.jpg', 'rb') as photo:
31         """
32         # Old fashioned way:
33         await bot.send_photo(
34             message.chat.id,
35             photo,
36             caption='Cats are here ',
37             reply_to_message_id=message.message_id,
38         )
39         """
40
41     await message.reply_photo(photo, caption='Cats are here ')
42
43
44 @dp.message_handler()
45 async def echo(message: types.Message):
46     # old style:
47     # await bot.send_message(message.chat.id, message.text)
48
49     await message.answer(message.text)
50
51
52 if __name__ == '__main__':
53     executor.start_polling(dp, skip_updates=True)

```


4.7.2 Inline bot

Listing 2: inline_bot.py

```

1 import hashlib
2 import logging
3
4 from aiogram import Bot, Dispatcher, executor
5 from aiogram.types import InlineQuery, \
6     InputTextMessageContent, InlineQueryResultArticle
7
8 API_TOKEN = 'BOT_TOKEN_HERE'
9
10 logging.basicConfig(level=logging.DEBUG)
11
12 bot = Bot(token=API_TOKEN)
13 dp = Dispatcher(bot)
14
15
16 @dp.inline_handler()
17 async def inline_echo(inline_query: InlineQuery):
18     # id affects both preview and content,
19     # so it has to be unique for each result
20     # (Unique identifier for this result, 1-64 Bytes)
21     # you can set your unique id's
22     # but for example i'll generate it based on text because I know, that
23     # only text will be passed in this example
24     text = inline_query.query or 'echo'
25     input_content = InputTextMessageContent(text)
26     result_id: str = hashlib.md5(text.encode()).hexdigest()
27     item = InlineQueryResultArticle(
28         id=result_id,
29         title=f'Result {text!r}',
30         input_message_content=input_content,
31     )
32     # don't forget to set cache_time=1 for testing (default is 300s or 5m)
33     await bot.answer_inline_query(inline_query.id, results=[item], cache_time=1)
34
35
36 if __name__ == '__main__':
37     executor.start_polling(dp, skip_updates=True)

```

4.7.3 Advanced executor example

Listing 3: advanced_executor_example.py

```

1 #!/usr/bin/env python3
2 """
3 **This example is outdated**
4 In this example used ArgumentParser for configuring Your bot.
5
6 Provided to start bot with webhook:

```

(continues on next page)

(continued from previous page)

```

7     python advanced_executor_example.py \
8         --token TOKEN_HERE \
9         --host 0.0.0.0 \
10        --port 8084 \
11        --host-name example.com \
12        --webhook-port 443
13
14 Or long polling:
15     python advanced_executor_example.py --token TOKEN_HERE
16
17 So... In this example found small trouble:
18     can't get bot instance in handlers.
19
20
21 If you want to automatic change getting updates method use executor utils (from aiogram.
22 ↪utils.executor)
23 """
24 # TODO: Move token to environment variables.
25
26 import argparse
27 import logging
28 import ssl
29 import sys
30
31 from aiogram import Bot
32 from aiogram.dispatcher import Dispatcher
33 from aiogram.dispatcher.webhook import *
34 from aiogram.utils.executor import start_polling, start_webhook
35
36 logging.basicConfig(level=logging.INFO)
37
38 # Configure arguments parser.
39 parser = argparse.ArgumentParser(description='Python telegram bot')
40 parser.add_argument('--token', '-t', nargs='?', type=str, default=None, help='Set_
41 ↪working directory')
42 parser.add_argument('--sock', help='UNIX Socket path')
43 parser.add_argument('--host', help='Webserver host')
44 parser.add_argument('--port', type=int, help='Webserver port')
45 parser.add_argument('--cert', help='Path to SSL certificate')
46 parser.add_argument('--pkey', help='Path to SSL private key')
47 parser.add_argument('--host-name', help='Set webhook host name')
48 parser.add_argument('--webhook-port', type=int, help='Port for webhook (default=port)')
49 parser.add_argument('--webhook-path', default='/webhook', help='Port for webhook_
50 ↪(default=port)')
51
52
53
54 async def cmd_start(message: types.Message):
55     return SendMessage(message.chat.id, f"Hello, {message.from_user.full_name}!")
56
57
58 def setup_handlers(dispatcher: Dispatcher):
59     # This example has only one messages handler

```

(continues on next page)

(continued from previous page)

```

56     dispatcher.register_message_handler(cmd_start, commands=['start', 'welcome'])
57
58
59     async def on_startup(dispatcher, url=None, cert=None):
60         setup_handlers(dispatcher)
61
62         bot = dispatcher.bot
63
64         # Get current webhook status
65         webhook = await bot.get_webhook_info()
66
67         if url:
68             # If URL is bad
69             if webhook.url != url:
70                 # If URL doesnt match with by current remove webhook
71                 if not webhook.url:
72                     await bot.delete_webhook()
73
74                 # Set new URL for webhook
75                 if cert:
76                     with open(cert, 'rb') as cert_file:
77                         await bot.set_webhook(url, certificate=cert_file)
78                 else:
79                     await bot.set_webhook(url)
80             elif webhook.url:
81                 # Otherwise remove webhook.
82                 await bot.delete_webhook()
83
84
85     async def on_shutdown(dispatcher):
86         print('Shutdown.')
87
88
89     def main(arguments):
90         args = parser.parse_args(arguments)
91         token = args.token
92         sock = args.sock
93         host = args.host
94         port = args.port
95         cert = args.cert
96         pkey = args.pkey
97         host_name = args.host_name or host
98         webhook_port = args.webhook_port or port
99         webhook_path = args.webhook_path
100
101         # Fi webhook path
102         if not webhook_path.startswith('/'):
103             webhook_path = '/' + webhook_path
104
105         # Generate webhook URL
106         webhook_url = f"https://{host_name}:{webhook_port}{webhook_path}"
107

```

(continues on next page)

(continued from previous page)

```

108     # Create bot & dispatcher instances.
109     bot = Bot(token)
110     dispatcher = Dispatcher(bot)
111
112     if (sock or host) and host_name:
113         if cert and pkey:
114             ssl_context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)
115             ssl_context.load_cert_chain(cert, pkey)
116         else:
117             ssl_context = None
118
119         start_webhook(dispatcher, webhook_path,
120                       on_startup=functools.partial(on_startup, url=webhook_url,
121 ↪ cert=cert),
122                       on_shutdown=on_shutdown,
123                       host=host, port=port, path=sock, ssl_context=ssl_context)
124     else:
125         start_polling(dispatcher, on_startup=on_startup, on_shutdown=on_shutdown)
126
127 if __name__ == '__main__':
128     argv = sys.argv[1:]
129
130     if not len(argv):
131         parser.print_help()
132         sys.exit(1)
133
134     main(argv)

```

4.7.4 Proxy and emoji

Listing 4: proxy_and_emoji.py

```

1  import logging
2
3  import aiohttp
4
5  from aiogram import Bot, types
6  from aiogram.dispatcher import Dispatcher
7  from aiogram.types import ParseMode
8  from aiogram.utils.emoji import emojiize
9  from aiogram.utils.executor import start_polling
10 from aiogram.utils.markdown import bold, code, italic, text
11
12 # Configure bot here
13 API_TOKEN = 'BOT_TOKEN_HERE'
14 PROXY_URL = 'http://PROXY_URL' # Or 'socks5://host:port'
15
16 # NOTE: If authentication is required in your proxy then uncomment next line and change
17 ↪ login/password for it

```

(continues on next page)

(continued from previous page)

```

17 # PROXY_AUTH = aiohttp.BasicAuth(login='login', password='password')
18 # And add `proxy_auth=PROXY_AUTH` argument in line 30, like this:
19 # >>> bot = Bot(token=API_TOKEN, proxy=PROXY_URL, proxy_auth=PROXY_AUTH)
20 # Also you can use Socks5 proxy but you need manually install aiohttp-socks package.
21
22 # Get my ip URL
23 GET_IP_URL = 'http://bot.whatismyipaddress.com/'
24
25 logging.basicConfig(level=logging.INFO)
26
27 bot = Bot(token=API_TOKEN, proxy=PROXY_URL)
28
29 # If auth is required:
30 # bot = Bot(token=API_TOKEN, proxy=PROXY_URL, proxy_auth=PROXY_AUTH)
31 dp = Dispatcher(bot)
32
33
34 async def fetch(url, session):
35     async with session.get(url) as response:
36         return await response.text()
37
38
39 @dp.message_handler(commands=['start'])
40 async def cmd_start(message: types.Message):
41     # fetching urls will take some time, so notify user that everything is OK
42     await types.ChatActions.typing()
43
44     content = []
45
46     # Make request (without proxy)
47     async with aiohttp.ClientSession() as session:
48         ip = await fetch(GET_IP_URL, session)
49         content.append(text(':globe_showing_Americas:', bold('IP:'), code(ip)))
50         # This line is formatted to ' *IP:* `YOUR IP` '
51
52     # Make request through bot's proxy
53     ip = await fetch(GET_IP_URL, bot.session)
54     content.append(text(':locked_with_key:', bold('IP:'), code(ip), italic('via proxy'))))
55     # This line is formatted to ' *IP:* `YOUR IP` _via proxy_'
56
57     # Send content
58     await bot.send_message(message.chat.id, emojiize(text(*content, sep='\n')), parse_
↳ mode=ParseMode.MARKDOWN)
59
60     # In this example you can see emoji codes: ":globe_showing_Americas:" and ":locked_
↳ with_key:"
61     # You can find full emoji cheat sheet at https://www.webpagefx.com/tools/emoji-cheat-
↳ sheet/
62     # For representing emoji codes into real emoji use emoji util (aiogram.utils.emoji)
63     # (you have to install emoji module)
64
65     # For example emojiize('Moon face :new_moon_face:') is transformed to 'Moon face '

```

(continues on next page)

(continued from previous page)

```

66
67
68 if __name__ == '__main__':
69     start_polling(dp, skip_updates=True)

```

4.7.5 Finite state machine example

Listing 5: finite_state_machine_example.py

```

1  import logging
2
3  import aiogram.utils.markdown as md
4  from aiogram import Bot, Dispatcher, types
5  from aiogram.contrib.fsm_storage.memory import MemoryStorage
6  from aiogram.dispatcher import FSMContext
7  from aiogram.dispatcher.filters import Text
8  from aiogram.dispatcher.filters.state import State, StatesGroup
9  from aiogram.types import ParseMode
10 from aiogram.utils import executor
11
12 logging.basicConfig(level=logging.INFO)
13
14 API_TOKEN = 'BOT TOKEN HERE'
15
16
17 bot = Bot(token=API_TOKEN)
18
19 # For example use simple MemoryStorage for Dispatcher.
20 storage = MemoryStorage()
21 dp = Dispatcher(bot, storage=storage)
22
23
24 # States
25 class Form(StatesGroup):
26     name = State() # Will be represented in storage as 'Form:name'
27     age = State() # Will be represented in storage as 'Form:age'
28     gender = State() # Will be represented in storage as 'Form:gender'
29
30
31 @dp.message_handler(commands='start')
32 async def cmd_start(message: types.Message):
33     """
34     Conversation's entry point
35     """
36     # Set state
37     await Form.name.set()
38
39     await message.reply("Hi there! What's your name?")
40
41

```

(continues on next page)

(continued from previous page)

