

---

# **aiogram Documentation**

***Release 3.0.0***

**aiogram Team**

**Sep 01, 2023**



# CONTENTS

<b>1</b>	<b>Features</b>	<b>3</b>
1.1	Simple usage . . . . .	4
<b>2</b>	<b>Contents</b>	<b>7</b>
2.1	Installation . . . . .	7
2.1.1	From PyPI . . . . .	7
2.1.2	From Arch Linux Repository . . . . .	7
2.1.3	From PyPI . . . . .	7
2.1.4	From GitHub . . . . .	7
2.2	Migration FAQ (2.x -> 3.0) . . . . .	7
2.2.1	Dispatcher . . . . .	8
2.2.2	Filtering events . . . . .	8
2.2.3	Bot API . . . . .	8
2.2.4	Middlewares . . . . .	9
2.2.5	Keyboard Markup . . . . .	9
2.2.6	Callbacks data . . . . .	9
2.2.7	Finite State machine . . . . .	9
2.2.8	Sending Files . . . . .	9
2.2.9	Webhook . . . . .	10
2.3	Bot API . . . . .	10
2.3.1	Bot . . . . .	10
2.3.2	Client session . . . . .	12
2.3.3	Types . . . . .	17
2.3.4	Methods . . . . .	236
2.3.5	Enums . . . . .	376
2.3.6	How to download file? . . . . .	388
2.3.7	How to upload file? . . . . .	390
2.4	Handling events . . . . .	392
2.4.1	Router . . . . .	393
2.4.2	Dispatcher . . . . .	397
2.4.3	Dependency injection . . . . .	400
2.4.4	Filtering events . . . . .	402
2.4.5	Long-polling . . . . .	413
2.4.6	Webhook . . . . .	415
2.4.7	Finite State Machine . . . . .	424
2.4.8	Middlewares . . . . .	433
2.4.9	Errors . . . . .	436
2.4.10	Flags . . . . .	438
2.4.11	Class based handlers . . . . .	440
2.5	Utils . . . . .	445

2.5.1	Keyboard builder . . . . .	445
2.5.2	Translation . . . . .	448
2.5.3	Chat action sender . . . . .	453
2.5.4	WebApp . . . . .	455
2.5.5	Callback answer . . . . .	458
2.5.6	Formatting . . . . .	460
2.6	Changelog . . . . .	467
2.6.1	3.0.0 (2023-09-01) . . . . .	467
2.6.2	3.0.0rc2 (2023-08-18) . . . . .	467
2.6.3	3.0.0rc1 (2023-08-06) . . . . .	468
2.6.4	3.0.0b9 (2023-07-30) . . . . .	469
2.6.5	3.0.0b8 (2023-07-17) . . . . .	470
2.6.6	3.0.0b7 (2023-02-18) . . . . .	472
2.6.7	3.0.0b6 (2022-11-18) . . . . .	474
2.6.8	3.0.0b5 (2022-10-02) . . . . .	475
2.6.9	3.0.0b4 (2022-08-14) . . . . .	476
2.6.10	3.0.0b3 (2022-04-19) . . . . .	477
2.6.11	3.0.0b2 (2022-02-19) . . . . .	478
2.6.12	3.0.0b1 (2021-12-12) . . . . .	478
2.6.13	3.0.0a18 (2021-11-10) . . . . .	479
2.6.14	3.0.0a17 (2021-09-24) . . . . .	480
2.6.15	3.0.0a16 (2021-09-22) . . . . .	480
2.6.16	3.0.0a15 (2021-09-10) . . . . .	481
2.6.17	3.0.0a14 (2021-08-17) . . . . .	481
2.6.18	2.14.3 (2021-07-21) . . . . .	482
2.6.19	2.14.2 (2021-07-26) . . . . .	482
2.6.20	2.14 (2021-07-27) . . . . .	482
2.6.21	2.13 (2021-04-28) . . . . .	482
2.6.22	2.12.1 (2021-03-22) . . . . .	483
2.6.23	2.12 (2021-03-14) . . . . .	483
2.6.24	2.11.2 (2021-11-10) . . . . .	484
2.6.25	2.11.1 (2021-11-10) . . . . .	484
2.6.26	2.11 (2021-11-08) . . . . .	484
2.6.27	2.10.1 (2021-09-14) . . . . .	484
2.6.28	2.10 (2021-09-13) . . . . .	484
2.6.29	2.9.2 (2021-06-13) . . . . .	485
2.6.30	2.9 (2021-06-08) . . . . .	485
2.6.31	2.8 (2021-04-26) . . . . .	486
2.6.32	2.7 (2021-04-07) . . . . .	486
2.6.33	2.6.1 (2021-01-25) . . . . .	486
2.6.34	2.6 (2021-01-23) . . . . .	486
2.6.35	2.5.3 (2021-01-05) . . . . .	486
2.6.36	2.5.2 (2021-01-01) . . . . .	487
2.6.37	2.5.1 (2021-01-01) . . . . .	487
2.6.38	2.5 (2021-01-01) . . . . .	487
2.6.39	2.4 (2021-10-29) . . . . .	487
2.6.40	2.3 (2021-08-16) . . . . .	488
2.6.41	2.2 (2021-06-09) . . . . .	488
2.6.42	2.1 (2021-04-18) . . . . .	488
2.6.43	2.0.1 (2021-12-31) . . . . .	489
2.6.44	2.0 (2021-10-28) . . . . .	489
2.6.45	1.4 (2021-08-03) . . . . .	489
2.6.46	1.3.3 (2021-07-16) . . . . .	489
2.6.47	1.3.2 (2021-05-27) . . . . .	489

2.6.48	1.3.1 (2018-05-27)	490
2.6.49	1.3 (2021-04-22)	490
2.6.50	1.2.3 (2018-04-14)	490
2.6.51	1.2.2 (2018-04-08)	490
2.6.52	1.2.1 (2018-03-25)	490
2.6.53	1.2 (2018-02-23)	490
2.6.54	1.1 (2018-01-27)	491
2.6.55	1.0.4 (2018-01-10)	491
2.6.56	1.0.3 (2018-01-07)	491
2.6.57	1.0.2 (2017-11-29)	491
2.6.58	1.0.1 (2017-11-21)	491
2.6.59	1.0 (2017-11-19)	491
2.6.60	0.4.1 (2017-08-03)	492
2.6.61	0.4 (2017-08-05)	492
2.6.62	0.3.4 (2017-08-04)	492
2.6.63	0.3.3 (2017-07-05)	492
2.6.64	0.3.2 (2017-07-04)	492
2.6.65	0.3.1 (2017-07-04)	492
2.6.66	0.2b1 (2017-06-00)	492
2.6.67	0.1 (2017-06-03)	492
2.7	Contributing	492
2.7.1	Developing	492
2.7.2	Star on GitHub	495
2.7.3	Guides	495
2.7.4	Take answers	495
2.7.5	Funding	495
<b>Python Module Index</b>		<b>497</b>
<b>Index</b>		<b>501</b>



**aiogram** is a modern and fully asynchronous framework for [Telegram Bot API](#) written in Python 3.8 using [asyncio](#) and [aiohttp](#).

Make your bots faster and more powerful!

**Documentation:**

- [English](#)
- [Ukrainian](#)





## FEATURES

- Asynchronous ([asyncio docs](#), **PEP 492**)
- Has type hints (**PEP 484**) and can be used with [mypy](#)
- Supports [PyPy](#)
- Supports [Telegram Bot API 6.8](#) and gets fast updates to the latest versions of the Bot API
- Telegram Bot API integration code was [autogenerated](#) and can be easily re-generated when API gets updated
- Updates router (Blueprints)
- Has Finite State Machine
- Uses powerful [magic filters](#)
- Middlewares (incoming updates and API calls)
- Provides [Replies into Webhook](#)
- Integrated I18n/L10n support with GNU Gettext (or Fluent)

**Warning:** It is strongly advised that you have prior experience working with [asyncio](#) before beginning to use **aiogram**.

If you have any questions, you can visit our community chats on Telegram:

- [@aiogram](#)
- [@aiogramua](#)
- [@aiogram\\_uz](#)
- [@aiogram\\_kz](#)
- [@aiogram\\_ru](#)
- [@aiogram\\_fa](#)
- [@aiogram\\_it](#)
- [@aiogram\\_br](#)

## 1.1 Simple usage

```
import asyncio
import logging
import sys
from os import getenv

from aiogram import Bot, Dispatcher, Router, types
from aiogram.enums import ParseMode
from aiogram.filters import CommandStart
from aiogram.types import Message
from aiogram.utils.markdown import hbold

# Bot token can be obtained via https://t.me/BotFather
TOKEN = getenv("BOT_TOKEN")

# All handlers should be attached to the Router (or Dispatcher)
dp = Dispatcher()

@dp.message(CommandStart())
async def command_start_handler(message: Message) -> None:
    """
    This handler receives messages with `/start` command
    """
    # Most event objects have aliases for API methods that can be called in events'
    ↪ context
    # For example if you want to answer to incoming message you can use `message.answer(
    ↪ ..)` alias
    # and the target chat will be passed to :ref:`aiogram.methods.send_message
    ↪ SendMessage`
    # method automatically or call API method directly via
    # Bot instance: `bot.send_message(chat_id=message.chat.id, ...)`
    await message.answer(f"Hello, {hbold(message.from_user.full_name)}!")

@dp.message()
async def echo_handler(message: types.Message) -> None:
    """
    Handler will forward receive a message back to the sender

    By default, message handler will handle all message types (like a text, photo,
    ↪ sticker etc.)
    """
    try:
        # Send a copy of the received message
        await message.send_copy(chat_id=message.chat.id)
    except TypeError:
        # But not all the types is supported to be copied so need to handle it
        await message.answer("Nice try!")
```

(continues on next page)

(continued from previous page)

```
async def main() -> None:
    # Initialize Bot instance with a default parse mode which will be passed to all API
    ↪calls
    bot = Bot(TOKEN, parse_mode=ParseMode.HTML)
    # And the run events dispatching
    await dp.start_polling(bot)

if __name__ == "__main__":
    logging.basicConfig(level=logging.INFO, stream=sys.stdout)
    asyncio.run(main())
```



## CONTENTS

## 2.1 Installation

### 2.1.1 From PyPI

```
pip install -U aiogram
```

### 2.1.2 From Arch Linux Repository

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

Development build (3.x)

### 2.1.3 From PyPI

```
pip install -U aiogram
```

### 2.1.4 From GitHub

```
pip install https://github.com/aiogram/aiogram/archive/refs/heads/dev-3.x.zip
```

## 2.2 Migration FAQ (2.x -> 3.0)

**Danger:** This guide is still in progress.

This version introduces much many breaking changes and architectural improvements, helping to reduce global variables count in your code, provides useful mechanisms to separate your code to modules or just make sharable modules via packages on the PyPi, makes middlewares and filters more controllable and others.

On this page you can read about points that changed corresponding to last stable 2.x version.

**Note:** This page is most like a detailed changelog than a migration guide, but it will be updated in the future.

Feel free to contribute to this page, if you find something that is not mentioned here.

---

## 2.2.1 Dispatcher

- `Dispatcher` class no longer accepts the *Bot* instance into the initializer, it should be passed to dispatcher only for starting polling or handling event from webhook. Also this way adds possibility to use multiple bot instances at the same time (“multibot”)
- `Dispatcher` now can be extended with another Dispatcher-like thing named `Router` ([Read more »](#)). With routes you can easily separate your code to multiple modules and may be share this modules between projects.
- Removed the `_handler` suffix from all event handler decorators and registering methods. ([Read more »](#))
- `Executor` entirely removed, now you can use `Dispatcher` directly to start polling or webhook.
- `Throttling` method is completely removed, now you can use middlewares to control the execution context and use any throttling mechanism you want.
- Removed global context variables from the API types, `Bot` and `Dispatcher` object, from now if you want to get current bot instance inside handlers or filters you should accept the argument `bot`: `Bot` and use it instead of `Bot.get_current()` Inside middlewares it can be accessed via `data["bot"]`.
- Now to skip pending updates, you should call the `aiogram.methods.delete_webhook.DeleteWebhook` method directly instead of passing `skip_updates=True` to start polling method.

## 2.2.2 Filtering events

- Keyword filters can no more be used, use filters explicitly. ([Read more »](#))
- In due to keyword filters was removed all enabled by default filters (`state` and `content_type` now is not enabled), so you should specify them explicitly if you want to use. For example instead of using `@dp.message_handler(content_types=ContentType.PHOTO)` you should use `@router.message(F.photo)`
- Most of common filters is replaced by “magic filter”. ([Read more »](#))
- Now by default message handler receives any content type, if you want specific one just add the filters (Magic or any other)
- `State` filter now is not enabled by default, that’s mean if you using `state="*"` in v2 then you should not pass any state filter in v3, and vice versa, if the state in v2 is not specified now you should specify the state.
- Added possibility to register per-router global filters, that helps to reduces the number of repetitions in the code and makes easily way to control for what each router will be used.

## 2.2.3 Bot API

- Now all API methods is classes with validation (via `pydantic`) (all API calls is also available as methods in the `Bot` class).
- Added more pre-defined Enums and moved into `aiogram.enums` sub-package. For example chat type enum now is `aiogram.enums.ChatType` instead of `aiogram.types.chat.ChatType`. ([Read more »](#))
- Separated HTTP client session into container that can be reused between different Bot instances in the application.
- API Exceptions is no more classified by specific message in due to Telegram has no documented error codes. But all errors is classified by HTTP status code and for each method only one case can be caused with the same

code, so in most cases you should check that only error type (by status-code) without checking error message. ([Read more »](#))

## 2.2.4 Middlewares

- Middlewares can now control a execution context, e.g. using context managers ([Read more »](#))
- All contextual data now is shared between middlewares, filters and handlers to end-to-end use. For example now you can easily pass some data into context inside middleware and get it in the filters layer as the same way as in the handlers via keyword arguments.
- Added mechanism named **flags**, that helps to customize handler behavior in conjunction with middlewares. ([Read more »](#))

## 2.2.5 Keyboard Markup

- Now `aiogram.types.inline_keyboard_markup.InlineKeyboardMarkup` and `aiogram.types.reply_keyboard_markup.ReplyKeyboardMarkup` has no methods to extend it, instead you have to use markup builders `aiogram.utils.keyboard.ReplyKeyboardBuilder` and `aiogram.utils.keyboard.KeyboardBuilder` respectively ([Read more »](#))

## 2.2.6 Callbacks data

- Callback data factory now is strictly typed via `pydantic` models ([Read more »](#))

## 2.2.7 Finite State machine

- State filter will no more added to all handlers, you will need to specify state if you want
- Added possibility to change FSM strategy, for example if you want to control state for each user in chat topics instead of user in chat you can specify it in the Dispatcher.
- Now `aiogram.fsm.state.State` and `aiogram.fsm.state.StateGroup` don't have helper methods like `.set()`, `.next()`, etc.

Instead of this you should set states by passing them directly to `aiogram.fsm.context.FSMContext` ([Read more »](#))

- State proxy is deprecated, you should update the state data by calling `state.set_data(...)` and `state.get_data()` respectively.

## 2.2.8 Sending Files

- From now you should wrap sending files into `InputFile` object before send instead of passing `IO` object directly to the API method. ([Read more »](#))

## 2.2.9 Webhook

- Simplified aiohttp web app configuration
- By default added possibility to upload files when you use reply into webhook

## 2.3 Bot API

**aiogram** now is fully support of [Telegram Bot API](#)

All methods and types is fully autogenerated from Telegram Bot API docs by parser with code-generator.

### 2.3.1 Bot

Bot instance can be created from `aiogram.Bot` (from `aiogram import Bot`) and you can't use methods without instance of bot with configured token.

This class has aliases for all methods and named in `lower_camel_case`.

For example `sendMessage` named `send_message` and has the same specification with all class-based methods.