```

42 # You can use state '*' if you need to handle all states
43 @dp.message_handler(state='*', commands='cancel')
44 @dp.message_handler(Text(equals='cancel', ignore_case=True), state='*')
45 async def cancel_handler(message: types.Message, state: FSMContext):
46     """
47     Allow user to cancel any action
48     """
49     current_state = await state.get_state()
50     if current_state is None:
51         return
52
53     logging.info('Cancelling state %r', current_state)
54     # Cancel state and inform user about it
55     await state.finish()
56     # And remove keyboard (just in case)
57     await message.reply('Cancelled.', reply_markup=types.ReplyKeyboardRemove())
58
59
60 @dp.message_handler(state=Form.name)
61 async def process_name(message: types.Message, state: FSMContext):
62     """
63     Process user name
64     """
65     async with state.proxy() as data:
66         data['name'] = message.text
67
68     await Form.next()
69     await message.reply("How old are you?")
70
71
72 # Check age. Age gotta be digit
73 @dp.message_handler(lambda message: not message.text.isdigit(), state=Form.age)
74 async def process_age_invalid(message: types.Message):
75     """
76     If age is invalid
77     """
78     return await message.reply("Age gotta be a number.\nHow old are you? (digits only)")
79
80
81 @dp.message_handler(lambda message: message.text.isdigit(), state=Form.age)
82 async def process_age(message: types.Message, state: FSMContext):
83     # Update state and data
84     await Form.next()
85     await state.update_data(age=int(message.text))
86
87     # Configure ReplyKeyboardMarkup
88     markup = types.ReplyKeyboardMarkup(resize_keyboard=True, selective=True)
89     markup.add("Male", "Female")
90     markup.add("Other")
91
92     await message.reply("What is your gender?", reply_markup=markup)
93

```

(continues on next page)

(continued from previous page)

```

94
95 @dp.message_handler(lambda message: message.text not in ["Male", "Female", "Other"],
96                      state=Form.gender)
97
98     """
99     In this example gender has to be one of: Male, Female, Other.
100     """
101
102     return await message.reply("Bad gender name. Choose your gender from the keyboard.")
103
104 @dp.message_handler(state=Form.gender)
105 async def process_gender(message: types.Message, state: FSMContext):
106     async with state.proxy() as data:
107         data['gender'] = message.text
108
109         # Remove keyboard
110         markup = types.ReplyKeyboardRemove()
111
112         # And send message
113         await bot.send_message(
114             message.chat.id,
115             md.text(
116                 md.text('Hi! Nice to meet you,', md.bold(data['name'])),
117                 md.text('Age:', md.code(data['age'])),
118                 md.text('Gender:', data['gender']),
119                 sep='\n',
120             ),
121             reply_markup=markup,
122             parse_mode=ParseMode.MARKDOWN,
123         )
124
125         # Finish conversation
126         await state.finish()
127
128 if __name__ == '__main__':
129     executor.start_polling(dp, skip_updates=True)

```

4.7.6 Throttling example

Listing 6: throttling_example.py

```

1  """
2  Example for throttling manager.
3
4  You can use that for flood controlling.
5  """
6
7  import logging
8

```

(continues on next page)

(continued from previous page)

```

9  from aiogram import Bot, types
10 from aiogram.contrib.fsm_storage.memory import MemoryStorage
11 from aiogram.dispatcher import Dispatcher
12 from aiogram.utils.exceptions import Throttled
13 from aiogram.utils.executor import start_polling
14
15
16 API_TOKEN = 'BOT_TOKEN_HERE'
17
18 logging.basicConfig(level=logging.INFO)
19
20 bot = Bot(token=API_TOKEN)
21
22 # Throttling manager does not work without Leaky Bucket.
23 # You need to use a storage. For example use simple in-memory storage.
24 storage = MemoryStorage()
25 dp = Dispatcher(bot, storage=storage)
26
27
28 @dp.message_handler(commands=['start'])
29 async def send_welcome(message: types.Message):
30     try:
31         # Execute throttling manager with rate-limit equal to 2 seconds for key "start"
32         await dp.throttle('start', rate=2)
33     except Throttled:
34         # If request is throttled, the `Throttled` exception will be raised
35         await message.reply('Too many requests!')
36     else:
37         # Otherwise do something
38         await message.reply("Hi!\nI'm EchoBot!\nPowered by aiogram.")
39
40
41 @dp.message_handler(commands=['hi'])
42 @dp.throttled(lambda msg, loop, *args, **kwargs: loop.create_task(bot.send_message(msg.
43     ↪ from_user.id, "Throttled")),
44     rate=5)
45 # loop is added to the function to run coroutines from it
46 async def say_hi(message: types.Message):
47     await message.answer("Hi")
48
49 # the on_throttled object can be either a regular function or coroutine
50 async def hello_throttled(*args, **kwargs):
51     # args will be the same as in the original handler
52     # kwargs will be updated with parameters given to .throttled (rate, key, user_id, ↪
53     ↪ chat_id)
54     print(f"hello_throttled was called with args={args} and kwargs={kwargs}")
55     message = args[0] # as message was the first argument in the original handler
56     await message.answer("Throttled")
57
58 @dp.message_handler(commands=['hello'])

```

(continues on next page)

(continued from previous page)

```

59 @dp.throttled(hello_throttled, rate=4)
60 async def say_hello(message: types.Message):
61     await message.answer("Hello!")
62
63
64 @dp.message_handler(commands=['help'])
65 @dp.throttled(rate=5)
66 # nothing will happen if the handler will be throttled
67 async def help_handler(message: types.Message):
68     await message.answer('Help!')
69
70 if __name__ == '__main__':
71     start_polling(dp, skip_updates=True)

```

4.7.7 I18n example

Listing 7: i18n_example.py

```

1  """
2  Internationalize your bot
3
4  Step 1: extract texts
5      # pybabel extract --input-dirs=. -o locales/mybot.pot
6
7      Some useful options:
8      - Extract texts with pluralization support
9      # -k __:1,2
10     - Add comments for translators, you can use another tag if you want (TR)
11     # --add-comments=NOTE
12     - Disable comments with string location in code
13     # --no-location
14     - Set project name
15     # --project=MySuperBot
16     - Set version
17     # --version=2.2
18
19 Step 2: create *.po files. E.g. create en, ru, uk locales.
20     # pybabel init -i locales/mybot.pot -d locales -D mybot -l en
21     # pybabel init -i locales/mybot.pot -d locales -D mybot -l ru
22     # pybabel init -i locales/mybot.pot -d locales -D mybot -l uk
23
24 Step 3: translate texts located in locales/{language}/LC_MESSAGES/mybot.po
25     To open .po file you can use basic text editor or any PO editor, e.g. https://poedit.
26     ↪net/
27
28 Step 4: compile translations
29     # pybabel compile -d locales -D mybot
30
31 Step 5: When you change the code of your bot you need to update po & mo files.
32     Step 5.1: regenerate pot file:

```

(continues on next page)

(continued from previous page)

```

32     command from step 1
33     Step 5.2: update po files
34     # pybabel update -d locales -D mybot -i locales/mybot.pot
35     Step 5.3: update your translations
36     location and tools you know from step 3
37     Step 5.4: compile mo files
38     command from step 4
39     """
40
41     from pathlib import Path
42
43     from aiogram import Bot, Dispatcher, executor, types
44     from aiogram.contrib.middlewares.i18n import I18nMiddleware
45
46     TOKEN = 'BOT_TOKEN_HERE'
47     I18N_DOMAIN = 'mybot'
48
49     BASE_DIR = Path(__file__).parent
50     LOCALES_DIR = BASE_DIR / 'locales'
51
52     bot = Bot(TOKEN, parse_mode=types.ParseMode.HTML)
53     dp = Dispatcher(bot)
54
55     # Setup i18n middleware
56     i18n = I18nMiddleware(I18N_DOMAIN, LOCALES_DIR)
57     dp.middleware.setup(i18n)
58
59     # Alias for gettext method
60     _ = i18n.gettext
61
62
63     @dp.message_handler(commands='start')
64     async def cmd_start(message: types.Message):
65         # Simply use `_(message)` instead of `message` and never use f-strings for
66         ↳ translatable texts.
67         await message.reply(_('Hello, <b>{user}</b>!').format(user=message.from_user.full_
68         ↳ name))
69
70     @dp.message_handler(commands='lang')
71     async def cmd_lang(message: types.Message, locale):
72         # For setting custom lang you have to modify i18n middleware
73         await message.reply(_('Your current language: <i>{language}</i>').
74         ↳ format(language=locale))
75
76     # If you care about pluralization, here's small handler
77     # And also, there's an example of comments for translators. Most translation tools
78     ↳ support them.
79
80     # Alias for gettext method, parser will understand double underscore as plural (aka
81     ↳ ngettext)
82     __ = i18n.gettext

```

(continues on next page)

(continued from previous page)

```

79
80
81 # some likes manager
82 LIKES_STORAGE = {'count': 0}
83
84
85 def get_likes() -> int:
86     return LIKES_STORAGE['count']
87
88
89 def increase_likes() -> int:
90     LIKES_STORAGE['count'] += 1
91     return get_likes()
92
93
94 @dp.message_handler(commands='like')
95 async def cmd_like(message: types.Message, locale):
96     likes = increase_likes()
97
98     # NOTE: This is comment for a translator
99     await message.reply(__('Aiogram has {number} like!', 'Aiogram has {number} likes!',
100 ↪ likes).format(number=likes))
101
102 if __name__ == '__main__':
103     executor.start_polling(dp, skip_updates=True)

```

4.7.8 Regexp commands filter example

Listing 8: regexp_commands_filter_example.py

```

1 from aiogram import Bot, types
2 from aiogram.dispatcher import Dispatcher, filters
3 from aiogram.utils import executor
4
5
6 bot = Bot(token='BOT_TOKEN_HERE', parse_mode=types.ParseMode.HTML)
7 dp = Dispatcher(bot)
8
9
10 @dp.message_handler(filters.RegexpCommandsFilter(regexp_commands=['item_([0-9]*)']))
11 async def send_welcome(message: types.Message, regexp_command):
12     await message.reply(f"You have requested an item with id <code>{regexp_command.
13 ↪ group(1)}</code>")
14
15 @dp.message_handler(commands='start')
16 async def create_deeplink(message: types.Message):
17     bot_user = await bot.me
18     bot_username = bot_user.username

```

(continues on next page)

(continued from previous page)

```

19     deeplink = f'https://t.me/{bot_username}?start=item_12345'
20     text = (
21         f'Either send a command /item_1234 or follow this link {deeplink} and then click ↩️',
22         '↩️start\n'
23         'It also can be hidden in a inline button\n\n'
24         'Or just send <code>/start item_123</code>'
25     )
26     await message.reply(text, disable_web_page_preview=True)
27
28 if __name__ == '__main__':
29     executor.start_polling(dp, skip_updates=True)

```

4.7.9 Check user language

Babel is required.

Listing 9: check_user_language.py

```

1  """
2  Babel is required.
3  """
4
5  import logging
6
7  from aiogram import Bot, Dispatcher, executor, md, types
8
9  API_TOKEN = 'BOT TOKEN HERE'
10
11 logging.basicConfig(level=logging.INFO)
12
13
14 bot = Bot(token=API_TOKEN, parse_mode=types.ParseMode.MARKDOWN)
15 dp = Dispatcher(bot)
16
17
18 @dp.message_handler()
19 async def check_language(message: types.Message):
20     locale = message.from_user.locale
21
22     await message.reply(md.text(
23         md.bold('Info about your language:'),
24         md.text(' ', md.bold('Code:'), md.code(locale.language)),
25         md.text(' ', md.bold('Territory:'), md.code(locale.territory or 'Unknown')),
26         md.text(' ', md.bold('Language name:'), md.code(locale.language_name)),
27         md.text(' ', md.bold('English language name:'), md.code(locale.english_name)),
28         sep='\n',
29     ))
30
31
32 if __name__ == '__main__':

```

(continues on next page)

(continued from previous page)

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

4.7.10 Middleware and antiflood

Listing 10: middleware_and_antiflood.py

```
1 import asyncio
2
3 from aiogram import Bot, Dispatcher, executor, types
4 from aiogram.contrib.fsm_storage.redis import RedisStorage2
5 from aiogram.dispatcher import DEFAULT_RATE_LIMIT
6 from aiogram.dispatcher.handler import CancelHandler, current_handler
7 from aiogram.dispatcher.middlewares import BaseMiddleware
8 from aiogram.utils.exceptions import Throttled
9
10 TOKEN = 'BOT_TOKEN_HERE'
11
12 # In this example Redis storage is used
13 storage = RedisStorage2(db=5)
14
15 bot = Bot(token=TOKEN)
16 dp = Dispatcher(bot, storage=storage)
17
18
19 def rate_limit(limit: int, key=None):
20     """
21     Decorator for configuring rate limit and key in different functions.
22
23     :param limit:
24     :param key:
25     :return:
26     """
27
28     def decorator(func):
29         setattr(func, 'throttling_rate_limit', limit)
30         if key:
31             setattr(func, 'throttling_key', key)
32         return func
33
34     return decorator
35
36
37 class ThrottlingMiddleware(BaseMiddleware):
38     """
39     Simple middleware
40     """
41
42     def __init__(self, limit=DEFAULT_RATE_LIMIT, key_prefix='antiflood_'):
43         self.rate_limit = limit
44         self.prefix = key_prefix
```

(continues on next page)

(continued from previous page)

```

45     super(ThrottlingMiddleware, self).__init__()
46
47     async def on_process_message(self, message: types.Message, data: dict):
48         """
49         This handler is called when dispatcher receives a message
50
51         :param message:
52         """
53         # Get current handler
54         handler = current_handler.get()
55
56         # Get dispatcher from context
57         dispatcher = Dispatcher.get_current()
58         # If handler was configured, get rate limit and key from handler
59         if handler:
60             limit = getattr(handler, 'throttling_rate_limit', self.rate_limit)
61             key = getattr(handler, 'throttling_key', f"{self.prefix}_{handler.__name__}")
62         else:
63             limit = self.rate_limit
64             key = f"{self.prefix}_message"
65
66         # Use Dispatcher.throttle method.
67         try:
68             await dispatcher.throttle(key, rate=limit)
69         except Throttled as t:
70             # Execute action
71             await self.message_throttled(message, t)
72
73             # Cancel current handler
74             raise CancelHandler()
75
76     async def message_throttled(self, message: types.Message, throttled: Throttled):
77         """
78         Notify user only on first exceed and notify about unlocking only on last exceed
79
80         :param message:
81         :param throttled:
82         """
83         handler = current_handler.get()
84         dispatcher = Dispatcher.get_current()
85         if handler:
86             key = getattr(handler, 'throttling_key', f"{self.prefix}_{handler.__name__}")
87         else:
88             key = f"{self.prefix}_message"
89
90         # Calculate how many time is left till the block ends
91         delta = throttled.rate - throttled.delta
92
93         # Prevent flooding
94         if throttled.exceeded_count <= 2:
95             await message.reply('Too many requests! ')
96

```

(continues on next page)

(continued from previous page)

```

97     # Sleep.
98     await asyncio.sleep(delta)
99
100    # Check lock status
101    thr = await dispatcher.check_key(key)
102
103    # If current message is not last with current key - do not send message
104    if thr.exceeded_count == throttled.exceeded_count:
105        await message.reply('Unlocked.')
106
107
108    @dp.message_handler(commands=['start'])
109    @rate_limit(5, 'start') # this is not required but you can configure throttling manager
110    ↪ for current handler using it
111    async def cmd_test(message: types.Message):
112        # You can use this command every 5 seconds
113        await message.reply('Test passed! You can use this command every 5 seconds.')
114
115    if __name__ == '__main__':
116        # Setup middleware
117        dp.middleware.setup(ThrottlingMiddleware())
118
119        # Start long-polling
120        executor.start_polling(dp)

```

4.7.11 Webhook example

Listing 11: webhook_example.py

```

1  import logging
2
3  from aiogram import Bot, types
4  from aiogram.contrib.middlewares.logging import LoggingMiddleware
5  from aiogram.dispatcher import Dispatcher
6  from aiogram.dispatcher.webhook import SendMessage
7  from aiogram.utils.executor import start_webhook
8
9
10 API_TOKEN = 'BOT_TOKEN_HERE'
11
12 # webhook settings
13 WEBHOOK_HOST = 'https://your.domain'
14 WEBHOOK_PATH = '/path/to/api'
15 WEBHOOK_URL = f'{WEBHOOK_HOST}{WEBHOOK_PATH}'
16
17 # webserver settings
18 WEBAPP_HOST = 'localhost' # or ip
19 WEBAPP_PORT = 3001
20

```

(continues on next page)

(continued from previous page)

```

21 logging.basicConfig(level=logging.INFO)
22
23 bot = Bot(token=API_TOKEN)
24 dp = Dispatcher(bot)
25 dp.middleware.setup(LoggingMiddleware())
26
27
28 @dp.message_handler()
29 async def echo(message: types.Message):
30     # Regular request
31     # await bot.send_message(message.chat.id, message.text)
32
33     # or reply INTO webhook
34     return SendMessage(message.chat.id, message.text)
35
36
37 async def on_startup(dp):
38     await bot.set_webhook(WEBHOOK_URL)
39     # insert code here to run it after start
40
41
42 async def on_shutdown(dp):
43     logging.warning('Shutting down..')
44
45     # insert code here to run it before shutdown
46
47     # Remove webhook (not acceptable in some cases)
48     await bot.delete_webhook()
49
50     # Close DB connection (if used)
51     await dp.storage.close()
52     await dp.storage.wait_closed()
53
54     logging.warning('Bye!')
55
56
57 if __name__ == '__main__':
58     start_webhook(
59         dispatcher=dp,
60         webhook_path=WEBHOOK_PATH,
61         on_startup=on_startup,
62         on_shutdown=on_shutdown,
63         skip_updates=True,
64         host=WEBAPP_HOST,
65         port=WEBAPP_PORT,
66     )

```

4.7.12 Webhook example old

Listing 12: webhook_example_2.py

```

1  """
2  Example outdated
3  """
4
5  import asyncio
6  import ssl
7  import sys
8
9  from aiohttp import web
10
11 import aiogram
12 from aiogram import Bot, types
13 from aiogram.contrib.fsm_storage.memory import MemoryStorage
14 from aiogram.dispatcher import Dispatcher
15 from aiogram.dispatcher.webhook import get_new_configured_app, SendMessage
16 from aiogram.types import ChatType, ParseMode, ContentType
17 from aiogram.utils.markdown import hbold, bold, text, link
18
19 TOKEN = 'BOT TOKEN HERE'
20
21 WEBHOOK_HOST = 'example.com' # Domain name or IP address which your bot is located.
22 WEBHOOK_PORT = 443 # Telegram Bot API allows only for usage next ports: 443, 80, 88 or
    ↳ 8443
23 WEBHOOK_URL_PATH = '/webhook' # Part of URL
24
25 # This options needed if you use self-signed SSL certificate
26 # Instructions: https://core.telegram.org/bots/self-signed
27 WEBHOOK_SSL_CERT = './webhook_cert.pem' # Path to the ssl certificate
28 WEBHOOK_SSL_PRIV = './webhook_pkey.pem' # Path to the ssl private key
29
30 WEBHOOK_URL = f"https://{WEBHOOK_HOST}:{WEBHOOK_PORT}{WEBHOOK_URL_PATH}"
31
32 # Web app settings:
33 # Use LAN address to listen webhooks
34 # 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
35 WEBAPP_HOST = 'localhost'
36 WEBAPP_PORT = 3001
37
38 BAD_CONTENT = ContentType.PHOTO & ContentType.DOCUMENT & ContentType.STICKER &
    ↳ ContentType.AUDIO
39
40 bot = Bot(TOKEN)
41 storage = MemoryStorage()
42 dp = Dispatcher(bot, storage=storage)
43
44
45 async def cmd_start(message: types.Message):
46     # Yep. aiogram allows to respond into webhook.

```

(continues on next page)

(continued from previous page)

```

47     # https://core.telegram.org/bots/api#making-requests-when-getting-updates
48     return SendMessage(chat_id=message.chat.id, text='Hi from webhook!',
49                        reply_to_message_id=message.message_id)
50
51
52 async def cmd_about(message: types.Message):
53     # In this function markdown utils are used for formatting message text
54     return SendMessage(message.chat.id, text(
55         bold('Hi! I\'m just a simple telegram bot.'),
56         '',
57         text('I\'m powered by', bold('Python', Version(*sys.version_info[:]))),
58         text('With', link(text('aiogram', aiogram.VERSION), 'https://github.com/aiogram/
↪ aiogram')),
59         sep='\n'
60     ), parse_mode=ParseMode.MARKDOWN)
61
62
63 async def cancel(message: types.Message):
64     # Get current state context
65     state = dp.current_state(chat=message.chat.id, user=message.from_user.id)
66
67     # If current user in any state - cancel it.
68     if await state.get_state() is not None:
69         await state.set_state(state=None)
70         return SendMessage(message.chat.id, 'Current action is canceled.')
71     # Otherwise do nothing
72
73
74 async def unknown(message: types.Message):
75     """
76     Handler for unknown messages.
77     """
78     return SendMessage(message.chat.id,
79                        f"I don\'t know what to do with content type `{message.content_
↪ type()}`. Sorry :c")
80
81
82 async def cmd_id(message: types.Message):
83     """
84     Return info about user.
85     """
86     if message.reply_to_message:
87         target = message.reply_to_message.from_user
88         chat = message.chat
89     elif message.forward_from and message.chat.type == ChatType.PRIVATE:
90         target = message.forward_from
91         chat = message.forward_from or message.chat
92     else:
93         target = message.from_user
94         chat = message.chat
95
96     result_msg = [hbold('Info about user:'),

```

(continues on next page)

(continued from previous page)

```

97         f"First name: {target.first_name}"]
98     if target.last_name:
99         result_msg.append(f"Last name: {target.last_name}")
100     if target.username:
101         result_msg.append(f"Username: {target.mention}")
102     result_msg.append(f"User ID: {target.id}")
103
104     result_msg.extend([hbold('Chat:'),
105                       f"Type: {chat.type}",
106                       f"Chat ID: {chat.id}"])
107     if chat.type != ChatType.PRIVATE:
108         result_msg.append(f"Title: {chat.title}")
109     else:
110         result_msg.append(f"Title: {chat.full_name}")
111     return SendMessage(message.chat.id, '\n'.join(result_msg), reply_to_message_
↪ id=message.message_id,
112                        parse_mode=ParseMode.HTML)
113
114
115 async def on_startup(app):
116     # Demonstrate one of the available methods for registering handlers
117     # This command available only in main state (state=None)
118     dp.register_message_handler(cmd_start, commands=['start'])
119
120     # This handler is available in all states at any time.
121     dp.register_message_handler(cmd_about, commands=['help', 'about'], state='*')
122     dp.register_message_handler(unknown, content_types=BAD_CONTENT,
123                                func=lambda message: message.chat.type == ChatType.
↪ PRIVATE)
124
125     # You are able to register one function handler for multiple conditions
126     dp.register_message_handler(cancel, commands=['cancel'], state='*')
127     dp.register_message_handler(cancel, func=lambda message: message.text.lower().
↪ strip() in ['cancel'], state='*')
128
129     dp.register_message_handler(cmd_id, commands=['id'], state='*')
130     dp.register_message_handler(cmd_id, func=lambda message: message.forward_from or
131                                message.reply_to_message and
132                                message.chat.type ==
↪ ChatType.PRIVATE, state='*')
133
134     # Get current webhook status
135     webhook = await bot.get_webhook_info()
136
137     # If URL is bad
138     if webhook.url != WEBHOOK_URL:
139         # If URL doesnt match current - remove webhook
140         if not webhook.url:
141             await bot.delete_webhook()
142
143     # Set new URL for webhook
144     await bot.set_webhook(WEBHOOK_URL, certificate=open(WEBHOOK_SSL_CERT, 'rb'))

```

(continues on next page)

(continued from previous page)