**Warning:** A full list of methods can be found in the appropriate section of the documentation

```
class aiogram.client.bot.Bot(token: str, session: BaseSession | None = None, parse_mode: str | None =
    None, disable_web_page_preview: bool | None = None, protect_content: bool |
    None = None)
```

Bases: object

```
__init__(token: str, session: BaseSession | None = None, parse_mode: str | None = None,
    disable_web_page_preview: bool | None = None, protect_content: bool | None = None) → None
```

Bot class

#### Parameters

- **token** – Telegram Bot token [Obtained from @BotFather](#)
- **session** – HTTP Client session (For example `AiohttpSession`). If not specified it will be automatically created.
- **parse\_mode** – Default parse mode. If specified it will be propagated into the API methods at runtime.
- **disable\_web\_page\_preview** – Default `disable_web_page_preview` mode. If specified it will be propagated into the API methods at runtime.
- **protect\_content** – Default `protect_content` mode. If specified it will be propagated into the API methods at runtime.

#### Raises

**TokenValidationError** – When token has invalid format this exception will be raised

property token: str



**property id:** `int`

Get bot ID from token

#### Returns

**context**(*auto\_close: bool = True*) → `AsyncIterator[Bot]`

Generate bot context

#### Parameters

**auto\_close** – close session on exit

#### Returns

**async me()** → `User`

Cached alias for `getMe` method

#### Returns

**async download\_file**(*file\_path: str, destination: BinaryIO | Path | str | None = None, timeout: int = 30, chunk\_size: int = 65536, seek: bool = True*) → `BinaryIO | None`

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** – File path on Telegram server (You can get it from `aiogram.types.File`)
- **destination** – Filename, file path or instance of `io.IOBase`. For e.g. `io.BytesIO`, defaults to `None`
- **timeout** – Total timeout in seconds, defaults to 30
- **chunk\_size** – File chunks size, defaults to 64 kb
- **seek** – Go to start of file when downloading is finished. Used only for destination with `typing.BinaryIO` type, defaults to `True`

**async download**(*file: str | Downloadable, destination: BinaryIO | Path | str | None = None, timeout: int = 30, chunk\_size: int = 65536, seek: bool = True*) → `BinaryIO | None`

Download file by `file_id` or `Downloadable` object to destination.

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

#### Parameters

- **file** – `file_id` or `Downloadable` object
- **destination** – Filename, file path or instance of `io.IOBase`. For e.g. `io.BytesIO`, defaults to `None`
- **timeout** – Total timeout in seconds, defaults to 30
- **chunk\_size** – File chunks size, defaults to 64 kb
- **seek** – Go to start of file when downloading is finished. Used only for destination with `typing.BinaryIO` type, defaults to `True`

## 2.3.2 Client session

Client sessions is used for interacting with API server.

### Use Custom API server

For example, if you want to use self-hosted API server:

```
session = AiohttpSession(
    api=TelegramAPIServer.from_base('http://localhost:8082')
)
bot = Bot(..., session=session)
```

```
class aiogram.client.telegram.TelegramAPIServer(base: str, file: str, is_local: bool = False,
                                                wrap_local_file:
                                                ~aiogram.client.telegram.FilesPathWrapper =
                                                <aiogram.client.telegram.BareFilesPathWrapper
                                                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

**base:** str

Base URL

**file:** str

Files URL

**file\_url**(token: str, path: str) → str

Generate URL for downloading files

#### Parameters

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

#### Returns

URL

**classmethod from\_base**(base: str, \*\*kwargs: Any) → *TelegramAPIServer*

Use this method to auto-generate TelegramAPIServer instance from base URL

#### Parameters

**base** – Base URL

#### Returns

instance of *TelegramAPIServer*

**is\_local:** `bool = False`

Mark this server is in `local mode`.

**wrap\_local\_file:** `FilePathWrapper = <aiogram.client.telegram.BareFilePathWrapper object>`

Callback to wrap files path in local mode

## Base

Abstract session for all client sessions

```
class aiogram.client.session.base.BaseSession(api: ~aiogram.client.telegram.TelegramAPIServer =
    TelegramAPIServer(base='https://api.telegram.org/bot{token}/{method}',
    file='https://api.telegram.org/file/bot{token}/{path}',
    is_local=False,
    wrap_local_file=<aiogram.client.telegram.BareFilePathWrapper
    object>), json_loads: ~typing.Callable[[...],
    ~typing.Any] = <function loads>, json_dumps:
    ~typing.Callable[[...], str] = <function dumps>,
    timeout: float = 60.0)
```

This is base class for all HTTP sessions in aiogram.

If you want to create your own session, you must inherit from this class.

**check\_response**(bot: `Bot`, method: `TelegramMethod[TelegramType]`, status\_code: `int`, content: `str`) → `Response[TelegramType]`

Check response status

**abstract async close**() → `None`

Close client session

**abstract async make\_request**(bot: `Bot`, method: `TelegramMethod[TelegramType]`, timeout: `int` | `None` = `None`) → `TelegramType`

Make request to Telegram Bot API

### Parameters

- **bot** – Bot instance
- **method** – Method instance
- **timeout** – Request timeout

### Returns

### Raises

**TelegramApiError** –

**prepare\_value**(value: `Any`, bot: `Bot`, files: `Dict[str, Any]`, \_dumps\_json: `bool` = `True`) → `Any`

Prepare value before send

**abstract async stream\_content**(url: `str`, headers: `Dict[str, Any]` | `None` = `None`, timeout: `int` = `30`, chunk\_size: `int` = `65536`, raise\_for\_status: `bool` = `True`) → `AsyncGenerator[bytes, None]`

Stream reader

## aiohttp

AiohttpSession represents a wrapper-class around *ClientSession* from [aiohttp](#)

Currently *AiohttpSession* is a default session used in *aiogram.Bot*

```
class aiogram.client.session.aiohttp.AiohttpSession(proxy: Iterable[str | Tuple[str, BasicAuth]] | str |  
                                                    Tuple[str, BasicAuth] | None = None, **kwargs:  
                                                    Any)
```

## Usage example

```
from aiogram import Bot  
from aiogram.client.session.aiohttp import AiohttpSession  
  
session = AiohttpSession()  
bot = Bot('42:token', session=session)
```

## Proxy requests in AiohttpSession

In order to use AiohttpSession with proxy connector you have to install [aiohttp-socks](#)

Binding session to bot:

```
from aiogram import Bot  
from aiogram.client.session.aiohttp import AiohttpSession  
  
session = AiohttpSession(proxy="protocol://host:port/")  
bot = Bot(token="bot token", session=session)
```

---

**Note:** Only following protocols are supported: http(tunneling), socks4(a), socks5 as [aiohttp-socks documentation](#) claims.

---

## Authorization

Proxy authorization credentials can be specified in proxy URL or come as an instance of [aiohttp.BasicAuth](#) containing login and password.

Consider examples:

```
from aiohttp import BasicAuth  
from aiogram.client.session.aiohttp import AiohttpSession  
  
auth = BasicAuth(login="user", password="password")  
session = AiohttpSession(proxy=("protocol://host:port", auth))
```

or simply include your basic auth credential in URL

```
session = AiohttpSession(proxy="protocol://user:password@host:port")
```

---

**Note:** Aiogram prefers *BasicAuth* over username and password in URL, so if proxy URL contains login and password and *BasicAuth* object is passed at the same time aiogram will use login and password from *BasicAuth* instance.

---

## Proxy chains

Since `aiohttp-socks` supports proxy chains, you're able to use them in aiogram

Example of chain proxies:

```
from aiohttp import BasicAuth
from aiogram.client.session.aiohttp import AiohttpSession

auth = BasicAuth(login="user", password="password")
session = AiohttpSession(
    proxy={
        "protocol0://host0:port0",
        "protocol1://user:password@host1:port1",
        ("protocol2://host2:port2", auth),
    } # can be any iterable if not set
)
```

## Client session middlewares

In some cases you may want to add some middlewares to the client session to customize the behavior of the client.

Some useful cases that is:

- Log the outgoing requests
- Customize the request parameters
- Handle rate limiting errors and retry the request
- others ...

So, you can do it using client session middlewares. A client session middleware is a function (or callable class) that receives the request and the next middleware to call. The middleware can modify the request and then call the next middleware to continue the request processing.

## How to register client session middleware?

### Register using register method

```
bot.session.middleware(RequestLogging(ignore_methods=[GetUpdates]))
```

## Register using decorator

```
@bot.session.middleware()
async def my_middleware(
    make_request: NextRequestMiddlewareType[TelegramType],
    bot: "Bot",
    method: TelegramMethod[TelegramType],
) -> Response[TelegramType]:
    # do something with request
    return await make_request(bot, method)
```

## Example

### Class based session middleware

```
1 class RequestLogging(BaseRequestMiddleware):
2     def __init__(self, ignore_methods: Optional[List[Type[TelegramMethod[Any]]]] = None):
3         """
4         Middleware for logging outgoing requests
5
6         :param ignore_methods: methods to ignore in logging middleware
7         """
8         self.ignore_methods = ignore_methods if ignore_methods else []
9
10    async def __call__(
11        self,
12        make_request: NextRequestMiddlewareType[TelegramType],
13        bot: "Bot",
14        method: TelegramMethod[TelegramType],
15    ) -> Response[TelegramType]:
16        if type(method) not in self.ignore_methods:
17            loggers.middlewares.info(
18                "Make request with method=%r by bot id=%d",
19                type(method).__name__,
20                bot.id,
21            )
22        return await make_request(bot, method)
```

---

**Note:** this middleware is already implemented inside aiogram, so, if you want to use it you can just import it from `aiogram.client.session.middlewares.request_logging` import `RequestLogging`

---

## Function based session middleware

```

async def __call__(
    self,
    make_request: NextRequestMiddlewareType[TelegramType],
    bot: "Bot",
    method: TelegramMethod[TelegramType],
) -> Response[TelegramType]:
    try:
        # do something with request
        return await make_request(bot, method)
    finally:
        # do something after request

```

### 2.3.3 Types

Here is list of all available API types:

#### Inline mode

##### ChosenInlineResult

```

class aiogram.types.chosen_inline_result.ChosenInlineResult(*, result_id: str, from_user: User,
                                                            query: str, location: Location | None
                                                            = None, inline_message_id: str | None
                                                            = None, **extra_data: Any)

```

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.

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

**result\_id:** `str`

The unique identifier for the result that was chosen

**from\_user:** `User`

The user that chose the result

**query:** `str`

The query that was used to obtain the result

**location:** `Location | None`

*Optional.* Sender location, only for bots that require user location

**inline\_message\_id:** `str | None`

*Optional.* Identifier of the sent inline message. Available only if there is an [inline keyboard](#) attached to the message. Will be also received in [callback queries](#) and can be used to [edit](#) the message.

## InlineQuery

```
class aiogram.types.inline_query.InlineQuery(*, id: str, from_user: User, query: str, offset: str,
                                             chat_type: str | None = None, location: Location | None
                                             = None, **extra_data: Any)
```

This object represents an incoming inline query. When the user sends an empty query, your bot could return some default or trending results.

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

**id:** `str`

Unique identifier for this query

**from\_user:** `User`

Sender

**query:** `str`

Text of the query (up to 256 characters)

**offset:** `str`

Offset of the results to be returned, can be controlled by the bot

**chat\_type:** `str | None`

*Optional.* Type of the chat from which the inline query was sent. Can be either 'sender' for a private chat with the inline query sender, 'private', 'group', 'supergroup', or 'channel'. The chat type should be always known for requests sent from official clients and most third-party clients, unless the request was sent from a secret chat

**location:** `Location | None`

*Optional.* Sender location, only for bots that request user location

```
answer(results: List[InlineQueryResultCachedAudio | InlineQueryResultCachedDocument |
                    InlineQueryResultCachedGif | InlineQueryResultCachedMpeg4Gif | InlineQueryResultCachedPhoto
                    | InlineQueryResultCachedSticker | InlineQueryResultCachedVideo | InlineQueryResultCachedVoice
                    | InlineQueryResultArticle | InlineQueryResultAudio | InlineQueryResultContact |
                    InlineQueryResultGame | InlineQueryResultDocument | InlineQueryResultGif |
                    InlineQueryResultLocation | InlineQueryResultMpeg4Gif | InlineQueryResultPhoto |
                    InlineQueryResultVenue | InlineQueryResultVideo | InlineQueryResultVoice], cache_time: int |
None = None, is_personal: bool | None = None, next_offset: str | None = None, button:
InlineQueryResultsButton | None = None, switch_pm_parameter: str | None = None, switch_pm_text:
str | None = None, **kwargs: Any) → AnswerInlineQuery
```

Shortcut for method `aiogram.methods.answer_inline_query.AnswerInlineQuery` will automatically fill method attributes:

- `inline_query_id`

Use this method to send answers to an inline query. On success, True is returned.

No more than **50** results per query are allowed.

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

### Parameters

- **results** – A JSON-serialized array of results for the inline query
- **cache\_time** – 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** – 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** – 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.
- **button** – A JSON-serialized object describing a button to be shown above inline query results
- **switch\_pm\_parameter** – [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.
- **switch\_pm\_text** – 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`

#### Returns

instance of method `aiogram.methods.answer_inline_query.AnswerInlineQuery`

### InlineQueryResult

**class** `aiogram.types.inline_query_result.InlineQueryResult`(\*\**extra\_data*: Any)

This object represents one result of an inline query. Telegram clients currently support results of the following 20 types:

- `aiogram.types.inline_query_result_cached_audio.InlineQueryResultCachedAudio`
- `aiogram.types.inline_query_result_cached_document.InlineQueryResultCachedDocument`
- `aiogram.types.inline_query_result_cached_gif.InlineQueryResultCachedGif`
- `aiogram.types.inline_query_result_cached_mpeg4_gif.InlineQueryResultCachedMpeg4Gif`
- `aiogram.types.inline_query_result_cached_photo.InlineQueryResultCachedPhoto`
- `aiogram.types.inline_query_result_cached_sticker.InlineQueryResultCachedSticker`
- `aiogram.types.inline_query_result_cached_video.InlineQueryResultCachedVideo`
- `aiogram.types.inline_query_result_cached_voice.InlineQueryResultCachedVoice`
- `aiogram.types.inline_query_result_article.InlineQueryResultArticle`
- `aiogram.types.inline_query_result_audio.InlineQueryResultAudio`
- `aiogram.types.inline_query_result_contact.InlineQueryResultContact`
- `aiogram.types.inline_query_result_game.InlineQueryResultGame`
- `aiogram.types.inline_query_result_document.InlineQueryResultDocument`
- `aiogram.types.inline_query_result_gif.InlineQueryResultGif`
- `aiogram.types.inline_query_result_location.InlineQueryResultLocation`
- `aiogram.types.inline_query_result_mpeg4_gif.InlineQueryResultMpeg4Gif`
- `aiogram.types.inline_query_result_photo.InlineQueryResultPhoto`
- `aiogram.types.inline_query_result_venue.InlineQueryResultVenue`
- `aiogram.types.inline_query_result_video.InlineQueryResultVideo`

- `aiogram.types.inline_query_result_voice.InlineQueryResultVoice`

**Note:** All URLs passed in inline query results will be available to end users and therefore must be assumed to be **public**.

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

## InlineQueryResultArticle

```
class aiogram.types.inline_query_result_article.InlineQueryResultArticle(*, type: ~typing.Literal[<InlineQueryResultType.ARTICLE>] = 'article') =  
    InlineQueryResult-  
    Type.ARTICLE, id:  
    str, title: str, in-  
    put_message_content:  
    ~aiogram.types.input_text_message_content.InputTextMessageContent |  
    ~aiogram.types.input_location_message_content.InputLocationMessageContent |  
    ~aiogram.types.input_venue_message_content.InputVenueMessageContent |  
    ~aiogram.types.input_contact_message_content.InputContactMessageContent |  
    ~aiogram.types.input_invoice_message_content.InputInvoiceMessageContent,  
    reply_markup:  
    ~aiogram.types.inline_keyboard_markup.InlineKeyboardMarkup | None = None, url:  
    str | None = None,  
    hide_url: bool | None  
    = None, description:  
    str | None = None,  
    thumbnail_url: str |  
    None = None,  
    thumbnail_width: int  
    | None = None,  
    thumbnail_height:  
    int | None = None,  
    **extra_data:  
    ~typing.Any)
```

Represents a link to an article or web page.

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

**type:** `Literal[InlineQueryResultType.ARTICLE]`

Type of the result, must be *article*

**id:** `str`

Unique identifier for this result, 1-64 Bytes

**title:** `str`

Title of the result

**input\_message\_content:** `InputTextMessageContent | InputLocationMessageContent |  
InputVenueMessageContent | InputContactMessageContent | InputInvoiceMessageContent`

Content of the message to be sent

**reply\_markup:** [\*InlineKeyboardMarkup\*](#) | None

*Optional.* [\*Inline keyboard\*](#) attached to the message

**url:** str | None

*Optional.* URL of the result

**hide\_url:** bool | None

*Optional.* Pass True if you don't want the URL to be shown in the message

**description:** str | None

*Optional.* Short description of the result

**thumbnail\_url:** str | None

*Optional.* Url of the thumbnail for the result

**thumbnail\_width:** int | None

*Optional.* Thumbnail width

**thumbnail\_height:** int | None

*Optional.* Thumbnail height

### **InlineQueryResultAudio**

```
class aiogram.types.inline_query_result_audio.InlineQueryResultAudio(*, type: ~typing.Literal[<InlineQueryResultType.AUDIO: 'audio'>] = InlineQueryResultType.AUDIO, id: str, audio_url: str, title: str, caption: str | None = None, parse_mode: str | None = None, input_message_content: ~aiogram.types.input_text_message_content.InputTextMessageContent | ~aiogram.types.input_location_message_content.InputLocationMessageContent | ~aiogram.types.input_venue_message_content.InputVenueMessageContent | ~aiogram.types.input_contact_message_content.InputContactMessageContent | ~aiogram.types.input_invoice_message_content.InputInvoiceMessageContent | None = None, **extra_data: ~typing.Any)
```

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.

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

**type:** `Literal[InlineQueryResultType.AUDIO]`

Type of the result, must be `audio`

**id:** `str`

Unique identifier for this result, 1-64 bytes

**audio\_url:** `str`

A valid URL for the audio file

**title:** `str`

Title

**caption:** `str | None`

*Optional.* Caption, 0-1024 characters after entities parsing

**parse\_mode:** `str | None`

*Optional.* Mode for parsing entities in the audio caption. See [formatting options](#) for more details.

**caption\_entities:** `List[MessageEntity]` | `None`

*Optional.* List of special entities that appear in the caption, which can be specified instead of *parse\_mode*

**performer:** `str` | `None`

*Optional.* Performer

**audio\_duration:** `int` | `None`

*Optional.* Audio duration in seconds

**reply\_markup:** `InlineKeyboardMarkup` | `None`

*Optional.* `Inline keyboard` attached to the message

**input\_message\_content:** `InputTextMessageContent` | `InputLocationMessageContent` | `InputVenueMessageContent` | `InputContactMessageContent` | `InputInvoiceMessageContent` | `None`

*Optional.* Content of the message to be sent instead of the audio

### **InlineQueryResultCachedAudio**

```
class aiogram.types.inline_query_result_cached_audio.InlineQueryResultCachedAudio(*, type:
    ~typing.Literal[<InlineQueryResultCachedAudio>]
    = InlineQueryResultCachedAudio,
    Type.AUDIO,
    id: str,
    audio_file_id:
    str,
    caption:
    str | None
    = None,
    parse_mode:
    str | None
    = sentinel.UNSET_PARSE_MODE,
    caption_entities:
    ~typing.List[~aiogram.types.message_entities.MessageEntity]
    | None =
    None,
    reply_markup:
    ~aiogram.types.inline_keyboard.InlineKeyboardMarkup
    | None =
    None,
    input_message_content:
    ~aiogram.types.input_text_message_content.InputTextMessageContent
    |
    ~aiogram.types.input_location_message_content.InputLocationMessageContent
    |
    ~aiogram.types.input_venue_message_content.InputVenueMessageContent
    |
    ~aiogram.types.input_contact_message_content.InputContactMessageContent
    |
    ~aiogram.types.input_invoice_message_content.InputInvoiceMessageContent
    | None =
    None,
    **kwargs: Any)
    tra_data:
    ~typing.Any)
```

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.

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

**type:** `Literal[InlineQueryResultType.AUDIO]`

Type of the result, must be *audio*

**id:** `str`

Unique identifier for this result, 1-64 bytes

**audio\_file\_id:** `str`

A valid file identifier for the audio file

**caption:** `str` | `None`

*Optional.* Caption, 0-1024 characters after entities parsing

**parse\_mode:** `str` | `None`

*Optional.* Mode for parsing entities in the audio caption. See [formatting options](#) for more details.

**caption\_entities:** `List[MessageEntity]` | `None`

*Optional.* List of special entities that appear in the caption, which can be specified instead of *parse\_mode*

**reply\_markup:** `InlineKeyboardMarkup` | `None`

*Optional.* [Inline keyboard](#) attached to the message

**input\_message\_content:** `InputTextMessageContent` | `InputLocationMessageContent` | `InputVenueMessageContent` | `InputContactMessageContent` | `InputInvoiceMessageContent` | `None`

*Optional.* Content of the message to be sent instead of the audio

### **InlineQueryResultCachedDocument**

```

class aiogram.types.inline_query_result_cached_document.InlineQueryResultCachedDocument(*,
                                                                                          type:
                                                                                          ~typ-
                                                                                          ing.Literal[<InlineQ
                                                                                          'doc-
                                                                                          u-
                                                                                          ment'>]
                                                                                          =
                                                                                          In-
                                                                                          line-
                                                                                          QueryRe-
                                                                                          sult-
                                                                                          Type.DOCUMENT,
                                                                                          id:
                                                                                          str,
                                                                                          ti-
                                                                                          tle:
                                                                                          str,
                                                                                          doc-
                                                                                          u-
                                                                                          ment_file_id:
                                                                                          str,
                                                                                          de-
                                                                                          scrip-
                                                                                          tion:
                                                                                          str
                                                                                          |
                                                                                          None
                                                                                          =
                                                                                          None,
                                                                                          cap-
                                                                                          tion:
                                                                                          str
                                                                                          |
                                                                                          None
                                                                                          =
                                                                                          None,
                                                                                          parse_mode:
                                                                                          str
                                                                                          |
                                                                                          None
                                                                                          =
                                                                                          sen-
                                                                                          tinel.UNSET_PARS
                                                                                          cap-
                                                                                          tion_entities:
                                                                                          ~typ-
                                                                                          ing.List[~aiogram.ty
                                                                                          |
                                                                                          None
                                                                                          =
                                                                                          None,
                                                                                          re-
                                                                                          ply_markup:
                                                                                          ~aiogram.types.inlin
                                                                                          |
                                                                                          None
                                                                                          =
                                                                                          None,
                                                                                          in-

```



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.

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

**type:** `Literal[InlineQueryResultType.DOCUMENT]`

Type of the result, must be *document*

**id:** `str`

Unique identifier for this result, 1-64 bytes

**title:** `str`

Title for the result

**document\_file\_id:** `str`

A valid file identifier for the file

**description:** `str | None`

*Optional.* Short description of the result

**caption:** `str | None`

*Optional.* Caption of the document to be sent, 0-1024 characters after entities parsing

**parse\_mode:** `str | None`

*Optional.* Mode for parsing entities in the document caption. See [formatting options](#) for more details.

**caption\_entities:** `List[MessageEntity] | None`

*Optional.* List of special entities that appear in the caption, which can be specified instead of *parse\_mode*

**reply\_markup:** `InlineKeyboardMarkup | None`

*Optional.* [Inline keyboard](#) attached to the message

**input\_message\_content:** `InputTextMessageContent | InputLocationMessageContent | InputVenueMessageContent | InputContactMessageContent | InputInvoiceMessageContent | None`

*Optional.* Content of the message to be sent instead of the file

### InlineQueryResultCachedGif

```
class aiogram.types.inline_query_result_cached_gif.InlineQueryResultCachedGif(*, type: ~typing.Literal[<InlineQueryResultType.GIF>] = InlineQueryResultType.GIF, id: str, gif_file_id: str, title: str | None = None, caption: str | None = None, parse_mode: str | None = sentinel.UNSET_PARSE_MODE, caption_entities: ~typing.List[~aiogram.types.message_entities.MessageEntity] | None = None, reply_markup: ~aiogram.types.inline_keyboard.InlineKeyboardMarkup | None = None, input_message_content: ~aiogram.types.input_text_message_content.InputTextMessageContent | ~aiogram.types.input_location_message_content.InputLocationMessageContent | ~aiogram.types.input_venue_message_content.InputVenueMessageContent | ~aiogram.types.input_contact_message_content.InputContactMessageContent | ~aiogram.types.input_invoice_message_content.InputInvoiceMessageContent | None = None, **extra_data: ~typing.Any)
```

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.

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

**type:** `Literal[InlineQueryResultType.GIF]`

Type of the result, must be `gif`

**id:** `str`

Unique identifier for this result, 1-64 bytes

**gif\_file\_id:** `str`

A valid file identifier for the GIF file

**title:** `str | None`

*Optional.* Title for the result

**caption:** `str` | `None`

*Optional.* Caption of the GIF file to be sent, 0-1024 characters after entities parsing

**parse\_mode:** `str` | `None`

*Optional.* Mode for parsing entities in the caption. See [formatting options](#) for more details.

**caption\_entities:** `List[MessageEntity]` | `None`

*Optional.* List of special entities that appear in the caption, which can be specified instead of *parse\_mode*

**reply\_markup:** `InlineKeyboardMarkup` | `None`

*Optional.* [Inline keyboard](#) attached to the message

**input\_message\_content:** `InputTextMessageContent` | `InputLocationMessageContent` | `InputVenueMessageContent` | `InputContactMessageContent` | `InputInvoiceMessageContent` | `None`

*Optional.* Content of the message to be sent instead of the GIF animation

### **InlineQueryResultCachedMpeg4Gif**

```
class aiogram.types.inline_query_result_cached_mpeg4_gif.InlineQueryResultCachedMpeg4Gif(*,
                                                                                          type:
                                                                                          ~typ-
                                                                                          ing.Literal[<Inline
                                                                                          'mpeg4_gif'>]
                                                                                          =
                                                                                          In-
                                                                                          line-
                                                                                          QueryRe-
                                                                                          sult-
                                                                                          Type.MPEG4_GIF
                                                                                          id:
                                                                                          str,
                                                                                          mpeg4_file_id:
                                                                                          str,
                                                                                          ti-
                                                                                          tle:
                                                                                          str
                                                                                          |
                                                                                          None
                                                                                          =
                                                                                          None,
                                                                                          cap-
                                                                                          tion:
                                                                                          str
                                                                                          |
                                                                                          None
                                                                                          =
                                                                                          None,
                                                                                          parse_mode:
                                                                                          str
                                                                                          |
                                                                                          None
                                                                                          =
                                                                                          sen-
                                                                                          tinel.UNSET_PAR.
                                                                                          cap-
                                                                                          tion_entities:
                                                                                          ~typ-
                                                                                          ing.List[~aiogram.
                                                                                          |
                                                                                          None
                                                                                          =
                                                                                          None,
                                                                                          re-
                                                                                          ply_markup:
                                                                                          ~aiogram.types.inl
                                                                                          |
                                                                                          None
                                                                                          =
                                                                                          None,
                                                                                          in-
                                                                                          put_message_conta
                                                                                          ~aiogram.types.inp
                                                                                          |
                                                                                          ~aiogram.types.inp
                                                                                          ~aiogram.types.inp
                                                                                          |
                                                                                          ~aiogram.types.inp
```

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.

Source: <https://core.telegram.org/bots/api#inlinequeryresultcachedmpeg4gif>

**type:** `Literal[InlineQueryResultType.MPEG4_GIF]`

Type of the result, must be `mpeg4_gif`

**id:** `str`

Unique identifier for this result, 1-64 bytes

**mpeg4\_file\_id:** `str`

A valid file identifier for the MPEG4 file

**title:** `str | None`

*Optional.* Title for the result

**caption:** `str | None`

*Optional.* Caption of the MPEG-4 file to be sent, 0-1024 characters after entities parsing

**parse\_mode:** `str | None`

*Optional.* Mode for parsing entities in the caption. See [formatting options](#) for more details.

**caption\_entities:** `List[MessageEntity] | None`

*Optional.* List of special entities that appear in the caption, which can be specified instead of `parse_mode`

**reply\_markup:** `InlineKeyboardMarkup | None`

*Optional.* [Inline keyboard](#) attached to the message

**input\_message\_content:** `InputTextMessageContent | InputLocationMessageContent | InputVenueMessageContent | InputContactMessageContent | InputInvoiceMessageContent | None`

*Optional.* Content of the message to be sent instead of the video animation

## InlineQueryResultCachedPhoto

```
class aiogram.types.inline_query_result_cached_photo.InlineQueryResultCachedPhoto(*, type:
    ~typing.Literal[<InlineQueryResultCachedPhoto>]
    = InlineQueryResultCachedPhoto,
    id: str,
    photo_file_id: str, title: str | None
    = None,
    description: str | None = None,
    caption: str | None = None,
    parse_mode: str | None = None,
    disable_notification: bool = False,
    reply_markup: ~aiogram.types.inline_keyboard.InlineKeyboardMarkup | None = None,
    input_message_content: ~aiogram.types.input_text_message_content.InputTextMessageContent
    | ~aiogram.types.input_location_message_content.InputLocationMessageContent
    | ~aiogram.types.input_venue_message_content.InputVenueMessageContent
    | ~aiogram.types.input_contact_message_content.InputContactMessageContent
    | ~aiogram.types.input_invoice_message_content.InputInvoiceMessageContent
    | None = None,
    **kwargs: Any)
    tra_data: Any)
```

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.

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

**type:** `Literal[InlineQueryResultType.PHOTO]`

Type of the result, must be *photo*

**id:** `str`

Unique identifier for this result, 1-64 bytes

**photo\_file\_id:** `str`

A valid file identifier of the photo

**title:** `str | None`

*Optional.* Title for the result

**description:** `str | None`

*Optional.* Short description of the result

**caption:** `str | None`

*Optional.* Caption of the photo to be sent, 0-1024 characters after entities parsing

**parse\_mode:** `str | None`

*Optional.* Mode for parsing entities in the photo caption. See [formatting options](#) for more details.

**caption\_entities:** `List[MessageEntity] | None`

*Optional.* List of special entities that appear in the caption, which can be specified instead of *parse\_mode*

**reply\_markup:** `InlineKeyboardMarkup | None`

*Optional.* [Inline keyboard](#) attached to the message

**input\_message\_content:** `InputTextMessageContent | InputLocationMessageContent | InputVenueMessageContent | InputContactMessageContent | InputInvoiceMessageContent | None`

*Optional.* Content of the message to be sent instead of the photo

## InlineQueryResultCachedSticker