```

145     # If you want to use free certificate signed by LetsEncrypt you need to set only_
↪URL without sending certificate.
146
147
148 async def on_shutdown(app):
149     """
150     Graceful shutdown. This method is recommended by aiohttp docs.
151     """
152     # Remove webhook.
153     await bot.delete_webhook()
154
155     # Close Redis connection.
156     await dp.storage.close()
157     await dp.storage.wait_closed()
158
159
160 if __name__ == '__main__':
161     # Get instance of :class:`aiohttp.web.Application` with configured router.
162     app = get_new_configured_app(dispatcher=dp, path=WEBHOOK_URL_PATH)
163
164     # Setup event handlers.
165     app.on_startup.append(on_startup)
166     app.on_shutdown.append(on_shutdown)
167
168     # Generate SSL context
169     context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)
170     context.load_cert_chain(WEBHOOK_SSL_CERT, WEBHOOK_SSL_PRIV)
171
172     # Start web-application.
173     web.run_app(app, host=WEBAPP_HOST, port=WEBAPP_PORT, ssl_context=context)
174     # Note:
175     # If you start your bot using nginx or Apache web server, SSL context is not_
↪required.
176     # Otherwise you need to set `ssl_context` parameter.

```

4.7.13 Payments

Listing 13: payments.py

```

1 from aiogram import Bot
2 from aiogram import types
3 from aiogram.dispatcher import Dispatcher
4 from aiogram.types.message import ContentType
5 from aiogram.utils import executor
6
7
8 BOT_TOKEN = 'BOT_TOKEN_HERE'
9 PAYMENTS_PROVIDER_TOKEN = '123456789:TEST:1422'
10
11 bot = Bot(BOT_TOKEN)

```

(continues on next page)

(continued from previous page)

```

12 dp = Dispatcher(bot)
13
14 # Setup prices
15 prices = [
16     types.LabeledPrice(label='Working Time Machine', amount=5750),
17     types.LabeledPrice(label='Gift wrapping', amount=500),
18 ]
19
20 # Setup shipping options
21 shipping_options = [
22     types.ShippingOption(id='instant', title='WorldWide Teleporter').add(types.
↳ LabeledPrice('Teleporter', 1000)),
23     types.ShippingOption(id='pickup', title='Local pickup').add(types.LabeledPrice(
↳ 'Pickup', 300)),
24 ]
25
26
27 @dp.message_handler(commands=['start'])
28 async def cmd_start(message: types.Message):
29     await bot.send_message(message.chat.id,
30                             "Hello, I'm the demo merchant bot."
31                             " I can sell you a Time Machine."
32                             " Use /buy to order one, /terms for Terms and Conditions")
33
34
35 @dp.message_handler(commands=['terms'])
36 async def cmd_terms(message: types.Message):
37     await bot.send_message(message.chat.id,
38                             'Thank you for shopping with our demo bot. We hope you like_
↳ your new time machine!\n'
39                             '1. If your time machine was not delivered on time, please_
↳ rethink your concept of time'
40                             ' and try again.\n'
41                             '2. If you find that your time machine is not working, kindly_
↳ contact our future service'
42                             ' workshops on Trappist-1e. They will be accessible anywhere_
↳ between'
43                             ' May 2075 and November 4000 C.E.\n'
44                             '3. If you would like a refund, kindly apply for one_
↳ yesterday and we will have sent it'
45                             ' to you immediately.')
46
47
48 @dp.message_handler(commands=['buy'])
49 async def cmd_buy(message: types.Message):
50     await bot.send_message(message.chat.id,
51                             "Real cards won't work with me, no money will be debited from_
↳ your account."
52                             " Use this test card number to pay for your Time Machine:_
↳ `4242 4242 4242 4242`"
53                             "\n\nThis is your demo invoice:", parse_mode='Markdown')
54     await bot.send_invoice(message.chat.id, title='Working Time Machine',

```

(continues on next page)

(continued from previous page)

```

55         description='Want to visit your great-great-great-
↳ grandparents?'
56             ' Make a fortune at the races?'
57             ' Shake hands with Hammurabi and take a stroll in_
↳ the Hanging Gardens?'
58             ' Order our Working Time Machine today!',
59         provider_token=PAYMENTS_PROVIDER_TOKEN,
60         currency='usd',
61         photo_url='https://telegra.ph/file/d08ff863531f10bf2ea4b.jpg',
62         photo_height=512, # !=0/None or picture won't be shown
63         photo_width=512,
64         photo_size=512,
65         is_flexible=True, # True If you need to set up Shipping Fee
66         prices=prices,
67         start_parameter='time-machine-example',
68         payload='HAPPY FRIDAYS COUPON')
69
70
71 @dp.shipping_query_handler(lambda query: True)
72 async def shipping(shipping_query: types.ShippingQuery):
73     await bot.answer_shipping_query(shipping_query.id, ok=True, shipping_
↳ options=shipping_options,
74                                     error_message='Oh, seems like our Dog couriers are_
↳ having a lunch right now.'
75                                     ' Try again later!')
76
77
78 @dp.pre_checkout_query_handler(lambda query: True)
79 async def checkout(pre_checkout_query: types.PreCheckoutQuery):
80     await bot.answer_pre_checkout_query(pre_checkout_query.id, ok=True,
81                                         error_message="Aliens tried to steal your card's_
↳ CVV,"
82                                         " but we successfully protected_
↳ your credentials,"
83                                         " try to pay again in a few_
↳ minutes, we need a small rest.")
84
85
86 @dp.message_handler(content_types=ContentTypes.SUCCESSFUL_PAYMENT)
87 async def got_payment(message: types.Message):
88     await bot.send_message(message.chat.id,
89                             'Hoooooray! Thanks for payment! We will proceed your order_
↳ for `{}` `{}`'
90                             ' as fast as possible! Stay in touch.'
91                             '\n\nUse /buy again to get a Time Machine for your friend!'.
↳ format(
92                                     message.successful_payment.total_amount / 100, message.
↳ successful_payment.currency),
93                                     parse_mode='Markdown')
94
95
96 if __name__ == '__main__':

```

(continues on next page)

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

4.7.14 Broadcast example

Listing 14: broadcast_example.py

```

1 import asyncio
2 import logging
3
4 from aiogram import Bot, Dispatcher, types
5 from aiogram.utils import exceptions, executor
6
7 API_TOKEN = 'BOT TOKEN HERE'
8
9 logging.basicConfig(level=logging.INFO)
10 log = logging.getLogger('broadcast')
11
12 bot = Bot(token=API_TOKEN, parse_mode=types.ParseMode.HTML)
13 dp = Dispatcher(bot)
14
15
16 def get_users():
17     """
18     Return users list
19
20     In this example returns some random ID's
21     """
22     yield from (61043901, 78238238, 78378343, 98765431, 12345678)
23
24
25 async def send_message(user_id: int, text: str, disable_notification: bool = False) -> bool:
26     """
27     Safe messages sender
28
29     :param user_id:
30     :param text:
31     :param disable_notification:
32     :return:
33     """
34     try:
35         await bot.send_message(user_id, text, disable_notification=disable_notification)
36     except exceptions.BotBlocked:
37         log.error(f"Target [ID:{user_id}]: blocked by user")
38     except exceptions.ChatNotFound:
39         log.error(f"Target [ID:{user_id}]: invalid user ID")
40     except exceptions.RetryAfter as e:
41         log.error(f"Target [ID:{user_id}]: Flood limit is exceeded. Sleep {e.timeout} seconds.")
42         await asyncio.sleep(e.timeout)

```

(continues on next page)

(continued from previous page)

```

43     return await send_message(user_id, text) # Recursive call
44 except exceptions.UserDeactivated:
45     log.error(f"Target [ID:{user_id}]: user is deactivated")
46 except exceptions.TelegramAPIError:
47     log.exception(f"Target [ID:{user_id}]: failed")
48 else:
49     log.info(f"Target [ID:{user_id}]: success")
50     return True
51 return False
52
53
54 async def broadcaster() -> int:
55     """
56     Simple broadcaster
57
58     :return: Count of messages
59     """
60     count = 0
61     try:
62         for user_id in get_users():
63             if await send_message(user_id, '<b>Hello!</b>'):
64                 count += 1
65             await asyncio.sleep(.05) # 20 messages per second (Limit: 30 messages per
66     second)
67     finally:
68         log.info(f"{count} messages successful sent.")
69
70     return count
71
72 if __name__ == '__main__':
73     # Execute broadcaster
74     executor.start(dp, broadcaster())

```

4.7.15 Media group

Listing 15: media_group.py

```

1 import asyncio
2
3 from aiogram import Bot, Dispatcher, executor, filters, types
4
5
6 API_TOKEN = 'BOT_TOKEN_HERE'
7
8 bot = Bot(token=API_TOKEN)
9 dp = Dispatcher(bot)
10
11
12 @dp.message_handler(filters.CommandStart())

```

(continues on next page)

(continued from previous page)

```

13 async def send_welcome(message: types.Message):
14     # So... At first I want to send something like this:
15     await message.reply("Do you want to see many pussies? Are you ready?")
16
17     # Wait a little...
18     await asyncio.sleep(1)
19
20     # Good bots should send chat actions...
21     await types.ChatActions.upload_photo()
22
23     # Create media group
24     media = types.MediaGroup()
25
26     # Attach local file
27     media.attach_photo(types.InputFile('data/cat.jpg'), 'Cat!')
28     # More local files and more cats!
29     media.attach_photo(types.InputFile('data/cats.jpg'), 'More cats!')
30
31     # You can also use URL's
32     # For example: get random puss:
33     media.attach_photo('http://lorempixel.com/400/200/cats/', 'Random cat.')
34
35     # And you can also use file ID:
36     # media.attach_photo('<file_id>', 'cat-cat-cat.')
37
38     # Done! Send media group
39     await message.reply_media_group(media=media)
40
41
42 if __name__ == '__main__':
43     executor.start_polling(dp, skip_updates=True)

```

4.7.16 Local server

Listing 16: local_server.py

```

1 import logging
2
3 from aiogram import Bot, Dispatcher, executor, types
4 from aiogram.bot.api import TelegramAPIServer
5 from aiogram.types import ContentType
6
7 API_TOKEN = 'BOT TOKEN HERE'
8
9 # Configure logging
10 logging.basicConfig(level=logging.INFO)
11
12 # Create private Bot API server endpoints wrapper
13 local_server = TelegramAPIServer.from_base('http://localhost')
14

```

(continues on next page)

(continued from previous page)