```
class aiogram.types.inline_query_result_cached_sticker.InlineQueryResultCachedSticker(*,
                                                                                         type:
                                                                                         ~typ-
                                                                                         ing.Literal[<InlineQue-
                                                                                         'sticker'>]
                                                                                         =
                                                                                         In-
                                                                                         line-
                                                                                         QueryRe-
                                                                                         sult-
                                                                                         Type.STICKER,
                                                                                         id:
                                                                                         str,
                                                                                         sticker_file_id:
                                                                                         str,
                                                                                         re-
                                                                                         ply_markup:
                                                                                         ~aiogram.types.inline_
                                                                                         |
                                                                                         None
                                                                                         =
                                                                                         None,
                                                                                         in-
                                                                                         put_message_content:
                                                                                         ~aiogram.types.input_
                                                                                         |
                                                                                         ~aiogram.types.input_
                                                                                         |
                                                                                         ~aiogram.types.input_
                                                                                         |
                                                                                         ~aiogram.types.input_
                                                                                         |
                                                                                         None
                                                                                         =
                                                                                         None,
                                                                                         **ex-
                                                                                         tra_data:
                                                                                         ~typ-
                                                                                         ing.Any)
```

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 for static stickers and after 06 July, 2019 for *animated stickers*. Older clients will ignore them.

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

**type:** `Literal[InlineQueryResultType.STICKER]`

Type of the result, must be *sticker*

**id:** `str`



Unique identifier for this result, 1-64 bytes

**sticker\_file\_id:** `str`

A valid file identifier of the sticker

**reply\_markup:** `InlineKeyboardMarkup` | `None`

*Optional.* `Inline keyboard` attached to the message

**input\_message\_content:** `InputTextMessageContent` | `InputLocationMessageContent` | `InputVenueMessageContent` | `InputContactMessageContent` | `InputInvoiceMessageContent` | `None`

*Optional.* Content of the message to be sent instead of the sticker

### **InlineQueryResultCachedVideo**

```
class aiogram.types.inline_query_result_cached_video.InlineQueryResultCachedVideo(*, type:
    ~typing.Literal[<InlineQueryResultCachedVideo>]
    = InlineQueryResultCachedVideo,
    Type.VIDEO,
    id: str,
    video_file_id: str, title: str, description: str | None
    = None,
    caption: str | None
    = None,
    parse_mode: str | None
    = sentinel.UNSET_PARSE_MODE,
    caption_entities: ~typing.List[~aiogram.types.message.MessageEntity]
    | None = None,
    reply_markup: ~aiogram.types.inline_keyboard.InlineKeyboardMarkup
    | None = None,
    input_message_content: ~aiogram.types.input_text_message_content.InputTextMessageContent
    | ~aiogram.types.input_location_message_content.InputLocationMessageContent
    | ~aiogram.types.input_venue_message_content.InputVenueMessageContent
    | ~aiogram.types.input_contact_message_content.InputContactMessageContent
    | ~aiogram.types.input_invoice_message_content.InputInvoiceMessageContent
    | None = None,
    **extra_data: typing.Any)
```

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.

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

**type:** `Literal[InlineQueryResultType.VIDEO]`

Type of the result, must be *video*

**id:** `str`

Unique identifier for this result, 1-64 bytes

**video\_file\_id:** `str`

A valid file identifier for the video file

**title:** `str`

Title for the result

**description:** `str` | `None`

*Optional.* Short description of the result

**caption:** `str` | `None`

*Optional.* Caption of the video to be sent, 0-1024 characters after entities parsing

**parse\_mode:** `str` | `None`

*Optional.* Mode for parsing entities in the video caption. See [formatting options](#) for more details.

**caption\_entities:** `List[MessageEntity]` | `None`

*Optional.* List of special entities that appear in the caption, which can be specified instead of *parse\_mode*

**reply\_markup:** `InlineKeyboardMarkup` | `None`

*Optional.* [Inline keyboard](#) attached to the message

**input\_message\_content:** `InputTextMessageContent` | `InputLocationMessageContent` | `InputVenueMessageContent` | `InputContactMessageContent` | `InputInvoiceMessageContent` | `None`

*Optional.* Content of the message to be sent instead of the video

### **InlineQueryResultCachedVoice**

```
class aiogram.types.inline_query_result_cached_voice.InlineQueryResultCachedVoice(*, type:
    ~typing.Literal[<InlineQueryResultCachedVoice.Voice>]
    = InlineQueryResultCachedVoice.Voice,
    id: str,
    voice_file_id: str, title: str,
    caption: str | None = None,
    parse_mode: str | None = sentinel.UNSET_PARSE_MODE,
    caption_entities: ~typing.List[~aiogram.types.message.MessageEntity] | None = None,
    reply_markup: ~aiogram.types.inline_keyboard.InlineKeyboardMarkup | None = None,
    input_message_content: ~aiogram.types.input_text_message_content.InputTextMessageContent
    | ~aiogram.types.input_location_message_content.InputLocationMessageContent
    | ~aiogram.types.input_venue_message_content.InputVenueMessageContent
    | ~aiogram.types.input_contact_message_content.InputContactMessageContent
    | ~aiogram.types.input_invoice_message_content.InputInvoiceMessageContent
    | None = None,
    **kwargs: Any) tra_data: ~typing.Any)
```

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.

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

**type:** `Literal[InlineQueryResultType.VOICE]`

Type of the result, must be *voice*

**id:** `str`

Unique identifier for this result, 1-64 bytes

**voice\_file\_id:** `str`

A valid file identifier for the voice message

**title:** `str`

Voice message title

**caption:** `str` | `None`

*Optional.* Caption, 0-1024 characters after entities parsing

**parse\_mode:** `str` | `None`

*Optional.* Mode for parsing entities in the voice message caption. See [formatting options](#) for more details.

**caption\_entities:** `List[MessageEntity]` | `None`

*Optional.* List of special entities that appear in the caption, which can be specified instead of *parse\_mode*

**reply\_markup:** `InlineKeyboardMarkup` | `None`

*Optional.* [Inline keyboard](#) attached to the message

**input\_message\_content:** `InputTextMessageContent` | `InputLocationMessageContent` | `InputVenueMessageContent` | `InputContactMessageContent` | `InputInvoiceMessageContent` | `None`

*Optional.* Content of the message to be sent instead of the voice message

### **InlineQueryResultContact**

```
class aiogram.types.inline_query_result_contact.InlineQueryResultContact(*, type: ~typing.Literal[<InlineQueryResultType.CONTACT>] = 'contact') =  
    InlineQueryResultType.CONTACT, id: str, phone_number: str, first_name: str, last_name: str | None = None, vcard: str | None = None, reply_markup: ~aiogram.types.inline_keyboard_markup.InlineKeyboardMarkup | None = None, input_message_content: ~aiogram.types.input_text_message_content.InputTextMessageContent | ~aiogram.types.input_location_message_content.InputLocationMessageContent | ~aiogram.types.input_venue_message_content.InputVenueMessageContent | ~aiogram.types.input_contact_message_content.InputContactMessageContent | ~aiogram.types.input_invoice_message_content.InputInvoiceMessageContent | None = None, thumbnail_url: str | None = None, thumbnail_width: int | None = None, thumbnail_height: int | None = None, **extra_data: ~typing.Any)
```

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.

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

**type:** `Literal[InlineQueryResultType.CONTACT]`

Type of the result, must be *contact*

**id:** `str`

Unique identifier for this result, 1-64 Bytes

**phone\_number:** `str`

Contact's phone number

**first\_name:** `str`

Contact's first name

**last\_name:** `str | None`

*Optional.* Contact's last name

**vcard:** `str | None`

*Optional.* Additional data about the contact in the form of a `vCard`, 0-2048 bytes

**reply\_markup:** [\*InlineKeyboardMarkup\*](#) | None

*Optional.* Inline keyboard attached to the message

**input\_message\_content:** [\*InputTextMessageContent\*](#) | [\*InputLocationMessageContent\*](#) | [\*InputVenueMessageContent\*](#) | [\*InputContactMessageContent\*](#) | [\*InputInvoiceMessageContent\*](#) | None

*Optional.* Content of the message to be sent instead of the contact

**thumbnail\_url:** str | None

*Optional.* Url of the thumbnail for the result

**thumbnail\_width:** int | None

*Optional.* Thumbnail width

**thumbnail\_height:** int | None

*Optional.* Thumbnail height

### **InlineQueryResultDocument**

```
class aiogram.types.inline_query_result_document.InlineQueryResultDocument(*, type: ~typing.Literal[<InlineQueryResultType.DOCUMENT>] = 'document') = InlineQueryResultType.DOCUMENT, id: str, title: str, document_url: str, mime_type: str, caption: str | None = None, parse_mode: str | None = sentinel.UNSET_PARSE_MODE, caption_entities: ~typing.List[~aiogram.types.message_entity | None] = None, description: str | None = None, reply_markup: ~aiogram.types.inline_keyboard_markup | None = None, input_message_content: ~aiogram.types.input_text_message_content | ~aiogram.types.input_location_message_content | ~aiogram.types.input_venue_message_content | ~aiogram.types.input_contact_message_content | ~aiogram.types.input_invoice_message_content | None = None, thumbnail_url: str | None = None, thumbnail_width: int | None = None, thumbnail_height: int | None = None, **extra_data: ~typing.Any)
```

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.

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

**type:** `Literal[InlineQueryResultType.DOCUMENT]`

Type of the result, must be *document*

**id:** `str`

Unique identifier for this result, 1-64 bytes



**title: str**  
 Title for the result

**document\_url: str**  
 A valid URL for the file

**mime\_type: str**  
 MIME type of the content of the file, either 'application/pdf' or 'application/zip'

**caption: str | None**  
*Optional.* Caption of the document to be sent, 0-1024 characters after entities parsing

**parse\_mode: str | None**  
*Optional.* Mode for parsing entities in the document caption. See [formatting options](#) for more details.

**caption\_entities: List[MessageEntity] | None**  
*Optional.* List of special entities that appear in the caption, which can be specified instead of *parse\_mode*

**description: str | None**  
*Optional.* Short description of the result

**reply\_markup: InlineKeyboardMarkup | None**  
*Optional.* Inline keyboard attached to the message

**input\_message\_content: InputTextMessageContent | InputLocationMessageContent | InputVenueMessageContent | InputContactMessageContent | InputInvoiceMessageContent | None**  
*Optional.* Content of the message to be sent instead of the file

**thumbnail\_url: str | None**  
*Optional.* URL of the thumbnail (JPEG only) for the file

**thumbnail\_width: int | None**  
*Optional.* Thumbnail width

**thumbnail\_height: int | None**  
*Optional.* Thumbnail height

### InlineQueryResultGame

```
class aiogram.types.inline_query_result_game.InlineQueryResultGame(*, type: ~typing.Literal[<InlineQueryResultType.GAME: 'game'>] = InlineQueryResultType.GAME, id: str, game_short_name: str, reply_markup: ~aiogram.types.inline_keyboard_markup.InlineKeyboardMarkup | None = None, **extra_data: ~typing.Any)
```

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.

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

**type: Literal[InlineQueryResultType.GAME]**

Type of the result, must be *game*

**id:** `str`  
Unique identifier for this result, 1-64 bytes

**game\_short\_name:** `str`  
Short name of the game

**reply\_markup:** `InlineKeyboardMarkup` | `None`  
*Optional.* `Inline keyboard` attached to the message

## InlineQueryResultGif

```
class aiogram.types.inline_query_result_gif.InlineQueryResultGif(*, type: ~typing.Literal[<InlineQueryResultType.GIF: 'gif'>] =
    InlineQueryResultType.GIF, id: str, gif_url: str, thumbnail_url: str, gif_width: int | None =
    None, gif_height: int | None = None, gif_duration: int | None = None, thumbnail_mime_type:
    str | None = None, title: str | None = None, caption: str | None = None, parse_mode: str
    | None = sentinel.UNSET_PARSE_MODE, caption_entities: ~typing.List[~aiogram.types.message_entity.MessageEntity]
    | None = None, reply_markup: ~aiogram.types.inline_keyboard_markup.InlineKeyboardMarkup
    | None = None, input_message_content: ~aiogram.types.input_text_message_content.InputTextMessageContent
    | ~aiogram.types.input_location_message_content.InputLocationMessageContent
    | ~aiogram.types.input_venue_message_content.InputVenueMessageContent
    | ~aiogram.types.input_contact_message_content.InputContactMessageContent
    | ~aiogram.types.input_invoice_message_content.InputInvoiceMessageContent
    | None = None, **extra_data: ~typing.Any)
```

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.

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

**type:** `Literal[InlineQueryResultType.GIF]`  
Type of the result, must be `gif`

**id:** `str`  
Unique identifier for this result, 1-64 bytes

**gif\_url: str***A valid URL for the GIF file. File size must not exceed 1MB***thumbnail\_url: str***URL of the static (JPEG or GIF) or animated (MPEG4) thumbnail for the result***gif\_width: int | None***Optional. Width of the GIF***gif\_height: int | None***Optional. Height of the GIF***gif\_duration: int | None***Optional. Duration of the GIF in seconds***thumbnail\_mime\_type: str | None***Optional. MIME type of the thumbnail, must be one of 'image/jpeg', 'image/gif', or 'video/mp4'. Defaults to 'image/jpeg'***title: str | None***Optional. Title for the result***caption: str | None***Optional. Caption of the GIF file to be sent, 0-1024 characters after entities parsing***parse\_mode: str | None***Optional. Mode for parsing entities in the caption. See [formatting options](#) for more details.***caption\_entities: List[[MessageEntity](#)] | None***Optional. List of special entities that appear in the caption, which can be specified instead of *parse\_mode****reply\_markup: [InlineKeyboardMarkup](#) | None***Optional. [Inline keyboard](#) attached to the message***input\_message\_content: [InputTextMessageContent](#) | [InputLocationMessageContent](#) | [InputVenueMessageContent](#) | [InputContactMessageContent](#) | [InputInvoiceMessageContent](#) | None***Optional. Content of the message to be sent instead of the GIF animation*

### **[InlineQueryResultLocation](#)**

```
class aiogram.types.inline_query_result_location.InlineQueryResultLocation(*, type: ~typing.Literal[<InlineQueryResultType.LOCATION, 'location'>] = InlineQueryResultType.LOCATION, id: str, latitude: float, longitude: float, title: str, horizontal_accuracy: float | None = None, live_period: int | None = None, heading: int | None = None, proximity_alert_radius: int | None = None, reply_markup: ~aiogram.types.inline_keyboard_markup.InlineKeyboardMarkup | None = None, input_message_content: ~aiogram.types.input_text_message_content.InputTextMessageContent | ~aiogram.types.input_location_message_content.InputLocationMessageContent | ~aiogram.types.input_venue_message_content.InputVenueMessageContent | ~aiogram.types.input_contact_message_content.InputContactMessageContent | ~aiogram.types.input_invoice_message_content.InputInvoiceMessageContent | None = None, thumbnail_url: str | None = None, thumbnail_width: int | None = None, thumbnail_height: int | None = None, **extra_data: ~typing.Any)
```

Represents a location on a map. By default, the location will be sent by the user. Alternatively, you can use *input\_message\_content* to send a message with the specified content instead of the location. **Note:** This will only work in Telegram versions released after 9 April, 2016. Older clients will ignore them.

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

**type:** `Literal[InlineQueryResultType.LOCATION]`

Type of the result, must be *location*

**id:** `str`

Unique identifier for this result, 1-64 Bytes

**latitude:** `float`

Location latitude in degrees

**longitude: float**

Location longitude in degrees

**title: str**

Location title

**horizontal\_accuracy: float | None**

*Optional.* The radius of uncertainty for the location, measured in meters; 0-1500

**live\_period: int | None**

*Optional.* Period in seconds for which the location can be updated, should be between 60 and 86400.

**heading: int | None**

*Optional.* For live locations, a direction in which the user is moving, in degrees. Must be between 1 and 360 if specified.

**proximity\_alert\_radius: int | None**

*Optional.* 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: [InlineKeyboardMarkup](#) | None**

*Optional.* [Inline keyboard](#) attached to the message

**input\_message\_content: [InputTextMessageContent](#) | [InputLocationMessageContent](#) | [InputVenueMessageContent](#) | [InputContactMessageContent](#) | [InputInvoiceMessageContent](#) | None**

*Optional.* Content of the message to be sent instead of the location

**thumbnail\_url: str | None**

*Optional.* Url of the thumbnail for the result

**thumbnail\_width: int | None**

*Optional.* Thumbnail width

**thumbnail\_height: int | None**

*Optional.* Thumbnail height

## InlineQueryResultMpeg4Gif