```
15 # Initialize bot with using local server
16 bot = Bot(token=API_TOKEN, server=local_server)
17 # ... and dispatcher
18 dp = Dispatcher(bot)
19
20
21 @dp.message_handler(content_types=ContentType.ANY)
22 async def echo(message: types.Message):
23     await message.copy_to(message.chat.id)
24
25
26 if __name__ == '__main__':
27     executor.start_polling(dp, skip_updates=True)
```

4.8 Contribution

TODO

4.9 Links

TODO

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

a

- `aiogram.bot.api`, 62
- `aiogram.utils.auth_widget`, 188
- `aiogram.utils.deep_linking`, 202
- `aiogram.utils.deprecated`, 200
- `aiogram.utils.emoji`, 202
- `aiogram.utils.exceptions`, 192
- `aiogram.utils.executor`, 189
- `aiogram.utils.helper`, 200
- `aiogram.utils.json`, 202
- `aiogram.utils.markdown`, 197
- `aiogram.utils.parts`, 202
- `aiogram.utils.payload`, 201

A

AbstractFilter (class in *aiogram.dispatcher.filters*), 171
add() (*aiogram.types.inline_keyboard.InlineKeyboardMarkup* method), 73
add() (*aiogram.types.reply_keyboard.ReplyKeyboardMarkup* method), 77
add() (*aiogram.types.shipping_option.ShippingOption* method), 125
add_sticker_to_set() (*aiogram.bot.bot.Bot* method), 55
AdminFilter (class in *aiogram.dispatcher.filters*), 169
aiogram.bot.api module, 62
aiogram.utils.auth_widget module, 188
aiogram.utils.deep_linking module, 202
aiogram.utils.deprecated module, 200
aiogram.utils.emoji module, 202
aiogram.utils.exceptions module, 192
aiogram.utils.executor module, 189
aiogram.utils.helper module, 200
aiogram.utils.json module, 202
aiogram.utils.markdown module, 197
aiogram.utils.parts module, 202
aiogram.utils.payload module, 201
AIOGramWarning, 194
AllowedUpdates (class in *aiogram.types.update*), 123
Animation (class in *aiogram.types.animation*), 95
answer() (*aiogram.types.callback_query.CallbackQuery* method), 70
answer() (*aiogram.types.inline_query.InlineQuery* method), 94
answer() (*aiogram.types.message.Message* method), 128
answer_animation() (*aiogram.types.message.Message* method), 131
answer_audio() (*aiogram.types.message.Message* method), 129
answer_callback_query() (*aiogram.bot.bot.Bot* method), 49
answer_chat_action() (*aiogram.types.message.Message* method), 141
answer_contact() (*aiogram.types.message.Message* method), 138
answer_dice() (*aiogram.types.message.Message* method), 140
answer_document() (*aiogram.types.message.Message* method), 132
answer_inline_query() (*aiogram.bot.bot.Bot* method), 57
answer_location() (*aiogram.types.message.Message* method), 136
answer_media_group() (*aiogram.types.message.Message* method), 135
answer_photo() (*aiogram.types.message.Message* method), 128
answer_poll() (*aiogram.types.message.Message* method), 139
answer_pre_checkout_query() (*aiogram.bot.bot.Bot* method), 60
answer_shipping_query() (*aiogram.bot.bot.Bot* method), 59
answer_sticker() (*aiogram.types.message.Message* method), 138
answer_venue() (*aiogram.types.message.Message* method), 137
answer_video() (*aiogram.types.message.Message* method), 133
answer_video_note() (*aiogram.types.message.Message* method), 135

[answer_voice\(\)](#) (*aiogram.types.message.Message* method), 134
[api_url\(\)](#) (*aiogram.bot.api.TelegramAPIServer* method), 62
[as_json\(\)](#) (*aiogram.types.base.TelegramObject* method), 65
[async_task\(\)](#) (*aiogram.Dispatcher* method), 187
[attach\(\)](#) (*aiogram.types.input_media.MediaGroup* method), 98
[attach_audio\(\)](#) (*aiogram.types.input_media.MediaGroup* method), 98
[attach_document\(\)](#) (*aiogram.types.input_media.MediaGroup* method), 99
[attach_many\(\)](#) (*aiogram.types.input_media.MediaGroup* method), 98
[attach_photo\(\)](#) (*aiogram.types.input_media.MediaGroup* method), 99
[attach_video\(\)](#) (*aiogram.types.input_media.MediaGroup* method), 99
[Audio](#) (class in *aiogram.types.audio*), 88
[AuthWidgetData](#) (class in *aiogram.types.auth_widget_data*), 160

B

[BadRequest](#), 194
[BadWebhook](#), 196
[BadWebhookAddrInfo](#), 196
[BadWebhookNoAddressAssociatedWithHostname](#), 196
[BadWebhookPort](#), 196
[ban_chat_member\(\)](#) (*aiogram.bot.bot.Bot* method), 40
[BaseBot](#) (class in *aiogram.bot.base*), 17
[BaseField](#) (class in *aiogram.types.fields*), 65
[bind\(\)](#) (*aiogram.dispatcher.filters.FiltersFactory* method), 161
[bind_filter\(\)](#) (*aiogram.Dispatcher* method), 187
[bold\(\)](#) (in module *aiogram.utils.markdown*), 198
[Bot](#) (class in *aiogram.bot.bot*), 19
[BotBlocked](#), 197
[BotDomainInvalid](#), 196
[BotKicked](#), 197
[BoundFilter](#) (class in *aiogram.dispatcher.filters*), 171
[ButtonDataInvalid](#), 196
[ButtonURLInvalid](#), 195

C

[calc_timeout\(\)](#) (*aiogram.types.chat.ChatActions* class method), 86
[callback_query_handler\(\)](#) (*aiogram.Dispatcher* method), 182
[CallbackGame](#) (class in *aiogram.types.callback_game*), 76
[CallbackQuery](#) (class in *aiogram.types.callback_query*), 70
[CantDemoteChatCreator](#), 196
[CantGetUpdates](#), 197
[CantInitiateConversation](#), 197
[CantParseEntities](#), 196
[CantParseUrl](#), 196
[CantRestrictChatOwner](#), 196
[CantRestrictSelf](#), 196
[CantTalkWithBots](#), 197
[channel_post_handler\(\)](#) (*aiogram.Dispatcher* method), 179
[Chat](#) (class in *aiogram.types.chat*), 79
[chat_member_handler\(\)](#) (*aiogram.Dispatcher* method), 185
[ChatActions](#) (class in *aiogram.types.chat*), 86
[ChatAdminRequired](#), 196
[ChatDescriptionIsNotModified](#), 195
[ChatIdIsEmpty](#), 195
[ChatMember](#) (class in *aiogram.types.chat_member*), 125
[ChatMemberStatus](#) (class in *aiogram.types.chat_member*), 125
[ChatNotFound](#), 195
[ChatPhoto](#) (class in *aiogram.types.chat_photo*), 126
[ChatType](#) (class in *aiogram.types.chat*), 85
[ChatTypeFilter](#) (class in *aiogram.dispatcher.filters*), 170
[check\(\)](#) (*aiogram.dispatcher.filters.AbstractFilter* method), 171
[check\(\)](#) (*aiogram.dispatcher.filters.AdminFilter* method), 169
[check\(\)](#) (*aiogram.dispatcher.filters.builtin.IDFilter* method), 169
[check\(\)](#) (*aiogram.dispatcher.filters.ChatTypeFilter* method), 170
[check\(\)](#) (*aiogram.dispatcher.filters.Command* method), 162
[check\(\)](#) (*aiogram.dispatcher.filters.CommandStart* method), 163
[check\(\)](#) (*aiogram.dispatcher.filters.ContentTypeFilter* method), 167
[check\(\)](#) (*aiogram.dispatcher.filters.ExceptionsFilter* method), 168
[check\(\)](#) (*aiogram.dispatcher.filters.ForwardedMessageFilter* method), 170
[check\(\)](#) (*aiogram.dispatcher.filters.HashTag* method), 166
[check\(\)](#) (*aiogram.dispatcher.filters.IsReplyFilter* method), 169
[check\(\)](#) (*aiogram.dispatcher.filters.IsSenderContact* method), 168
[check\(\)](#) (*aiogram.dispatcher.filters.MediaGroupFilter* method), 170
[check\(\)](#) (*aiogram.dispatcher.filters.Regexp* method), 167
[check\(\)](#) (*aiogram.dispatcher.filters.RegexpCommandsFilter* method), 167

- [check\(\)](#) (*aiogram.dispatcher.filters.StateFilter* method), 168
[check\(\)](#) (*aiogram.dispatcher.filters.Text* method), 166
[check_integrity\(\)](#) (in module *aiogram.utils.auth_widget*), 189
[check_key\(\)](#) (*aiogram.Dispatcher* method), 186
[check_result\(\)](#) (in module *aiogram.bot.api*), 63
[check_signature\(\)](#) (in module *aiogram.utils.auth_widget*), 188
[check_token\(\)](#) (in module *aiogram.bot.api*), 63
[check_token\(\)](#) (in module *aiogram.utils.auth_widget*), 188
[chosen_inline_handler\(\)](#) (*aiogram.Dispatcher* method), 181
[ChosenInlineResult](#) (class in *aiogram.types.chosen_inline_result*), 123
[clean\(\)](#) (*aiogram.types.base.TelegramObject* method), 64
[close\(\)](#) (*aiogram.bot.base.BaseBot* method), 18
[close_bot\(\)](#) (*aiogram.bot.bot.Bot* method), 22
[code\(\)](#) (in module *aiogram.utils.markdown*), 198
[command\(\)](#) (*aiogram.dispatcher.filters.Command.CommandObject* attribute), 163
[Command](#) (class in *aiogram.dispatcher.filters*), 162
[Command.CommandObj](#) (class in *aiogram.dispatcher.filters*), 163
[CommandHelp](#) (class in *aiogram.dispatcher.filters*), 164
[CommandPrivacy](#) (class in *aiogram.dispatcher.filters*), 165
[CommandSettings](#) (class in *aiogram.dispatcher.filters*), 164
[CommandStart](#) (class in *aiogram.dispatcher.filters*), 163
[compose_data\(\)](#) (in module *aiogram.bot.api*), 63
[ConflictError](#), 197
[Contact](#) (class in *aiogram.types.contact*), 126
[ContentType](#) (class in *aiogram.types.message*), 157
[ContentTypeFilter](#) (class in *aiogram.dispatcher.filters*), 167
[ContentTypes](#) (class in *aiogram.types.message*), 158
[copy_message\(\)](#) (*aiogram.bot.bot.Bot* method), 23
[create\(\)](#) (*aiogram.types.force_reply.ForceReply* class method), 88
[create_chat_invite_link\(\)](#) (*aiogram.bot.bot.Bot* method), 43
[create_invite_link\(\)](#) (*aiogram.types.chat.Chat* method), 85
[create_new_sticker_set\(\)](#) (*aiogram.bot.bot.Bot* method), 55
[CurrencyTotalAmountInvalid](#), 196
[current_state\(\)](#) (*aiogram.Dispatcher* method), 186
- D**
- [DateTimeField](#) (class in *aiogram.types.fields*), 67
[decode_payload\(\)](#) (in module *aiogram.utils.deep_linking*), 203
[default](#) (*aiogram.dispatcher.filters.BoundFilter* attribute), 171
[delete\(\)](#) (*aiogram.types.message.Message* method), 156
[delete_chat_photo\(\)](#) (*aiogram.bot.bot.Bot* method), 45
[delete_chat_sticker_set\(\)](#) (*aiogram.bot.bot.Bot* method), 48
[delete_from_set\(\)](#) (*aiogram.types.sticker.Sticker* method), 93
[delete_message\(\)](#) (*aiogram.bot.bot.Bot* method), 53
[delete_message\(\)](#) (*aiogram.types.chat.Chat* method), 85
[delete_my_commands\(\)](#) (*aiogram.bot.bot.Bot* method), 49
[delete_photo\(\)](#) (*aiogram.types.chat.Chat* method), 79
[delete_reply_markup\(\)](#) (*aiogram.types.message.Message* method), 156
[delete_sticker_from_set\(\)](#) (*aiogram.bot.bot.Bot* method), 56
[delete_sticker_set\(\)](#) (*aiogram.types.chat.Chat* method), 84
[delete_webhook\(\)](#) (*aiogram.bot.bot.Bot* method), 21
[deprecated\(\)](#) (in module *aiogram.utils.deprecated*), 200
[DeprecatedReadOnlyClassVar](#) (class in *aiogram.utils.deprecated*), 201
[deserialize\(\)](#) (*aiogram.types.fields.BaseField* method), 66
[deserialize\(\)](#) (*aiogram.types.fields.DateTimeField* method), 68
[deserialize\(\)](#) (*aiogram.types.fields.Field* method), 66
[deserialize\(\)](#) (*aiogram.types.fields.ListField* method), 67
[deserialize\(\)](#) (*aiogram.types.fields.ListOfLists* method), 67
[deserialize\(\)](#) (*aiogram.types.fields.TextField* method), 68
[Dispatcher](#) (class in *aiogram*), 175
[do\(\)](#) (*aiogram.types.chat.Chat* method), 84
[Document](#) (class in *aiogram.types.document*), 88
[download\(\)](#) (*aiogram.types.mixins.Downloadable* method), 68
[download_big\(\)](#) (*aiogram.types.chat_photo.ChatPhoto* method), 126
[download_file\(\)](#) (*aiogram.bot.base.BaseBot* method), 18
[download_file_by_id\(\)](#) (*aiogram.bot.bot.Bot* method), 20
[download_small\(\)](#) (*aiogram.types.chat_photo.ChatPhoto* method), 126

Downloadable (class in aiogram.types.mixins), 68

E

edit_caption() (aiogram.types.message.Message method), 154

edit_chat_invite_link() (aiogram.bot.bot.Bot method), 44

edit_invite_link() (aiogram.types.chat.Chat method), 85

edit_live_location() (aiogram.types.message.Message method), 156

edit_media() (aiogram.types.message.Message method), 155

edit_message_caption() (aiogram.bot.bot.Bot method), 51

edit_message_live_location() (aiogram.bot.bot.Bot method), 33

edit_message_media() (aiogram.bot.bot.Bot method), 52

edit_message_reply_markup() (aiogram.bot.bot.Bot method), 52

edit_message_text() (aiogram.bot.bot.Bot method), 50

edit_reply_markup() (aiogram.types.message.Message method), 155

edit_text() (aiogram.types.message.Message method), 154

edited_channel_post_handler() (aiogram.Dispatcher method), 180

edited_message_handler() (aiogram.Dispatcher method), 178

encode_payload() (in module aiogram.utils.deep_linking), 203

EncryptedCredentials (class in aiogram.types.encrypted_credentials), 69

EncryptedPassportElement (class in aiogram.types.encrypted_passport_element), 75

errors_handler() (aiogram.Dispatcher method), 186

escape_md() (in module aiogram.utils.markdown), 197

ExceptionsFilter (class in aiogram.dispatcher.filters), 168

Executor (class in aiogram.utils.executor), 190

export() (aiogram.types.fields.BaseField method), 66

export_chat_invite_link() (aiogram.bot.bot.Bot method), 43

export_invite_link() (aiogram.types.chat.Chat method), 85

F

Field (class in aiogram.types.fields), 66

File (class in aiogram.types.file), 76

file_url() (aiogram.bot.api.TelegramAPIServer method), 63

FileIsTooBig, 196

Filter (class in aiogram.dispatcher.filters), 171

FiltersFactory (class in aiogram.dispatcher.filters), 161

find_location() (aiogram.types.chat.ChatActions class method), 87

ForceReply (class in aiogram.types.force_reply), 88

forward() (aiogram.types.message.Message method), 154

forward_message() (aiogram.bot.bot.Bot method), 23

ForwardedMessageFilter (class in aiogram.dispatcher.filters), 170

from_url() (aiogram.types.input_file.InputFile class method), 119

FSMStorageWarning, 194

full_name (aiogram.types.user.User property), 74

G

Game (class in aiogram.types.game), 75

GameHighScore (class in aiogram.types.game_high_score), 93

generate_hash() (in module aiogram.utils.auth_widget), 188

generate_payload() (in module aiogram.utils.payload), 201

get_administrators() (aiogram.types.chat.Chat method), 83

get_args() (aiogram.types.message.Message method), 127

get_chat() (aiogram.bot.bot.Bot method), 47

get_chat_administrators() (aiogram.bot.bot.Bot method), 47

get_chat_member() (aiogram.bot.bot.Bot method), 48

get_chat_member_count() (aiogram.bot.bot.Bot method), 47

get_chat_members_count() (aiogram.bot.bot.Bot method), 48

get_command() (aiogram.types.message.Message method), 127

get_file() (aiogram.bot.bot.Bot method), 39

get_file() (aiogram.types.input_file.InputFile method), 119

get_file() (aiogram.types.mixins.Downloadable method), 69

get_filename() (aiogram.types.input_file.InputFile method), 119

get_full_command() (aiogram.types.message.Message method), 127

get_game_high_scores() (aiogram.bot.bot.Bot method), 62

get_me() (aiogram.bot.bot.Bot method), 22

get_member() (aiogram.types.chat.Chat method), 84

<code>get_member_count()</code> (<i>aiogram.types.chat.Chat method</i>), 84	<code>InlineQueryResultCachedAudio</code> (<i>class in aiogram.types.inline_query_result</i>), 118
<code>get_members_count()</code> (<i>aiogram.types.chat.Chat method</i>), 84	<code>InlineQueryResultCachedDocument</code> (<i>class in aiogram.types.inline_query_result</i>), 115
<code>get_my_commands()</code> (<i>aiogram.bot.bot.Bot method</i>), 50	<code>InlineQueryResultCachedGif</code> (<i>class in aiogram.types.inline_query_result</i>), 113
<code>get_start_link()</code> (<i>in module aiogram.utils.deep_linking</i>), 203	<code>InlineQueryResultCachedMpeg4Gif</code> (<i>class in aiogram.types.inline_query_result</i>), 114
<code>get_startgroup_link()</code> (<i>in module aiogram.utils.deep_linking</i>), 203	<code>InlineQueryResultCachedPhoto</code> (<i>class in aiogram.types.inline_query_result</i>), 112
<code>get_sticker_set()</code> (<i>aiogram.bot.bot.Bot method</i>), 54	<code>InlineQueryResultCachedSticker</code> (<i>class in aiogram.types.inline_query_result</i>), 114
<code>get_text()</code> (<i>aiogram.types.message_entity.MessageEntity method</i>), 71	<code>InlineQueryResultCachedVideo</code> (<i>class in aiogram.types.inline_query_result</i>), 116
<code>get_updates()</code> (<i>aiogram.bot.bot.Bot method</i>), 20	<code>InlineQueryResultCachedVoice</code> (<i>class in aiogram.types.inline_query_result</i>), 117
<code>get_url()</code> (<i>aiogram.types.chat.Chat method</i>), 79	<code>InlineQueryResultContact</code> (<i>class in aiogram.types.inline_query_result</i>), 111
<code>get_url()</code> (<i>aiogram.types.mixins.Downloadable method</i>), 69	<code>InlineQueryResultDocument</code> (<i>class in aiogram.types.inline_query_result</i>), 108
<code>get_user_profile_photos()</code> (<i>aiogram.bot.bot.Bot method</i>), 39	<code>InlineQueryResultGame</code> (<i>class in aiogram.types.inline_query_result</i>), 111
<code>get_value()</code> (<i>aiogram.types.fields.BaseField method</i>), 65	<code>InlineQueryResultGif</code> (<i>class in aiogram.types.inline_query_result</i>), 103
<code>get_webhook_info()</code> (<i>aiogram.bot.bot.Bot method</i>), 21	<code>InlineQueryResultLocation</code> (<i>class in aiogram.types.inline_query_result</i>), 109
<code>GroupDeactivated</code> , 196	<code>InlineQueryResultMpeg4Gif</code> (<i>class in aiogram.types.inline_query_result</i>), 104
<code>guess_filename()</code> (<i>in module aiogram.bot.api</i>), 63	<code>InlineQueryResultPhoto</code> (<i>class in aiogram.types.inline_query_result</i>), 102
H	<code>InlineQueryResultVenue</code> (<i>class in aiogram.types.inline_query_result</i>), 110
<code>HashTag</code> (<i>class in aiogram.dispatcher.filters</i>), 166	<code>InlineQueryResultVideo</code> (<i>class in aiogram.types.inline_query_result</i>), 105
<code>hbold()</code> (<i>in module aiogram.utils.markdown</i>), 198	<code>InlineQueryResultVoice</code> (<i>class in aiogram.types.inline_query_result</i>), 107
<code>hcode()</code> (<i>in module aiogram.utils.markdown</i>), 198	<code>InputContactMessageContent</code> (<i>class in aiogram.types.input_message_content</i>), 120
<code>hide_link()</code> (<i>in module aiogram.utils.markdown</i>), 199	<code>InputFile</code> (<i>class in aiogram.types.input_file</i>), 118
<code>hitalic()</code> (<i>in module aiogram.utils.markdown</i>), 198	<code>InputLocationMessageContent</code> (<i>class in aiogram.types.input_message_content</i>), 121
<code>hlink()</code> (<i>in module aiogram.utils.markdown</i>), 199	<code>InputMedia</code> (<i>class in aiogram.types.input_media</i>), 95
<code>hpre()</code> (<i>in module aiogram.utils.markdown</i>), 199	<code>InputMediaAnimation</code> (<i>class in aiogram.types.input_media</i>), 96
<code>hstrikethrough()</code> (<i>in module aiogram.utils.markdown</i>), 199	<code>InputMediaAudio</code> (<i>class in aiogram.types.input_media</i>), 97
<code>html_text</code> (<i>aiogram.types.message.Message property</i>), 127	<code>InputMediaDocument</code> (<i>class in aiogram.types.input_media</i>), 96
<code>hunderline()</code> (<i>in module aiogram.utils.markdown</i>), 199	<code>InputMediaPhoto</code> (<i>class in aiogram.types.input_media</i>), 97
I	<code>InputMediaVideo</code> (<i>class in aiogram.types.input_media</i>), 98
<code>IDFilter</code> (<i>class in aiogram.dispatcher.filters.builtin</i>), 169	<code>InputMessageContent</code> (<i>class in aiogram.types.input_message_content</i>), 120
<code>inline_handler()</code> (<i>aiogram.Dispatcher method</i>), 180	
<code>InlineKeyboardButton</code> (<i>class in aiogram.types.inline_keyboard</i>), 74	
<code>InlineKeyboardExpected</code> , 195	
<code>InlineKeyboardMarkup</code> (<i>class in aiogram.types.inline_keyboard</i>), 73	
<code>InlineQuery</code> (<i>class in aiogram.types.inline_query</i>), 94	