```
class aiogram.types.inline_query_result_mpeg4_gif.InlineQueryResultMpeg4Gif(*, type: ~typing.Literal[<InlineQueryResultType.MPEG4_GIF>] = 'mpeg4_gif') = InlineQueryResultType.MPEG4_GIF, id: str, mpeg4_url: str, thumbnail_url: str, mpeg4_width: int | None = None, mpeg4_height: int | None = None, mpeg4_duration: int | None = None, thumbnail_mime_type: str | None = None, title: str | None = None, caption: str | None = None, parse_mode: str | None = None, UNSET_PARSE_MODE, caption_entities: ~typing.List[~aiogram.types.message_entity] | None = None, reply_markup: ~aiogram.types.inline_keyboard_markup | None = None, input_message_content: ~aiogram.types.input_text_message_content | ~aiogram.types.input_location_message_content | ~aiogram.types.input_venue_message_content | ~aiogram.types.input_contact_message_content | ~aiogram.types.input_invoice_message_content | None = None, **extra_data: ~typing.Any)
```

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.

Source: <https://core.telegram.org/bots/api#inlinequeryresultmpeg4gif>

**type:** `Literal[InlineQueryResultType.MPEG4_GIF]`

Type of the result, must be *mpeg4\_gif*

**id:** `str`

Unique identifier for this result, 1-64 bytes

**mpeg4\_url:** `str`

A valid URL for the MPEG4 file. File size must not exceed 1MB

**thumbnail\_url:** `str`

URL of the static (JPEG or GIF) or animated (MPEG4) thumbnail for the result

**mpeg4\_width:** `int | None`

*Optional.* Video width

**mpeg4\_height:** `int | None`

*Optional.* Video height

**mpeg4\_duration:** `int | None`

*Optional.* Video duration in seconds

**thumbnail\_mime\_type:** `str | None`

*Optional.* MIME type of the thumbnail, must be one of 'image/jpeg', 'image/gif', or 'video/mp4'. Defaults to 'image/jpeg'

**title:** `str | None`

*Optional.* Title for the result

**caption:** `str | None`

*Optional.* Caption of the MPEG-4 file to be sent, 0-1024 characters after entities parsing

**parse\_mode:** `str | None`

*Optional.* Mode for parsing entities in the caption. See [formatting options](#) for more details.

**caption\_entities:** `List[MessageEntity] | None`

*Optional.* List of special entities that appear in the caption, which can be specified instead of *parse\_mode*

**reply\_markup:** `InlineKeyboardMarkup | None`

*Optional.* [Inline keyboard](#) attached to the message

**input\_message\_content:** `InputTextMessageContent | InputLocationMessageContent | InputVenueMessageContent | InputContactMessageContent | InputInvoiceMessageContent | None`

*Optional.* Content of the message to be sent instead of the video animation

## InlineQueryResultPhoto

```
class aiogram.types.inline_query_result_photo.InlineQueryResultPhoto(*, type: ~typing.Literal[<InlineQueryResultType.PHOTO, 'photo'>] = InlineQueryResultType.PHOTO, id: str, photo_url: str, thumbnail_url: str, photo_width: int | None = None, photo_height: int | None = None, title: str | None = None, description: str | None = None, caption: str | None = None, parse_mode: str | None = None, caption_entities: ~typing.List[~aiogram.types.message_entity.MessageEntity] | None = None, reply_markup: ~aiogram.types.inline_keyboard_markup.InlineKeyboardMarkup | None = None, input_message_content: ~aiogram.types.input_text_message_content.InputTextMessageContent | ~aiogram.types.input_location_message_content.InputLocationMessageContent | ~aiogram.types.input_venue_message_content.InputVenueMessageContent | ~aiogram.types.input_contact_message_content.InputContactMessageContent | ~aiogram.types.input_invoice_message_content.InputInvoiceMessageContent | None = None, **extra_data: ~typing.Any)
```

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.

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

**type:** `Literal[InlineQueryResultType.PHOTO]`

Type of the result, must be *photo*

**id:** `str`

Unique identifier for this result, 1-64 bytes

**photo\_url:** `str`

A valid URL of the photo. Photo must be in **JPEG** format. Photo size must not exceed 5MB

**thumbnail\_url:** `str`

URL of the thumbnail for the photo

**photo\_width:** `int | None`

*Optional.* Width of the photo



**photo\_height:** `int` | `None`

*Optional.* Height of the photo

**title:** `str` | `None`

*Optional.* Title for the result

**description:** `str` | `None`

*Optional.* Short description of the result

**caption:** `str` | `None`

*Optional.* Caption of the photo to be sent, 0-1024 characters after entities parsing

**parse\_mode:** `str` | `None`

*Optional.* Mode for parsing entities in the photo caption. See [formatting options](#) for more details.

**caption\_entities:** `List[MessageEntity]` | `None`

*Optional.* List of special entities that appear in the caption, which can be specified instead of *parse\_mode*

**reply\_markup:** [InlineKeyboardMarkup](#) | `None`

*Optional.* [Inline keyboard](#) attached to the message

**input\_message\_content:** [InputTextMessageContent](#) | [InputLocationMessageContent](#) | [InputVenueMessageContent](#) | [InputContactMessageContent](#) | [InputInvoiceMessageContent](#) | `None`

*Optional.* Content of the message to be sent instead of the photo

### **InlineQueryResultVenue**

```
class aiogram.types.inline_query_result_venue.InlineQueryResultVenue(*, type: ~typing.Literal[<InlineQueryResultType.VENUE, 'venue'>] = InlineQueryResultType.VENUE, id: str, latitude: float, longitude: float, title: str, address: str, foursquare_id: str | None = None, foursquare_type: str | None = None, google_place_id: str | None = None, google_place_type: str | None = None, reply_markup: ~aiogram.types.inline_keyboard_markup.InlineKeyboardMarkup | None = None, input_message_content: ~aiogram.types.input_text_message_content.InputTextMessageContent | ~aiogram.types.input_location_message_content.InputLocationMessageContent | ~aiogram.types.input_venue_message_content.InputVenueMessageContent | ~aiogram.types.input_contact_message_content.InputContactMessageContent | ~aiogram.types.input_invoice_message_content.InputInvoiceMessageContent | None = None, thumbnail_url: str | None = None, thumbnail_width: int | None = None, thumbnail_height: int | None = None, **extra_data: ~typing.Any)
```

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.

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

**type:** `Literal[InlineQueryResultType.VENUE]`

Type of the result, must be *venue*

**id:** `str`

Unique identifier for this result, 1-64 Bytes

**latitude:** `float`

Latitude of the venue location in degrees

**longitude:** `float`

Longitude of the venue location in degrees

**title:** `str`

Title of the venue

**address: str**

Address of the venue

**foursquare\_id: str | None**

*Optional.* Foursquare identifier of the venue if known

**foursquare\_type: str | None**

*Optional.* Foursquare type of the venue, if known. (For example, 'arts\_entertainment/default', 'arts\_entertainment/aquarium' or 'food/icecream'.)

**google\_place\_id: str | None**

*Optional.* Google Places identifier of the venue

**google\_place\_type: str | None**

*Optional.* Google Places type of the venue. (See [supported types](#).)

**reply\_markup: [InlineKeyboardMarkup](#) | None**

*Optional.* [Inline keyboard](#) attached to the message

**input\_message\_content: [InputTextMessageContent](#) | [InputLocationMessageContent](#) | [InputVenueMessageContent](#) | [InputContactMessageContent](#) | [InputInvoiceMessageContent](#) | None**

*Optional.* Content of the message to be sent instead of the venue

**thumbnail\_url: str | None**

*Optional.* Url of the thumbnail for the result

**thumbnail\_width: int | None**

*Optional.* Thumbnail width

**thumbnail\_height: int | None**

*Optional.* Thumbnail height

## InlineQueryResultVideo

```
class aiogram.types.inline_query_result_video.InlineQueryResultVideo(*, type: ~typing.Literal[<InlineQueryResultType.VIDEO: 'video'>] = InlineQueryResultType.VIDEO, id: str, video_url: str, mime_type: str, thumbnail_url: str, title: str, caption: str | None = None, parse_mode: str | None = InlineQueryResultType.UNSET_PARSE_MODE, caption_entities: ~typing.List[~aiogram.types.message_entity.MessageEntity] | None = None, video_width: int | None = None, video_height: int | None = None, video_duration: int | None = None, description: str | None = None, reply_markup: ~aiogram.types.inline_keyboard_markup.InlineKeyboardMarkup | None = None, input_message_content: ~aiogram.types.input_text_message_content.InputTextMessageContent | ~aiogram.types.input_location_message_content.InputLocationMessageContent | ~aiogram.types.input_venue_message_content.InputVenueMessageContent | ~aiogram.types.input_contact_message_content.InputContactMessageContent | ~aiogram.types.input_invoice_message_content.InputInvoiceMessageContent | None = None, **extra_data: ~typing.Any)
```

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`.

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

**type:** `Literal[InlineQueryResultType.VIDEO]`

Type of the result, must be `video`

**id:** `str`

Unique identifier for this result, 1-64 bytes

**video\_url:** `str`

A valid URL for the embedded video player or video file

**mime\_type:** `str`

MIME type of the content of the video URL, ‘text/html’ or ‘video/mp4’

**thumbnail\_url:** `str`

URL of the thumbnail (JPEG only) for the video

**title:** `str`

Title for the result

**caption:** `str | None`

*Optional.* Caption of the video to be sent, 0-1024 characters after entities parsing

**parse\_mode:** `str | None`

*Optional.* Mode for parsing entities in the video caption. See [formatting options](#) for more details.

**caption\_entities:** `List[MessageEntity] | None`

*Optional.* List of special entities that appear in the caption, which can be specified instead of *parse\_mode*

**video\_width:** `int | None`

*Optional.* Video width

**video\_height:** `int | None`

*Optional.* Video height

**video\_duration:** `int | None`

*Optional.* Video duration in seconds

**description:** `str | None`

*Optional.* Short description of the result

**reply\_markup:** `InlineKeyboardMarkup | None`

*Optional.* [Inline keyboard](#) attached to the message

**input\_message\_content:** `InputTextMessageContent | InputLocationMessageContent | InputVenueMessageContent | InputContactMessageContent | InputInvoiceMessageContent | None`

*Optional.* Content of the message to be sent instead of the video. This field is **required** if `InlineQueryResultVideo` is used to send an HTML-page as a result (e.g., a YouTube video).

## InlineQueryResultVoice

```
class aiogram.types.inline_query_result_voice.InlineQueryResultVoice(*, type: ~typing.Literal[<InlineQueryResultType.VOICE: 'voice'>] = InlineQueryResultType.VOICE, id: str, voice_url: str, title: str, caption: str | None = None, parse_mode: str | None = sentinel.UNSET_PARSE_MODE, caption_entities: ~typing.List[~aiogram.types.message_entity.MessageEntity] | None = None, voice_duration: int | None = None, reply_markup: ~aiogram.types.inline_keyboard_markup.InlineKeyboardMarkup | None = None, input_message_content: ~aiogram.types.input_text_message_content.InputTextMessageContent | ~aiogram.types.input_location_message_content.InputLocationMessageContent | ~aiogram.types.input_venue_message_content.InputVenueMessageContent | ~aiogram.types.input_contact_message_content.InputContactMessageContent | ~aiogram.types.input_invoice_message_content.InputInvoiceMessageContent | None = None, **extra_data: ~typing.Any)
```

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.

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

**type:** `Literal[InlineQueryResultType.VOICE]`

Type of the result, must be *voice*

**id:** `str`

Unique identifier for this result, 1-64 bytes

**voice\_url:** `str`

A valid URL for the voice recording

**title:** `str`

Recording title

**caption:** `str | None`

*Optional.* Caption, 0-1024 characters after entities parsing

**parse\_mode:** `str | None`

*Optional.* Mode for parsing entities in the voice message caption. See [formatting options](#) for more details.

**caption\_entities:** `List[MessageEntity]` | `None`

*Optional.* List of special entities that appear in the caption, which can be specified instead of `parse_mode`

**voice\_duration:** `int` | `None`

*Optional.* Recording duration in seconds

**reply\_markup:** `InlineKeyboardMarkup` | `None`

*Optional.* `Inline keyboard` attached to the message

**input\_message\_content:** `InputTextMessageContent` | `InputLocationMessageContent` | `InputVenueMessageContent` | `InputContactMessageContent` | `InputInvoiceMessageContent` | `None`

*Optional.* Content of the message to be sent instead of the voice recording

### InlineQueryResultsButton

```
class aiogram.types.inline_query_results_button.InlineQueryResultsButton(*, text: str, web_app:
    WebAppInfo | None
    = None,
    start_parameter: str |
    None = None,
    **extra_data: Any)
```

This object represents a button to be shown above inline query results. You **must** use exactly one of the optional fields.

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

**text:** `str`

Label text on the button

**web\_app:** `WebAppInfo` | `None`

*Optional.* Description of the `Web App` that will be launched when the user presses the button. The Web App will be able to switch back to the inline mode using the method `switchInlineQuery` inside the Web App.

**start\_parameter:** `str` | `None`

*Optional.* `Deep-linking` parameter for the `/start` message sent to the bot when a user presses the button. 1-64 characters, only A-Z, a-z, 0-9, `_` and `-` are allowed.

### InputContactMessageContent

```
class aiogram.types.input_contact_message_content.InputContactMessageContent(*,
    phone_number:
    str, first_name:
    str, last_name:
    str | None =
    None, vcard: str
    | None = None,
    **extra_data:
    Any)
```

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

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

**phone\_number:** `str`

Contact's phone number

**first\_name:** `str`

Contact's first name

**last\_name:** `str | None`

*Optional.* Contact's last name

**vcard:** `str | None`

*Optional.* Additional data about the contact in the form of a [vCard](#), 0-2048 bytes

## InputInvoiceMessageContent



```

class aiogram.types.input_invoice_message_content.InputInvoiceMessageContent(*, title: str,
                                                                              description: str,
                                                                              payload: str,
                                                                              provider_token:
                                                                              str, currency:
                                                                              str, prices:
                                                                              List[LabeledPrice],
                                                                              max_tip_amount:
                                                                              int | None =
                                                                              None, sug-
                                                                              gested_tip_amounts:
                                                                              List[int] | None
                                                                              = None,
                                                                              provider_data:
                                                                              str | None =
                                                                              None,
                                                                              photo_url: str |
                                                                              None = None,
                                                                              photo_size: int |
                                                                              None = None,
                                                                              photo_width:
                                                                              int | None =
                                                                              None,
                                                                              photo_height:
                                                                              int | None =
                                                                              None,
                                                                              need_name:
                                                                              bool | None =
                                                                              None,
                                                                              need_phone_number:
                                                                              bool | None =
                                                                              None,
                                                                              need_email:
                                                                              bool | None =
                                                                              None,
                                                                              need_shipping_address:
                                                                              bool | None =
                                                                              None,
                                                                              send_phone_number_to_provider:
                                                                              bool | None =
                                                                              None,
                                                                              send_email_to_provider:
                                                                              bool | None =
                                                                              None,
                                                                              is_flexible: bool
                                                                              | None = None,
                                                                              **extra_data:
                                                                              Any)

```

Represents the `content` of an invoice message to be sent as the result of an inline query.

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

**title:** `str`

Product name, 1-32 characters

**description: str**

Product description, 1-255 characters

**payload: str**

Bot-defined invoice payload, 1-128 bytes. This will not be displayed to the user, use for your internal processes.

**provider\_token: str**

Payment provider token, obtained via [@BotFather](#)

**currency: str**

Three-letter ISO 4217 currency code, see [more on currencies](#)

**prices: List[LabeledPrice]**

Price breakdown, a JSON-serialized list of components (e.g. product price, tax, discount, delivery cost, delivery tax, bonus, etc.)

**max\_tip\_amount: int | None**

*Optional.* 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: List[int] | None**

*Optional.* A JSON-serialized array of suggested amounts of tip in the *smallest units* of the currency (integer, **not** float/double). At most 4 suggested tip amounts can be specified. The suggested tip amounts must be positive, passed in a strictly increased order and must not exceed `max_tip_amount`.

**provider\_data: str | None**

*Optional.* A JSON-serialized object for data about the invoice, which will be shared with the payment provider. A detailed description of the required fields should be provided by the payment provider.

**photo\_url: str | None**

*Optional.* URL of the product photo for the invoice. Can be a photo of the goods or a marketing image for a service.

**photo\_size: int | None**

*Optional.* Photo size in bytes

**photo\_width: int | None**

*Optional.* Photo width

**photo\_height: int | None**

*Optional.* Photo height

**need\_name: bool | None**

*Optional.* Pass True if you require the user's full name to complete the order

**need\_phone\_number: bool | None**

*Optional.* Pass True if you require the user's phone number to complete the order

**need\_email: bool | None**

*Optional.* Pass True if you require the user's email address to complete the order

**need\_shipping\_address: bool | None**

*Optional.* Pass True if you require the user's shipping address to complete the order

**send\_phone\_number\_to\_provider:** bool | None

*Optional.* Pass True if the user's phone number should be sent to provider

**send\_email\_to\_provider:** bool | None

*Optional.* Pass True if the user's email address should be sent to provider

**is\_flexible:** bool | None

*Optional.* Pass True if the final price depends on the shipping method

## InputLocationMessageContent

```
class aiogram.types.input_location_message_content.InputLocationMessageContent(*, latitude:
    float,
    longitude:
    float,
    horizontal_accuracy:
    float | None
    = None,
    live_period:
    int | None =
    None,
    heading: int |
    None =
    None,
    proximity_alert_radius:
    int | None =
    None, **extra_data:
    Any)
```

Represents the [content](#) of a location message to be sent as the result of an inline query.

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

**latitude:** float

Latitude of the location in degrees

**longitude:** float

Longitude of the location in degrees

**horizontal\_accuracy:** float | None

*Optional.* The radius of uncertainty for the location, measured in meters; 0-1500

**live\_period:** int | None

*Optional.* Period in seconds for which the location can be updated, should be between 60 and 86400.

**heading:** int | None

*Optional.* For live locations, a direction in which the user is moving, in degrees. Must be between 1 and 360 if specified.

**proximity\_alert\_radius:** int | None

*Optional.* For live locations, a maximum distance for proximity alerts about approaching another chat member, in meters. Must be between 1 and 100000 if specified.

## InputMessageContent

**class** aiogram.types.input\_message\_content.**InputMessageContent**(\*\*extra\_data: Any)

This object represents the content of a message to be sent as a result of an inline query. Telegram clients currently support the following 5 types:

- `aiogram.types.input_text_message_content.InputTextMessageContent`
- `aiogram.types.input_location_message_content.InputLocationMessageContent`
- `aiogram.types.input_venue_message_content.InputVenueMessageContent`
- `aiogram.types.input_contact_message_content.InputContactMessageContent`
- `aiogram.types.input_invoice_message_content.InputInvoiceMessageContent`

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

## InputTextMessageContent

**class** aiogram.types.input\_text\_message\_content.**InputTextMessageContent**(\*, message\_text: str, parse\_mode: str | None = sentinel.UNSET\_PARSE\_MODE, entities: List[MessageEntity] | None = None, disable\_web\_page\_preview: bool | None = sentinel.UNSET\_DISABLE\_WEB\_PAGE\_PREVIEW, \*\*extra\_data: Any)

Represents the [content](#) of a text message to be sent as the result of an inline query.

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

**message\_text:** str

Text of the message to be sent, 1-4096 characters

**parse\_mode:** str | None

*Optional.* Mode for parsing entities in the message text. See [formatting options](#) for more details.

**entities:** List[MessageEntity] | None

*Optional.* List of special entities that appear in message text, which can be specified instead of *parse\_mode*

**disable\_web\_page\_preview:** bool | None

*Optional.* Disables link previews for links in the sent message

## InputVenueMessageContent

```
class aiogram.types.input_venue_message_content.InputVenueMessageContent(*, latitude: float,
                                                                           longitude: float, title:
                                                                           str, address: str,
                                                                           foursquare_id: str |
                                                                           None = None,
                                                                           foursquare_type: str
                                                                           | None = None,
                                                                           google_place_id: str
                                                                           | None = None,
                                                                           google_place_type:
                                                                           str | None = None,
                                                                           **extra_data: Any)
```

Represents the [content](#) of a venue message to be sent as the result of an inline query.

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

**latitude: float**

Latitude of the venue in degrees

**longitude: float**

Longitude of the venue in degrees

**title: str**

Name of the venue

**address: str**

Address of the venue

**foursquare\_id: str | None**

*Optional.* Foursquare identifier of the venue, if known

**foursquare\_type: str | None**

*Optional.* Foursquare type of the venue, if known. (For example, 'arts\_entertainment/default', 'arts\_entertainment/aquarium' or 'food/icecream'.)

**google\_place\_id: str | None**

*Optional.* Google Places identifier of the venue

**google\_place\_type: str | None**

*Optional.* Google Places type of the venue. (See [supported types](#).)

## SentWebAppMessage

```
class aiogram.types.sent_web_app_message.SentWebAppMessage(*, inline_message_id: str | None =
                                                             None, **extra_data: Any)
```

Describes an inline message sent by a [Web App](#) on behalf of a user.

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

**inline\_message\_id: str | None**

*Optional.* Identifier of the sent inline message. Available only if there is an [inline keyboard](#) attached to the message.

## Available types

### Animation

```
class aiogram.types.animation.Animation(*, file_id: str, file_unique_id: str, width: int, height: int,
                                         duration: int, thumbnail: PhotoSize | None = None, file_name:
                                         str | None = None, mime_type: str | None = None, file_size: int |
                                         None = None, **extra_data: Any)
```

This object represents an animation file (GIF or H.264/MPEG-4 AVC video without sound).

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

**file\_id: str**

Identifier for this file, which can be used to download or reuse the file

**file\_unique\_id: str**

Unique identifier for this file, which is supposed to be the same over time and for different bots. Can't be used to download or reuse the file.

**width: int**

Video width as defined by sender

**height: int**

Video height as defined by sender

**duration: int**

Duration of the video in seconds as defined by sender

**thumbnail: PhotoSize | None**

*Optional.* Animation thumbnail as defined by sender

**file\_name: str | None**

*Optional.* Original animation filename as defined by sender

**mime\_type: str | None**

*Optional.* MIME type of the file as defined by sender

**file\_size: int | None**

*Optional.* File size in bytes. It can be bigger than  $2^{31}$  and some programming languages may have difficulty/silent defects in interpreting it. But it has at most 52 significant bits, so a signed 64-bit integer or double-precision float type are safe for storing this value.

### Audio

```
class aiogram.types.audio.Audio(*, file_id: str, file_unique_id: str, duration: int, performer: str | None =
                                  None, title: str | None = None, file_name: str | None = None, mime_type:
                                  str | None = None, file_size: int | None = None, thumbnail: PhotoSize |
                                  None = None, **extra_data: Any)
```

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

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

**file\_id: str**

Identifier for this file, which can be used to download or reuse the file

**file\_unique\_id:** `str`

Unique identifier for this file, which is supposed to be the same over time and for different bots. Can't be used to download or reuse the file.

**duration:** `int`

Duration of the audio in seconds as defined by sender

**performer:** `str | None`

*Optional.* Performer of the audio as defined by sender or by audio tags

**title:** `str | None`

*Optional.* Title of the audio as defined by sender or by audio tags

**file\_name:** `str | None`

*Optional.* Original filename as defined by sender

**mime\_type:** `str | None`

*Optional.* MIME type of the file as defined by sender

**file\_size:** `int | None`

*Optional.* File size in bytes. It can be bigger than  $2^{31}$  and some programming languages may have difficulty/silent defects in interpreting it. But it has at most 52 significant bits, so a signed 64-bit integer or double-precision float type are safe for storing this value.

**thumbnail:** `PhotoSize | None`

*Optional.* Thumbnail of the album cover to which the music file belongs

## BotCommand

**class** `aiogram.types.bot_command.BotCommand(*, command: str, description: str, **extra_data: Any)`

This object represents a bot command.

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

**command:** `str`

Text of the command; 1-32 characters. Can contain only lowercase English letters, digits and underscores.

**description:** `str`

Description of the command; 1-256 characters.

## BotCommandScope

**class** `aiogram.types.bot_command_scope.BotCommandScope(**extra_data: Any)`

This object represents the scope to which bot commands are applied. Currently, the following 7 scopes are supported:

- `aiogram.types.bot_command_scope_default.BotCommandScopeDefault`
- `aiogram.types.bot_command_scope_all_private_chats.BotCommandScopeAllPrivateChats`
- `aiogram.types.bot_command_scope_all_group_chats.BotCommandScopeAllGroupChats`
- `aiogram.types.bot_command_scope_all_chat_administrators.BotCommandScopeAllChatAdministrators`
- `aiogram.types.bot_command_scope_chat.BotCommandScopeChat`

- `aiogram.types.bot_command_scope_chat_administrators.BotCommandScopeChatAdministrators`
- `aiogram.types.bot_command_scope_chat_member.BotCommandScopeChatMember`

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

## BotCommandScopeAllChatAdministrators

```
class aiogram.types.bot_command_scope_all_chat_administrators.BotCommandScopeAllChatAdministrators(*,
                                                                                                     type:
                                                                                                     ~typ-
                                                                                                     ing.Li
                                                                                                     'all_c
                                                                                                     =
                                                                                                     Bot-
                                                                                                     Com-
                                                                                                     mand
                                                                                                     Scope
                                                                                                     **ex-
                                                                                                     tra_d
                                                                                                     ~typ-
                                                                                                     ing.A
```

Represents the `scope` of bot commands, covering all group and supergroup chat administrators.

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

**type:** `Literal[BotCommandScopeType.ALL_CHAT_ADMINISTRATORS]`

Scope type, must be `all_chat_administrators`

## BotCommandScopeAllGroupChats

```
class aiogram.types.bot_command_scope_all_group_chats.BotCommandScopeAllGroupChats(*, type:
                                                                                                     ~typ-
                                                                                                     ing.Literal[<BotCommand
                                                                                                     'all_group_chats'>]
                                                                                                     = Bot-
                                                                                                     Com-
                                                                                                     mand-
                                                                                                     ScopeType.ALL_GROUP_
                                                                                                     **ex-
                                                                                                     tra_data:
                                                                                                     ~typ-
                                                                                                     ing.Any)
```

Represents the `scope` of bot commands, covering all group and supergroup chats.

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

**type:** `Literal[BotCommandScopeType.ALL_GROUP_CHATS]`

Scope type, must be `all_group_chats`



### BotCommandScopeAllPrivateChats

```
class aiogram.types.bot_command_scope_all_private_chats.BotCommandScopeAllPrivateChats(*,
                                                                                             type:
                                                                                             ~typing.Literal[<BotCommandScopeType.ALL_PRIVATE_CHATS>] =
                                                                                             BotCommandScopeType.ALL_PRIVATE_CHATS,
                                                                                             **extra_data:
                                                                                             ~typing.Any)
```

Represents the `scope` of bot commands, covering all private chats.

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

**type:** `Literal[BotCommandScopeType.ALL_PRIVATE_CHATS]`

Scope type, must be `all_private_chats`

### BotCommandScopeChat

```
class aiogram.types.bot_command_scope_chat.BotCommandScopeChat(*, type: ~typing.Literal[<BotCommandScopeType.CHAT: 'chat'>] =
                                                                                             BotCommandScopeType.CHAT,
                                                                                             chat_id: int | str, **extra_data:
                                                                                             ~typing.Any)
```

Represents the `scope` of bot commands, covering a specific chat.

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

**type:** `Literal[BotCommandScopeType.CHAT]`

Scope type, must be `chat`

**chat\_id:** `int | str`

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

### BotCommandScopeChatAdministrators

```
class aiogram.types.bot_command_scope_chat_administrators.BotCommandScopeChatAdministrators(*,
                                                                                               type:
                                                                                               ~typing.
                                                                                               Literal[<BotCommandScopeType.CHAT_ADMINISTRATORS,
                                                                                               BotCommandScopeType.CHAT_MEMBER>],
                                                                                               chat_id:
                                                                                               int
                                                                                               |
                                                                                               str,
                                                                                               **extra_data:
                                                                                               ~typing.
                                                                                               Any)
```

Represents the [scope](#) of bot commands, covering all administrators of a specific group or supergroup chat.

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

**type:** `Literal[BotCommandScopeType.CHAT_ADMINISTRATORS]`

Scope type, must be *chat\_administrators*

**chat\_id:** `int | str`

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

### BotCommandScopeChatMember

```
class aiogram.types.bot_command_scope_chat_member.BotCommandScopeChatMember(*, type: ~typing.
                                                                                               Literal[<BotCommandScopeType.CHAT_MEMBER,
                                                                                               BotCommandScopeType.CHAT_ADMINISTRATORS>],
                                                                                               chat_id:
                                                                                               int
                                                                                               |
                                                                                               str,
                                                                                               user_id:
                                                                                               int,
                                                                                               **extra_data:
                                                                                               ~typing.
                                                                                               Any)
```

Represents the [scope](#) of bot commands, covering a specific member of a group or supergroup chat.

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

**type:** `Literal[BotCommandScopeType.CHAT_MEMBER]`

Scope type, must be *chat\_member*

**chat\_id:** `int | str`

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

**user\_id:** `int`

Unique identifier of the target user

## BotCommandScopeDefault

```
class aiogram.types.bot_command_scope_default.BotCommandScopeDefault(*, type: ~typing.Literal[<BotCommandScopeType.DEFAULT, 'default'>] = BotCommandScopeType.DEFAULT, **extra_data: ~typing.Any)
```

Represents the default `scope` of bot commands. Default commands are used if no commands with a `narrower scope` are specified for the user.

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

**type:** `Literal[BotCommandScopeType.DEFAULT]`

Scope type, must be *default*

## BotDescription

```
class aiogram.types.bot_description.BotDescription(*, description: str, **extra_data: Any)
```

This object represents the bot's description.

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

**description:** `str`

The bot's description

## BotName

```
class aiogram.types.bot_name.BotName(*, name: str, **extra_data: Any)
```

This object represents the bot's name.

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

**name:** `str`

The bot's name

## BotShortDescription

```
class aiogram.types.bot_short_description.BotShortDescription(*, short_description: str, **extra_data: Any)
```

This object represents the bot's short description.

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

**short\_description:** `str`

The bot's short description

## CallbackQuery