<code>InlineQueryResult</code> (<i>class in aiogram.types.inline_query_result</i>), 100	
<code>InlineQueryResultArticle</code> (<i>class in aiogram.types.inline_query_result</i>), 101	
<code>InlineQueryResultAudio</code> (<i>class in aiogram.types.inline_query_result</i>), 106	

[InputTextMessageContent](#) (class in [aiogram.types.input_message_content](#)), 121
[InputVenueMessageContent](#) (class in [aiogram.types.input_message_content](#)), 122
[insert\(\)](#) ([aiogram.types.inline_keyboard.InlineKeyboardMarkup](#) method), 73
[insert\(\)](#) ([aiogram.types.reply_keyboard.ReplyKeyboardMarkup](#) method), 77
[InvalidHttpUrlContent](#), 195
[InvalidPeerID](#), 195
[InvalidQueryID](#), 195
[InvalidStickersSet](#), 196
[InvalidUserId](#), 195
[Invoice](#) (class in [aiogram.types.invoice](#)), 160
[is_channel\(\)](#) ([aiogram.types.chat.ChatType](#) class method), 86
[is_command\(\)](#) ([aiogram.types.message.Message](#) method), 127
[is_forward\(\)](#) ([aiogram.types.message.Message](#) method), 127
[is_group\(\)](#) ([aiogram.types.chat.ChatType](#) class method), 85
[is_group_or_super_group\(\)](#) ([aiogram.types.chat.ChatType](#) class method), 86
[is_polling\(\)](#) ([aiogram.Dispatcher](#) method), 176
[is_private\(\)](#) ([aiogram.types.chat.ChatType](#) class method), 85
[is_super_group\(\)](#) ([aiogram.types.chat.ChatType](#) class method), 85
[IsReplyFilter](#) (class in [aiogram.dispatcher.filters](#)), 169
[IsSenderContact](#) (class in [aiogram.dispatcher.filters](#)), 168
[italic\(\)](#) (in module [aiogram.utils.markdown](#)), 198
[Item](#) (class in [aiogram.utils.helper](#)), 200
[ItemsList](#) (class in [aiogram.utils.helper](#)), 200
[iter_keys\(\)](#) ([aiogram.types.base.TelegramObject](#) method), 65
[iter_values\(\)](#) ([aiogram.types.base.TelegramObject](#) method), 65

K

[key](#) ([aiogram.dispatcher.filters.BoundFilter](#) attribute), 171
[KeyboardButton](#) (class in [aiogram.types.reply_keyboard](#)), 78
[kick\(\)](#) ([aiogram.types.chat.Chat](#) method), 80
[kick_chat_member\(\)](#) ([aiogram.bot.bot.Bot](#) method), 40

L

[LabeledPrice](#) (class in [aiogram.types.labeled_price](#)), 76
[leave\(\)](#) ([aiogram.types.chat.Chat](#) method), 83
[leave_chat\(\)](#) ([aiogram.bot.bot.Bot](#) method), 47
[link\(\)](#) ([aiogram.types.message.Message](#) method), 127
[link\(\)](#) (in module [aiogram.utils.markdown](#)), 199
[ListField](#) (class in [aiogram.types.fields](#)), 66
[ListItem](#) (class in [aiogram.utils.helper](#)), 200
[ListOfLists](#) (class in [aiogram.types.fields](#)), 67
[locale](#) ([aiogram.types.user.User](#) property), 74
[Location](#) (class in [aiogram.types.location](#)), 95
[log_out\(\)](#) ([aiogram.bot.bot.Bot](#) method), 22

M

[MaskPosition](#) (class in [aiogram.types.mask_position](#)), 159
[md_text](#) ([aiogram.types.message.Message](#) property), 127
[me](#) ([aiogram.bot.bot.Bot](#) property), 19
[MediaGroup](#) (class in [aiogram.types.input_media](#)), 98
[MediaGroupFilter](#) (class in [aiogram.dispatcher.filters](#)), 170
[MemoryStorage](#) (class in [aiogram.contrib.fsm_storage.memory](#)), 172
[mention](#) ([aiogram.dispatcher.filters.Command.CommandObj](#) attribute), 163
[mention](#) ([aiogram.types.chat.Chat](#) property), 79
[mention](#) ([aiogram.types.user.User](#) property), 74
[mentioned](#) ([aiogram.dispatcher.filters.Command.CommandObj](#) property), 163
[Message](#) (class in [aiogram.types.message](#)), 127
[message_handler\(\)](#) ([aiogram.Dispatcher](#) method), 177
[MessageCantBeDeleted](#), 195
[MessageCantBeEdited](#), 195
[MessageCantBeForwarded](#), 195
[MessageEntity](#) (class in [aiogram.types.message_entity](#)), 71
[MessageEntityType](#) (class in [aiogram.types.message_entity](#)), 72
[MessageError](#), 194
[MessageIdentifierNotSpecified](#), 195
[MessageIdInvalid](#), 194
[MessageIsNotAPoll](#), 195
[MessageIsTooLong](#), 195
[MessageNotModified](#), 194
[MessageTextIsEmpty](#), 195
[MessageToDeleteNotFound](#), 194
[MessageToEditNotFound](#), 195
[MessageToForwardNotFound](#), 194
[MessageToPinNotFound](#), 194
[MessageToReplyNotFound](#), 195
[MessageWithPollNotFound](#), 195
[MetaTelegramObject](#) (class in [aiogram.types.base](#)), 64
[MethodIsNotAvailable](#), 196
[MethodNotAvailableInPrivateChats](#), 196
[MethodNotKnown](#), 197
[Methods](#) (class in [aiogram.bot.api](#)), 63
[MigrateToChat](#), 197

- module
- aiogram.bot.api, 62
 - aiogram.utils.auth_widget, 188
 - aiogram.utils.deep_linking, 202
 - aiogram.utils.deprecated, 200
 - aiogram.utils.emoji, 202
 - aiogram.utils.exceptions, 192
 - aiogram.utils.executor, 189
 - aiogram.utils.helper, 200
 - aiogram.utils.json, 202
 - aiogram.utils.markdown, 197
 - aiogram.utils.parts, 202
 - aiogram.utils.payload, 201
- MongoStorage (class in *aiogram.contrib.fsm_storage.mongo*), 173
- my_chat_member_handler() (*aiogram.Dispatcher* method), 185
- ## N
- NeedAdministratorRightsInTheChannel, 196
- NetworkError, 197
- NoStickerInRequest, 196
- NotEnoughRightsToPinMessage, 196
- NotEnoughRightsToRestrict, 196
- NotFound, 197
- ## O
- ObjectExpectedAsReplyMarkup, 195
- on_shutdown() (*aiogram.utils.executor.Executor* method), 191
- on_startup() (*aiogram.utils.executor.Executor* method), 191
- OrderInfo (class in *aiogram.types.order_info*), 93
- ## P
- paginate() (in module *aiogram.utils.parts*), 202
- parse() (*aiogram.types.auth_widget_data.AuthWidgetData* class method), 160
- parse() (*aiogram.types.message_entity.MessageEntity* method), 71
- parse_entities() (*aiogram.types.message.Message* method), 127
- ParseMode (class in *aiogram.types.message*), 159
- PassportData (class in *aiogram.types.passport_data*), 73
- PassportElementError (class in *aiogram.types.passport_element_error*), 89
- PassportElementErrorDataField (class in *aiogram.types.passport_element_error*), 89
- PassportElementErrorFile (class in *aiogram.types.passport_element_error*), 90
- PassportElementErrorFiles (class in *aiogram.types.passport_element_error*), 90
- PassportElementErrorFrontSide (class in *aiogram.types.passport_element_error*), 91
- PassportElementErrorReverseSide (class in *aiogram.types.passport_element_error*), 91
- PassportElementErrorSelfie (class in *aiogram.types.passport_element_error*), 92
- PassportFile (class in *aiogram.types.passport_file*), 124
- PaymentProviderInvalid, 196
- PhotoAsInputFileRequired, 196
- PhotoDimensions, 196
- PhotoSize (class in *aiogram.types.photo_size*), 123
- pin() (*aiogram.types.message.Message* method), 157
- pin_chat_message() (*aiogram.bot.bot.Bot* method), 46
- pin_message() (*aiogram.types.chat.Chat* method), 82
- poll_answer_handler() (*aiogram.Dispatcher* method), 184
- poll_handler() (*aiogram.Dispatcher* method), 184
- PollCanBeRequestedInPrivateChatsOnly, 195
- PollCantBeStopped, 195
- PollCantHaveMoreOptions, 195
- PollError, 195
- PollHasAlreadyBeenClosed, 195
- PollMustHaveMoreOptions, 195
- PollOptionsLengthTooLong, 195
- PollOptionsMustBeNonEmpty, 195
- PollQuestionLengthTooLong, 195
- PollQuestionMustBeNonEmpty, 195
- PollsCantBeSentToPrivateChats, 195
- PollSizeError, 195
- pre() (in module *aiogram.utils.markdown*), 198
- pre_checkout_query_handler() (*aiogram.Dispatcher* method), 183
- PreCheckoutQuery (class in *aiogram.types.pre_checkout_query*), 119
- prefix (*aiogram.dispatcher.filters.Command.CommandObj* attribute), 163
- prepare_arg() (in module *aiogram.utils.payload*), 201
- process_update() (*aiogram.Dispatcher* method), 176
- process_updates() (*aiogram.Dispatcher* method), 175
- promote() (*aiogram.types.chat.Chat* method), 81
- promote_chat_member() (*aiogram.bot.bot.Bot* method), 41
- props (*aiogram.types.base.TelegramObject* property), 64
- props_aliases (*aiogram.types.base.TelegramObject* property), 64
- ## Q
- quote_html() (in module *aiogram.utils.markdown*), 197

R

- `record_audio()` (*aiogram.types.chat.ChatActions* class method), 87
- `record_video()` (*aiogram.types.chat.ChatActions* class method), 87
- `record_video_note()` (*aiogram.types.chat.ChatActions* class method), 87
- `record_voice()` (*aiogram.types.chat.ChatActions* class method), 87
- `RedisStorage2` (class in *aiogram.contrib.fsm_storage.redis*), 172
- `Regex` (class in *aiogram.dispatcher.filters*), 167
- `RegexCommandsFilter` (class in *aiogram.dispatcher.filters*), 167
- `register_callback_query_handler()` (*aiogram.Dispatcher* method), 181
- `register_channel_post_handler()` (*aiogram.Dispatcher* method), 179
- `register_chat_member_handler()` (*aiogram.Dispatcher* method), 185
- `register_chosen_inline_handler()` (*aiogram.Dispatcher* method), 181
- `register_edited_channel_post_handler()` (*aiogram.Dispatcher* method), 180
- `register_edited_message_handler()` (*aiogram.Dispatcher* method), 178
- `register_errors_handler()` (*aiogram.Dispatcher* method), 186
- `register_inline_handler()` (*aiogram.Dispatcher* method), 180
- `register_message_handler()` (*aiogram.Dispatcher* method), 176
- `register_my_chat_member_handler()` (*aiogram.Dispatcher* method), 184
- `register_poll_answer_handler()` (*aiogram.Dispatcher* method), 184
- `register_poll_handler()` (*aiogram.Dispatcher* method), 183
- `register_pre_checkout_query_handler()` (*aiogram.Dispatcher* method), 183
- `register_shipping_query_handler()` (*aiogram.Dispatcher* method), 182
- `release_key()` (*aiogram.Dispatcher* method), 187
- `renamed_argument()` (in *aiogram.utils.deprecated*), 200
- `reply()` (*aiogram.types.message.Message* method), 141
- `reply_animation()` (*aiogram.types.message.Message* method), 144
- `reply_audio()` (*aiogram.types.message.Message* method), 142
- `reply_contact()` (*aiogram.types.message.Message* method), 150
- `reply_dice()` (*aiogram.types.message.Message* method), 153
- `reply_document()` (*aiogram.types.message.Message* method), 145
- `reply_location()` (*aiogram.types.message.Message* method), 149
- `reply_media_group()` (*aiogram.types.message.Message* method), 148
- `reply_photo()` (*aiogram.types.message.Message* method), 142
- `reply_poll()` (*aiogram.types.message.Message* method), 151
- `reply_sticker()` (*aiogram.types.message.Message* method), 152
- `reply_venue()` (*aiogram.types.message.Message* method), 150
- `reply_video()` (*aiogram.types.message.Message* method), 146
- `reply_video_note()` (*aiogram.types.message.Message* method), 148
- `reply_voice()` (*aiogram.types.message.Message* method), 147
- `ReplyKeyboardMarkup` (class in *aiogram.types.reply_keyboard*), 77
- `ReplyKeyboardRemove` (class in *aiogram.types.reply_keyboard*), 78
- `request()` (*aiogram.bot.base.BaseBot* method), 18
- `request_timeout()` (*aiogram.bot.base.BaseBot* method), 18
- `required` (*aiogram.dispatcher.filters.BoundFilter* attribute), 171
- `reset_webhook()` (*aiogram.Dispatcher* method), 176
- `resolve()` (*aiogram.dispatcher.filters.FiltersFactory* method), 161
- `ResponseParameters` (class in *aiogram.types.response_parameters*), 92
- `RestartingTelegram`, 197
- `restrict()` (*aiogram.types.chat.Chat* method), 81
- `restrict_chat_member()` (*aiogram.bot.bot.Bot* method), 41
- `ResultIdDuplicate`, 196
- `RethinkDBStorage` (class in *aiogram.contrib.fsm_storage.rethinkdb*), 173
- `RetryAfter`, 197
- `revoke_chat_invite_link()` (*aiogram.bot.bot.Bot* method), 44
- `revoke_invite_link()` (*aiogram.types.chat.Chat* method), 85
- `row()` (*aiogram.types.inline_keyboard.InlineKeyboardMarkup* method), 73
- `row()` (*aiogram.types.reply_keyboard.ReplyKeyboardMarkup* method), 77

S

- `safe_split_text()` (in module `aiogram.utils.parts`), 202
- `save()` (`aiogram.types.input_file.InputFile` method), 119
- `send_animation()` (`aiogram.bot.bot.Bot` method), 29
- `send_audio()` (`aiogram.bot.bot.Bot` method), 25
- `send_chat_action()` (`aiogram.bot.bot.Bot` method), 39
- `send_contact()` (`aiogram.bot.bot.Bot` method), 36
- `send_copy()` (`aiogram.types.message.Message` method), 157
- `send_dice()` (`aiogram.bot.bot.Bot` method), 38
- `send_document()` (`aiogram.bot.bot.Bot` method), 27
- `send_file()` (`aiogram.bot.base.BaseBot` method), 18
- `send_game()` (`aiogram.bot.bot.Bot` method), 61
- `send_invoice()` (`aiogram.bot.bot.Bot` method), 57
- `send_location()` (`aiogram.bot.bot.Bot` method), 32
- `send_media_group()` (`aiogram.bot.bot.Bot` method), 32
- `send_message()` (`aiogram.bot.bot.Bot` method), 22
- `send_photo()` (`aiogram.bot.bot.Bot` method), 24
- `send_poll()` (`aiogram.bot.bot.Bot` method), 36
- `send_sticker()` (`aiogram.bot.bot.Bot` method), 53
- `send_venue()` (`aiogram.bot.bot.Bot` method), 35
- `send_video()` (`aiogram.bot.bot.Bot` method), 28
- `send_video_note()` (`aiogram.bot.bot.Bot` method), 31
- `send_voice()` (`aiogram.bot.bot.Bot` method), 30
- `serialize()` (`aiogram.types.fields.BaseField` method), 65
- `serialize()` (`aiogram.types.fields.DateTimeField` method), 67
- `serialize()` (`aiogram.types.fields.Field` method), 66
- `serialize()` (`aiogram.types.fields.ListField` method), 66
- `serialize()` (`aiogram.types.fields.ListOfLists` method), 67
- `serialize()` (`aiogram.types.fields.TextField` method), 68
- `set_administrator_custom_title()` (`aiogram.types.chat.Chat` method), 82
- `set_chat_administrator_custom_title()` (`aiogram.bot.bot.Bot` method), 43
- `set_chat_description()` (`aiogram.bot.bot.Bot` method), 46
- `set_chat_permissions()` (`aiogram.bot.bot.Bot` method), 43
- `set_chat_photo()` (`aiogram.bot.bot.Bot` method), 45
- `set_chat_sticker_set()` (`aiogram.bot.bot.Bot` method), 48
- `set_chat_title()` (`aiogram.bot.bot.Bot` method), 45
- `set_description()` (`aiogram.types.chat.Chat` method), 80
- `set_game_score()` (`aiogram.bot.bot.Bot` method), 61
- `set_my_commands()` (`aiogram.bot.bot.Bot` method), 49
- `set_passport_data_errors()` (`aiogram.bot.bot.Bot` method), 60
- `set_permissions()` (`aiogram.types.chat.Chat` method), 82
- `set_photo()` (`aiogram.types.chat.Chat` method), 79
- `set_position_in_set()` (`aiogram.types.sticker.Sticker` method), 93
- `set_sticker_position_in_set()` (`aiogram.bot.bot.Bot` method), 56
- `set_sticker_set()` (`aiogram.types.chat.Chat` method), 84
- `set_sticker_set_thumb()` (`aiogram.bot.bot.Bot` method), 56
- `set_title()` (`aiogram.types.chat.Chat` method), 79
- `set_value()` (`aiogram.types.fields.BaseField` method), 65
- `set_web_app()` (`aiogram.utils.executor.Executor` method), 190
- `set_webhook()` (`aiogram.bot.bot.Bot` method), 20
- `set_webhook()` (`aiogram.utils.executor.Executor` method), 191
- `set_webhook()` (in module `aiogram.utils.executor`), 189
- `setup_middleware()` (`aiogram.Dispatcher` method), 188
- `shifted_id` (`aiogram.types.chat.Chat` property), 79
- `shipping_query_handler()` (`aiogram.Dispatcher` method), 182
- `ShippingAddress` (class in `aiogram.types.shipping_address`), 92
- `ShippingOption` (class in `aiogram.types.shipping_option`), 125
- `ShippingQuery` (class in `aiogram.types.shipping_query`), 72
- `skip_updates()` (`aiogram.Dispatcher` method), 175
- `split_text()` (in module `aiogram.utils.parts`), 202
- `start()` (`aiogram.utils.executor.Executor` method), 192
- `start()` (in module `aiogram.utils.executor`), 190
- `start_polling()` (`aiogram.Dispatcher` method), 176
- `start_polling()` (`aiogram.utils.executor.Executor` method), 191
- `start_polling()` (in module `aiogram.utils.executor`), 189
- `start_webhook()` (`aiogram.utils.executor.Executor` method), 191
- `start_webhook()` (in module `aiogram.utils.executor`), 190
- `StartParamInvalid`, 196
- `StateFilter` (class in `aiogram.dispatcher.filters`), 168
- `Sticker` (class in `aiogram.types.sticker`), 93
- `StickerSet` (class in `aiogram.types.sticker_set`), 69
- `stop_live_location()` (`aiogram.types.message.Message` method), 156
- `stop_message_live_location()` (`aiogram.bot.bot.Bot` method), 34
- `stop_poll()` (`aiogram.bot.bot.Bot` method), 53

`stop_polling()` (*aiogram.Dispatcher* method), 176
`strikethrough()` (in module *aiogram.utils.markdown*), 199
`SuccessfulPayment` (class in *aiogram.types.successful_payment*), 71

T

`TelegramAPIError`, 194
`TelegramAPIServer` (class in *aiogram.bot.api*), 62
`TelegramObject` (class in *aiogram.types.base*), 64
`TerminatedByOtherGetUpdates`, 197
`text` (*aiogram.dispatcher.filters.Command.CommandObj* property), 163
`Text` (class in *aiogram.dispatcher.filters*), 166
`text()` (in module *aiogram.utils.markdown*), 197
`TextField` (class in *aiogram.types.fields*), 68
`throttle()` (*aiogram.Dispatcher* method), 186
`Throttled`, 197
`throttled()` (*aiogram.Dispatcher* method), 187
`TimeoutWarning`, 194
`to_object()` (*aiogram.types.base.TelegramObject* class method), 64
`to_python()` (*aiogram.types.base.TelegramObject* method), 64
`to_python()` (*aiogram.types.input_file.InputFile* method), 119
`to_python()` (*aiogram.types.input_media.MediaGroup* method), 100
`ToMuchMessages`, 195
`TypeOfFileMismatch`, 196
`typing()` (*aiogram.types.chat.ChatActions* class method), 86

U

`Unauthorized`, 197
`UnavailableMembers`, 196
`unban()` (*aiogram.types.chat.Chat* method), 80
`unban_chat_member()` (*aiogram.bot.bot.Bot* method), 40
`unbind()` (*aiogram.dispatcher.filters.FiltersFactory* method), 161
`unbind_filter()` (*aiogram.Dispatcher* method), 188
`underline()` (in module *aiogram.utils.markdown*), 199
`unpin()` (*aiogram.types.message.Message* method), 157
`unpin_all_chat_messages()` (*aiogram.bot.bot.Bot* method), 47
`unpin_all_messages()` (*aiogram.types.chat.Chat* method), 83
`unpin_chat_message()` (*aiogram.bot.bot.Bot* method), 46
`unpin_message()` (*aiogram.types.chat.Chat* method), 83
`UnsupportedUrlProtocol`, 196
`Update` (class in *aiogram.types.update*), 122

`update_chat()` (*aiogram.types.chat.Chat* method), 79
`upload_audio()` (*aiogram.types.chat.ChatActions* class method), 87
`upload_document()` (*aiogram.types.chat.ChatActions* class method), 87
`upload_photo()` (*aiogram.types.chat.ChatActions* class method), 86
`upload_sticker_file()` (*aiogram.bot.bot.Bot* method), 54
`upload_video()` (*aiogram.types.chat.ChatActions* class method), 87
`upload_video_note()` (*aiogram.types.chat.ChatActions* class method), 88
`upload_voice()` (*aiogram.types.chat.ChatActions* class method), 87
`url` (*aiogram.types.message.Message* property), 127
`URLHostIsEmpty`, 196
`User` (class in *aiogram.types.user*), 74
`UserDeactivated`, 197
`UserIsAnAdministratorOfTheChat`, 196
`UserProfilePhotos` (class in *aiogram.types.user_profile_photos*), 160

V

`validate()` (*aiogram.dispatcher.filters.AbstractFilter* class method), 171
`validate()` (*aiogram.dispatcher.filters.BoundFilter* class method), 171
`validate()` (*aiogram.dispatcher.filters.Command* class method), 162
`validate()` (*aiogram.dispatcher.filters.Filter* class method), 171
`ValidationError`, 194
`values` (*aiogram.types.base.TelegramObject* property), 64
`Venue` (class in *aiogram.types.venue*), 123
`Video` (class in *aiogram.types.video*), 75
`VideoNote` (class in *aiogram.types.video_note*), 124
`Voice` (class in *aiogram.types.voice*), 120

W

`wait_closed()` (*aiogram.Dispatcher* method), 176
`WebhookInfo` (class in *aiogram.types.webhook_info*), 124
`WebhookRequireHTTPS`, 196
`WrongFileIdentifier`, 196
`WrongRemoteFileIdSpecified`, 196