```
class aiogram.types.callback_query.CallbackQuery(*, id: str, from_user: User, chat_instance: str,
                                                message: Message | None = None,
                                                inline_message_id: str | None = None, data: str |
                                                None = None, game_short_name: str | None = None,
                                                **extra_data: Any)
```

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.

**NOTE:** After the user presses a callback button, Telegram clients will display a progress bar until you call `aiogram.methods.answer_callback_query.AnswerCallbackQuery`. It is, therefore, necessary to react by calling `aiogram.methods.answer_callback_query.AnswerCallbackQuery` even if no notification to the user is needed (e.g., without specifying any of the optional parameters).

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

**id: str**

Unique identifier for this query

**from\_user: User**

Sender

**chat\_instance: str**

Global identifier, uniquely corresponding to the chat to which the message with the callback button was sent. Useful for high scores in `aiogram.methods.games.Games`.

**message: Message | None**

*Optional.* Message with the callback button that originated the query. Note that message content and message date will not be available if the message is too old

**inline\_message\_id: str | None**

*Optional.* Identifier of the message sent via the bot in inline mode, that originated the query.

**data: str | None**

*Optional.* Data associated with the callback button. Be aware that the message originated the query can contain no callback buttons with this data.

**game\_short\_name: str | None**

*Optional.* Short name of a [Game](#) to be returned, serves as the unique identifier for the game

**answer**(text: str | None = None, show\_alert: bool | None = None, url: str | None = None, cache\_time: int | None = None, \*\*kwargs: Any) → `AnswerCallbackQuery`

Shortcut for method `aiogram.methods.answer_callback_query.AnswerCallbackQuery` will automatically fill method attributes:

- `callback_query_id`

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. On success, True is returned.

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** – Text of the notification. If not specified, nothing will be shown to the user, 0-200 characters
- **show\_alert** – 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** – URL that will be opened by the user’s client. If you have created a `aiogram.types.game.Game` and accepted the conditions via `@BotFather`, specify the URL that opens your game - note that this will only work if the query comes from a `https://core.telegram.org/bots/api#inlinekeyboardbutton callback_game` button.
- **cache\_time** – The maximum amount of time in seconds that the result of the callback query may be cached client-side. Telegram apps will support caching starting in version 3.14. Defaults to 0.

### Returns

instance of method `aiogram.methods.answer_callback_query.AnswerCallbackQuery`

## Chat

```
class aiogram.types.chat.Chat(*, id: int, type: str, title: str | None = None, username: str | None = None,
    first_name: str | None = None, last_name: str | None = None, is_forum: bool
    | None = None, photo: ChatPhoto | None = None, active_usernames: List[str]
    | None = None, emoji_status_custom_emoji_id: str | None = None,
    emoji_status_expiration_date: datetime[datetime] | None = None, bio: str |
    None = None, has_private_forwards: bool | None = None,
    has_restricted_voice_and_video_messages: bool | None = None,
    join_to_send_messages: bool | None = None, join_by_request: bool | None =
    None, description: str | None = None, invite_link: str | None = None,
    pinned_message: Message | None = None, permissions: ChatPermissions |
    None = None, slow_mode_delay: int | None = None,
    message_auto_delete_time: int | None = None,
    has_aggressive_anti_spam_enabled: bool | None = None,
    has_hidden_members: bool | None = None, has_protected_content: bool |
    None = None, sticker_set_name: str | None = None, can_set_sticker_set: bool
    | None = None, linked_chat_id: int | None = None, location: ChatLocation |
    None = None, **extra_data: Any)
```

This object represents a chat.

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

**id: int**

Unique identifier for this chat. This number may have more than 32 significant bits and some programming languages may have difficulty/silent defects in interpreting it. But it has at most 52 significant bits, so a signed 64-bit integer or double-precision float type are safe for storing this identifier.

**type: str**

Type of chat, can be either ‘private’, ‘group’, ‘supergroup’ or ‘channel’

**title: str | None**

*Optional.* Title, for supergroups, channels and group chats

**username:** `str` | `None`

*Optional.* Username, for private chats, supergroups and channels if available

**first\_name:** `str` | `None`

*Optional.* First name of the other party in a private chat

**last\_name:** `str` | `None`

*Optional.* Last name of the other party in a private chat

**is\_forum:** `bool` | `None`

*Optional.* True, if the supergroup chat is a forum (has [topics](#) enabled)

**photo:** [ChatPhoto](#) | `None`

*Optional.* Chat photo. Returned only in [aiogram.methods.get\\_chat.GetChat](#).

**active\_usernames:** `List[str]` | `None`

*Optional.* If non-empty, the list of all [active chat usernames](#); for private chats, supergroups and channels. Returned only in [aiogram.methods.get\\_chat.GetChat](#).

**emoji\_status\_custom\_emoji\_id:** `str` | `None`

*Optional.* Custom emoji identifier of emoji status of the other party in a private chat. Returned only in [aiogram.methods.get\\_chat.GetChat](#).

**emoji\_status\_expiration\_date:** `DateTime` | `None`

*Optional.* Expiration date of the emoji status of the other party in a private chat, if any. Returned only in [aiogram.methods.get\\_chat.GetChat](#).

**bio:** `str` | `None`

*Optional.* Bio of the other party in a private chat. Returned only in [aiogram.methods.get\\_chat.GetChat](#).

**has\_private\_forwards:** `bool` | `None`

*Optional.* True, if privacy settings of the other party in the private chat allows to use `tg://user?id=<user_id>` links only in chats with the user. Returned only in [aiogram.methods.get\\_chat.GetChat](#).

**has\_restricted\_voice\_and\_video\_messages:** `bool` | `None`

*Optional.* True, if the privacy settings of the other party restrict sending voice and video note messages in the private chat. Returned only in [aiogram.methods.get\\_chat.GetChat](#).

**join\_to\_send\_messages:** `bool` | `None`

*Optional.* True, if users need to join the supergroup before they can send messages. Returned only in [aiogram.methods.get\\_chat.GetChat](#).

**join\_by\_request:** `bool` | `None`

*Optional.* True, if all users directly joining the supergroup need to be approved by supergroup administrators. Returned only in [aiogram.methods.get\\_chat.GetChat](#).

**description:** `str` | `None`

*Optional.* Description, for groups, supergroups and channel chats. Returned only in [aiogram.methods.get\\_chat.GetChat](#).

**invite\_link:** `str` | `None`

*Optional.* Primary invite link, for groups, supergroups and channel chats. Returned only in [aiogram.methods.get\\_chat.GetChat](#).

**pinned\_message:** `Message` | `None`

*Optional.* The most recent pinned message (by sending date). Returned only in `aiogram.methods.get_chat.GetChat`.

**permissions:** `ChatPermissions` | `None`

*Optional.* Default chat member permissions, for groups and supergroups. Returned only in `aiogram.methods.get_chat.GetChat`.

**slow\_mode\_delay:** `int` | `None`

*Optional.* For supergroups, the minimum allowed delay between consecutive messages sent by each unprivileged user; in seconds. Returned only in `aiogram.methods.get_chat.GetChat`.

**message\_auto\_delete\_time:** `int` | `None`

*Optional.* The time after which all messages sent to the chat will be automatically deleted; in seconds. Returned only in `aiogram.methods.get_chat.GetChat`.

**has\_aggressive\_anti\_spam\_enabled:** `bool` | `None`

*Optional.* True, if aggressive anti-spam checks are enabled in the supergroup. The field is only available to chat administrators. Returned only in `aiogram.methods.get_chat.GetChat`.

**has\_hidden\_members:** `bool` | `None`

*Optional.* True, if non-administrators can only get the list of bots and administrators in the chat. Returned only in `aiogram.methods.get_chat.GetChat`.

**has\_protected\_content:** `bool` | `None`

*Optional.* True, if messages from the chat can't be forwarded to other chats. Returned only in `aiogram.methods.get_chat.GetChat`.

**sticker\_set\_name:** `str` | `None`

*Optional.* For supergroups, name of group sticker set. Returned only in `aiogram.methods.get_chat.GetChat`.

**can\_set\_sticker\_set:** `bool` | `None`

*Optional.* True, if the bot can change the group sticker set. Returned only in `aiogram.methods.get_chat.GetChat`.

**linked\_chat\_id:** `int` | `None`

*Optional.* Unique identifier for the linked chat, i.e. the discussion group identifier for a channel and vice versa; for supergroups and channel chats. This identifier may be greater than 32 bits and some programming languages may have difficulty/silent defects in interpreting it. But it is smaller than 52 bits, so a signed 64 bit integer or double-precision float type are safe for storing this identifier. Returned only in `aiogram.methods.get_chat.GetChat`.

**location:** `ChatLocation` | `None`

*Optional.* For supergroups, the location to which the supergroup is connected. Returned only in `aiogram.methods.get_chat.GetChat`.

**property shifted\_id:** `int`

Returns shifted chat ID (positive and without “-100” prefix). Mostly used for private links like `t.me/c/chat_id/message_id`

Currently supergroup/channel IDs have 10-digit ID after “-100” prefix removed. However, these IDs might become 11-digit in future. So, first we remove “-100” prefix and count remaining number length. Then we multiple  $-1 * 10^{\text{(number\_length + 2)}}$  Finally, `self.id` is subtracted from that number

**property full\_name:** str

Get full name of the Chat.

For private chat it is first\_name + last\_name. For other chat types it is title.

**ban\_sender\_chat**(sender\_chat\_id: int, \*\*kwargs: Any) → *BanChatSenderChat*

Shortcut for method *aiogram.methods.ban\_chat\_sender\_chat.BanChatSenderChat* will automatically fill method attributes:

- chat\_id

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

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

**Parameters**

**sender\_chat\_id** – Unique identifier of the target sender chat

**Returns**

instance of method *aiogram.methods.ban\_chat\_sender\_chat.BanChatSenderChat*

**unban\_sender\_chat**(sender\_chat\_id: int, \*\*kwargs: Any) → *UnbanChatSenderChat*

Shortcut for method *aiogram.methods.unban\_chat\_sender\_chat.UnbanChatSenderChat* will automatically fill method attributes:

- chat\_id

Use this method to unban a previously banned channel chat in a supergroup or channel. The bot must be an administrator for this to work and must have the appropriate administrator rights. Returns True on success.

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

**Parameters**

**sender\_chat\_id** – Unique identifier of the target sender chat

**Returns**

instance of method *aiogram.methods.unban\_chat\_sender\_chat.UnbanChatSenderChat*

**get\_administrators**(\*\*kwargs: Any) → *GetChatAdministrators*

Shortcut for method *aiogram.methods.get\_chat\_administrators.GetChatAdministrators* will automatically fill method attributes:

- chat\_id

Use this method to get a list of administrators in a chat, which aren't bots. Returns an Array of *aiogram.types.chat\_member.ChatMember* objects.

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

**Returns**

instance of method *aiogram.methods.get\_chat\_administrators.GetChatAdministrators*

**delete\_message**(message\_id: int, \*\*kwargs: Any) → *DeleteMessage*

Shortcut for method *aiogram.methods.delete\_message.DeleteMessage* will automatically fill method attributes:

- chat\_id



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.
- Service messages about a supergroup, channel, or forum topic creation can't be deleted.
- A dice message in a private chat can only be deleted if it was sent more than 24 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.

Returns True on success.

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

#### Parameters

**message\_id** – Identifier of the message to delete

#### Returns

instance of method `aiogram.methods.delete_message.DeleteMessage`

**revoke\_invite\_link**(*invite\_link: str, \*\*kwargs: Any*) → *RevokeChatInviteLink*

Shortcut for method `aiogram.methods.revoke_chat_invite_link.RevokeChatInviteLink` will automatically fill method attributes:

- `chat_id`

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 administrator rights. Returns the revoked invite link as `aiogram.types.chat_invite_link.ChatInviteLink` object.

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

#### Parameters

**invite\_link** – The invite link to revoke

#### Returns

instance of method `aiogram.methods.revoke_chat_invite_link.RevokeChatInviteLink`

**edit\_invite\_link**(*invite\_link: str, name: str | None = None, expire\_date: datetime.datetime | datetime.timedelta | int | None = None, member\_limit: int | None = None, creates\_join\_request: bool | None = None, \*\*kwargs: Any*) → *EditChatInviteLink*

Shortcut for method `aiogram.methods.edit_chat_invite_link.EditChatInviteLink` will automatically fill method attributes:

- `chat_id`

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 administrator rights. Returns the edited invite link as a `aiogram.types.chat_invite_link.ChatInviteLink` object.

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

#### Parameters

- **invite\_link** – The invite link to edit
- **name** – Invite link name; 0-32 characters
- **expire\_date** – Point in time (Unix timestamp) when the link will expire
- **member\_limit** – The maximum number of users that can be members of the chat simultaneously after joining the chat via this invite link; 1-99999
- **creates\_join\_request** – True, if users joining the chat via the link need to be approved by chat administrators. If True, *member\_limit* can't be specified

**Returns**

instance of method `aiogram.methods.edit_chat_invite_link.EditChatInviteLink`

**create\_invite\_link**(*name: str | None = None, expire\_date: datetime.datetime | datetime.timedelta | int | None = None, member\_limit: int | None = None, creates\_join\_request: bool | None = None, \*\*kwargs: Any*) → *CreateChatInviteLink*

Shortcut for method `aiogram.methods.create_chat_invite_link.CreateChatInviteLink` will automatically fill method attributes:

- **chat\_id**

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 administrator rights. The link can be revoked using the method `aiogram.methods.revoke_chat_invite_link.RevokeChatInviteLink`. Returns the new invite link as `aiogram.types.chat_invite_link.ChatInviteLink` object.

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

**Parameters**

- **name** – Invite link name; 0-32 characters
- **expire\_date** – Point in time (Unix timestamp) when the link will expire
- **member\_limit** – The maximum number of users that can be members of the chat simultaneously after joining the chat via this invite link; 1-99999
- **creates\_join\_request** – True, if users joining the chat via the link need to be approved by chat administrators. If True, *member\_limit* can't be specified

**Returns**

instance of method `aiogram.methods.create_chat_invite_link.CreateChatInviteLink`

**export\_invite\_link**(*\*\*kwargs: Any*) → *ExportChatInviteLink*

Shortcut for method `aiogram.methods.export_chat_invite_link.ExportChatInviteLink` will automatically fill method attributes:

- **chat\_id**

Use this method to generate a new primary invite link for a chat; any previously generated primary link is revoked. The bot must be an administrator in the chat for this to work and must have the appropriate administrator rights. Returns the new invite link as *String* on success.

Note: Each administrator in a chat generates their own invite links. Bots can't use invite links generated by other administrators. If you want your bot to work with invite links, it will need to generate its own link using `aiogram.methods.export_chat_invite_link.ExportChatInviteLink` or by calling the `aiogram.methods.get_chat.GetChat` method. If your bot needs to generate a new primary invite link replacing its previous one, use `aiogram.methods.export_chat_invite_link.ExportChatInviteLink` again.

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

#### Returns

instance of method `aiogram.methods.export_chat_invite_link.ExportChatInviteLink`

**do**(*action: str, message\_thread\_id: int | None = None, \*\*kwargs: Any*) → *SendChatAction*

Shortcut for method `aiogram.methods.send_chat_action.SendChatAction` will automatically fill method attributes:

- `chat_id`

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 `aiogram.methods.send_chat_action.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

- **action** – 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, `choose_sticker` for stickers, `find_location` for location data, `record_video_note` or `upload_video_note` for video notes.
- **message\_thread\_id** – Unique identifier for the target message thread; supergroups only

#### Returns

instance of method `aiogram.methods.send_chat_action.SendChatAction`

**delete\_sticker\_set**(*\*\*kwargs: Any*) → *DeleteChatStickerSet*

Shortcut for method `aiogram.methods.delete_chat_sticker_set.DeleteChatStickerSet` will automatically fill method attributes:

- `chat_id`

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 administrator rights. Use the field `can_set_sticker_set` optionally returned in `aiogram.methods.get_chat.GetChat` requests to check if the bot can use this method. Returns True on success.

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

#### Returns

instance of method `aiogram.methods.delete_chat_sticker_set.DeleteChatStickerSet`

**set\_sticker\_set**(*sticker\_set\_name: str, \*\*kwargs: Any*) → *SetChatStickerSet*

Shortcut for method `aiogram.methods.set_chat_sticker_set.SetChatStickerSet` will automatically fill method attributes:

- `chat_id`

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 administrator rights. Use the field `can_set_sticker_set` optionally returned in `aiogram.methods.get_chat.GetChat` requests to check if the bot can use this method. Returns `True` on success.

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

**Parameters**

**sticker\_set\_name** – Name of the sticker set to be set as the group sticker set

**Returns**

instance of method `aiogram.methods.set_chat_sticker_set.SetChatStickerSet`

**get\_member**(*user\_id: int, \*\*kwargs: Any*) → *GetChatMember*

Shortcut for method `aiogram.methods.get_chat_member.GetChatMember` will automatically fill method attributes:

- `chat_id`

Use this method to get information about a member of a chat. The method is only guaranteed to work for other users if the bot is an administrator in the chat. Returns a `aiogram.types.chat_member.ChatMember` object on success.

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

**Parameters**

**user\_id** – Unique identifier of the target user

**Returns**

instance of method `aiogram.methods.get_chat_member.GetChatMember`

**get\_member\_count**(*\*\*kwargs: Any*) → *GetChatMemberCount*

Shortcut for method `aiogram.methods.get_chat_member_count.GetChatMemberCount` will automatically fill method attributes:

- `chat_id`

Use this method to get the number of members in a chat. Returns *Int* on success.

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

**Returns**

instance of method `aiogram.methods.get_chat_member_count.GetChatMemberCount`

**leave**(*\*\*kwargs: Any*) → *LeaveChat*

Shortcut for method `aiogram.methods.leave_chat.LeaveChat` will automatically fill method attributes:

- `chat_id`

Use this method for your bot to leave a group, supergroup or channel. Returns `True` on success.

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

**Returns**

instance of method `aiogram.methods.leave_chat.LeaveChat`

**unpin\_all\_messages**(*\*\*kwargs: Any*) → *UnpinAllChatMessages*

Shortcut for method `aiogram.methods.unpin_all_chat_messages.UnpinAllChatMessages` will automatically fill method attributes:

- `chat_id`

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’ administrator right in a supergroup or ‘can\_edit\_messages’ administrator right in a channel. Returns True on success.

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

#### Returns

instance of method `aiogram.methods.unpin_all_chat_messages.UnpinAllChatMessages`

**unpin\_message**(*message\_id*: int | None = None, *\*\*kwargs*: Any) → *UnpinChatMessage*

Shortcut for method `aiogram.methods.unpin_chat_message.UnpinChatMessage` will automatically fill method attributes:

- `chat_id`

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’ administrator right in a supergroup or ‘can\_edit\_messages’ administrator right in a channel. Returns True on success.

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

#### Parameters

**message\_id** – Identifier of a message to unpin. If not specified, the most recent pinned message (by sending date) will be unpinned.

#### Returns

instance of method `aiogram.methods.unpin_chat_message.UnpinChatMessage`

**pin\_message**(*message\_id*: int, *disable\_notification*: bool | None = None, *\*\*kwargs*: Any) → *PinChatMessage*

Shortcut for method `aiogram.methods.pin_chat_message.PinChatMessage` will automatically fill method attributes:

- `chat_id`

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’ administrator right in a supergroup or ‘can\_edit\_messages’ administrator right in a channel. Returns True on success.

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

#### Parameters

- **message\_id** – Identifier of a message to pin
- **disable\_notification** – Pass True if it is not necessary to send a notification to all chat members about the new pinned message. Notifications are always disabled in channels and private chats.

#### Returns

instance of method `aiogram.methods.pin_chat_message.PinChatMessage`

**set\_administrator\_custom\_title**(*user\_id*: int, *custom\_title*: str, *\*\*kwargs*: Any) → *SetChatAdministratorCustomTitle*

Shortcut for method `aiogram.methods.set_chat_administrator_custom_title.SetChatAdministratorCustomTitle` will automatically fill method attributes:

- `chat_id`

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

instance of method `aiogram.methods.set_chat_administrator_custom_title.SetChatAdministratorCustomTitle`

**set\_permissions**(*permissions: ChatPermissions, use\_independent\_chat\_permissions: bool | None = None, \*\*kwargs: Any*) → `SetChatPermissions`

Shortcut for method `aiogram.methods.set_chat_permissions.SetChatPermissions` will automatically fill method attributes:

- **chat\_id**

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` administrator rights. Returns True on success.

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

#### Parameters

- **permissions** – A JSON-serialized object for new default chat permissions
- **use\_independent\_chat\_permissions** – Pass True if chat permissions are set independently. Otherwise, the `can_send_other_messages` and `can_add_web_page_previews` permissions will imply the `can_send_messages`, `can_send_audios`, `can_send_documents`, `can_send_photos`, `can_send_videos`, `can_send_video_notes`, and `can_send_voice_notes` permissions; the `can_send_polls` permission will imply the `can_send_messages` permission.

#### Returns

instance of method `aiogram.methods.set_chat_permissions.SetChatPermissions`

**promote**(*user\_id: int, is\_anonymous: bool | None = None, can\_manage\_chat: bool | None = None, can\_post\_messages: bool | None = None, can\_edit\_messages: bool | None = None, can\_delete\_messages: bool | None = None, can\_manage\_video\_chats: bool | None = None, can\_restrict\_members: bool | None = None, can\_promote\_members: bool | None = None, can\_change\_info: bool | None = None, can\_invite\_users: bool | None = None, can\_pin\_messages: bool | None = None, can\_manage\_topics: bool | None = None, \*\*kwargs: Any*) → `PromoteChatMember`

Shortcut for method `aiogram.methods.promote_chat_member.PromoteChatMember` will automatically fill method attributes:

- **chat\_id**

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 administrator rights. Pass False for all boolean parameters to demote a user. Returns True on success.

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

#### Parameters

- **user\_id** – Unique identifier of the target user
- **is\_anonymous** – Pass True if the administrator's presence in the chat is hidden
- **can\_manage\_chat** – 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\_post\_messages** – Pass True if the administrator can create channel posts, channels only
- **can\_edit\_messages** – Pass True if the administrator can edit messages of other users and can pin messages, channels only
- **can\_delete\_messages** – Pass True if the administrator can delete messages of other users
- **can\_manage\_video\_chats** – Pass True if the administrator can manage video chats
- **can\_restrict\_members** – Pass True if the administrator can restrict, ban or unban chat members
- **can\_promote\_members** – Pass True if the administrator can add new administrators with a subset of their own privileges or demote administrators that they have promoted, directly or indirectly (promoted by administrators that were appointed by him)
- **can\_change\_info** – Pass True if the administrator can change chat title, photo and other settings
- **can\_invite\_users** – Pass True if the administrator can invite new users to the chat
- **can\_pin\_messages** – Pass True if the administrator can pin messages, supergroups only
- **can\_manage\_topics** – Pass True if the user is allowed to create, rename, close, and reopen forum topics, supergroups only

#### Returns

instance of method `aiogram.methods.promote_chat_member.PromoteChatMember`

**restrict**(*user\_id: int, permissions: ChatPermissions, use\_independent\_chat\_permissions: bool | None = None, until\_date: datetime.datetime | datetime.timedelta | int | None = None, \*\*kwargs: Any*) → `RestrictChatMember`

Shortcut for method `aiogram.methods.restrict_chat_member.RestrictChatMember` will automatically fill method attributes:

- **chat\_id**

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 administrator rights. Pass True for all permissions to lift restrictions from a user. Returns True on success.

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

#### Parameters

- **user\_id** – Unique identifier of the target user
- **permissions** – A JSON-serialized object for new user permissions
- **use\_independent\_chat\_permissions** – Pass True if chat permissions are set independently. Otherwise, the `can_send_other_messages` and `can_add_web_page_previews` permissions will imply the `can_send_messages`, `can_send_audios`, `can_send_documents`, `can_send_photos`, `can_send_videos`, `can_send_video_notes`, and `can_send_voice_notes`



permissions; the `can_send_polls` permission will imply the `can_send_messages` permission.

- **until\_date** – Date when restrictions will be lifted for the user, unix time. If user is restricted for more than 366 days or less than 30 seconds from the current time, they are considered to be restricted forever

#### Returns

instance of method `aiogram.methods.restrict_chat_member.RestrictChatMember`

**unban**(*user\_id: int, only\_if\_banned: bool | None = None, \*\*kwargs: Any*) → *UnbanChatMember*

Shortcut for method `aiogram.methods.unban_chat_member.UnbanChatMember` will automatically fill method attributes:

- `chat_id`

Use this method to unban a previously banned 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** – Unique identifier of the target user
- **only\_if\_banned** – Do nothing if the user is not banned

#### Returns

instance of method `aiogram.methods.unban_chat_member.UnbanChatMember`

**ban**(*user\_id: int, until\_date: datetime.datetime | datetime.timedelta | int | None = None, revoke\_messages: bool | None = None, \*\*kwargs: Any*) → *BanChatMember*

Shortcut for method `aiogram.methods.ban_chat_member.BanChatMember` will automatically fill method attributes:

- `chat_id`

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 administrator rights. Returns **True** on success.

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

#### Parameters

- **user\_id** – Unique identifier of the target user
- **until\_date** – 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.

#### Returns

instance of method `aiogram.methods.ban_chat_member.BanChatMember`



**set\_description**(*description: str | None = None, \*\*kwargs: Any*) → *SetChatDescription*

Shortcut for method `aiogram.methods.set_chat_description.SetChatDescription` will automatically fill method attributes:

- `chat_id`

Use this method to change the description of a group, a supergroup or a channel. The bot must be an administrator in the chat for this to work and must have the appropriate administrator rights. Returns True on success.

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

#### Parameters

**description** – New chat description, 0-255 characters

#### Returns

instance of method `aiogram.methods.set_chat_description.SetChatDescription`

**set\_title**(*title: str, \*\*kwargs: Any*) → *SetChatTitle*

Shortcut for method `aiogram.methods.set_chat_title.SetChatTitle` will automatically fill method attributes:

- `chat_id`

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 administrator rights. Returns True on success.

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

#### Parameters

**title** – New chat title, 1-128 characters

#### Returns

instance of method `aiogram.methods.set_chat_title.SetChatTitle`

**delete\_photo**(*\*\*kwargs: Any*) → *DeleteChatPhoto*

Shortcut for method `aiogram.methods.delete_chat_photo.DeleteChatPhoto` will automatically fill method attributes:

- `chat_id`

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 administrator rights. Returns True on success.

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

#### Returns

instance of method `aiogram.methods.delete_chat_photo.DeleteChatPhoto`

**set\_photo**(*photo: InputFile, \*\*kwargs: Any*) → *SetChatPhoto*

Shortcut for method `aiogram.methods.set_chat_photo.SetChatPhoto` will automatically fill method attributes:

- `chat_id`

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 administrator rights. Returns True on success.

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

**Parameters**

**photo** – New chat photo, uploaded using multipart/form-data

**Returns**

instance of method `aiogram.methods.set_chat_photo.SetChatPhoto`

**unpin\_all\_general\_forum\_topic\_messages**(\*\*kwargs: Any) → *UnpinAllGeneralForumTopicMessages*

Shortcut for method `aiogram.methods.unpin_all_general_forum_topic_messages.UnpinAllGeneralForumTopicMessages` will automatically fill method attributes:

- `chat_id`

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

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

**Returns**

instance of method `aiogram.methods.unpin_all_general_forum_topic_messages.UnpinAllGeneralForumTopicMessages`

## ChatAdministratorRights

```
class aiogram.types.chat_administrator_rights.ChatAdministratorRights(*, is_anonymous: bool,
    can_manage_chat: bool,
    can_delete_messages: bool,
    can_manage_video_chats: bool,
    can_restrict_members: bool,
    can_promote_members: bool, can_change_info: bool, can_invite_users: bool,
    can_post_messages: bool | None = None,
    can_edit_messages: bool | None = None,
    can_pin_messages: bool | None = None,
    can_manage_topics: bool | None = None,
    **extra_data: Any)
```

Represents the rights of an administrator in a chat.

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

**is\_anonymous:** bool

True, if the user's presence in the chat is hidden

**can\_manage\_chat:** bool

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\_delete\_messages: bool**

True, if the administrator can delete messages of other users

**can\_manage\_video\_chats: bool**

True, if the administrator can manage video chats

**can\_restrict\_members: bool**

True, if the administrator can restrict, ban or unban chat members

**can\_promote\_members: bool**

True, if the administrator can add new administrators with a subset of their own privileges or demote administrators that they have promoted, directly or indirectly (promoted by administrators that were appointed by the user)

**can\_change\_info: bool**

True, if the user is allowed to change the chat title, photo and other settings

**can\_invite\_users: bool**

True, if the user is allowed to invite new users to the chat

**can\_post\_messages: bool | None***Optional.* True, if the administrator can post in the channel; channels only**can\_edit\_messages: bool | None***Optional.* True, if the administrator can edit messages of other users and can pin messages; channels only**can\_pin\_messages: bool | None***Optional.* True, if the user is allowed to pin messages; groups and supergroups only**can\_manage\_topics: bool | None***Optional.* True, if the user is allowed to create, rename, close, and reopen forum topics; supergroups only

## ChatInviteLink

```
class aiogram.types.chat_invite_link.ChatInviteLink(*, invite_link: str, creator: User,
                                                    creates_join_request: bool, is_primary: bool,
                                                    is_revoked: bool, name: str | None = None,
                                                    expire_date: datetime[datetime] | None = None,
                                                    member_limit: int | None = None,
                                                    pending_join_request_count: int | None = None,
                                                    **extra_data: Any)
```

Represents an invite link for a chat.

Source: <https://core.telegram.org/bots/api#chatinvitelink>**invite\_link: str**

The invite link. If the link was created by another chat administrator, then the second part of the link will be replaced with '...'.

**creator: User**

Creator of the link

**creates\_join\_request: bool**

True, if users joining the chat via the link need to be approved by chat administrators

**is\_primary:** `bool`

True, if the link is primary

**is\_revoked:** `bool`

True, if the link is revoked

**name:** `str | None`

*Optional.* Invite link name

**expire\_date:** `DateTime | None`

*Optional.* Point in time (Unix timestamp) when the link will expire or has been expired

**member\_limit:** `int | None`

*Optional.* The maximum number of users that can be members of the chat simultaneously after joining the chat via this invite link; 1-99999

**pending\_join\_request\_count:** `int | None`

*Optional.* Number of pending join requests created using this link

## ChatJoinRequest

```
class aiogram.types.chat_join_request.ChatJoinRequest(*, chat: Chat, from_user: User, user_chat_id:
    int, date: datetime[datetime], bio: str | None
    = None, invite_link: ChatInviteLink | None =
    None, **extra_data: Any)
```

Represents a join request sent to a chat.

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

**chat:** `Chat`

Chat to which the request was sent

**from\_user:** `User`

User that sent the join request

**user\_chat\_id:** `int`

Identifier of a private chat with the user who sent the join request. This number may have more than 32 significant bits and some programming languages may have difficulty/silent defects in interpreting it. But it has at most 52 significant bits, so a 64-bit integer or double-precision float type are safe for storing this identifier. The bot can use this identifier for 24 hours to send messages until the join request is processed, assuming no other administrator contacted the user.

**date:** `DateTime`

Date the request was sent in Unix time

**bio:** `str | None`

*Optional.* Bio of the user.

**invite\_link:** `ChatInviteLink | None`

*Optional.* Chat invite link that was used by the user to send the join request

**approve**(\*\*kwargs: Any) → `ApproveChatJoinRequest`

Shortcut for method `aiogram.methods.approve_chat_join_request.ApproveChatJoinRequest` will automatically fill method attributes:

- `chat_id`

- `user_id`

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

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

#### Returns

instance of method `aiogram.methods.approve_chat_join_request.ApproveChatJoinRequest`

**decline**(\*\*kwargs: Any) → *DeclineChatJoinRequest*

Shortcut for method `aiogram.methods.decline_chat_join_request.DeclineChatJoinRequest` will automatically fill method attributes:

- `chat_id`
- `user_id`

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

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

#### Returns

instance of method `aiogram.methods.decline_chat_join_request.DeclineChatJoinRequest`

**answer**(text: str, message\_thread\_id: int | None = None, parse\_mode: str | None = sentinel.UNSET\_PARSE\_MODE, entities: List[MessageEntity] | None = None, disable\_web\_page\_preview: bool | None = sentinel.UNSET\_DISABLE\_WEB\_PAGE\_PREVIEW, disable\_notification: bool | None = None, protect\_content: bool | None = sentinel.UNSET\_PROTECT\_CONTENT, reply\_to\_message\_id: int | None = None, allow\_sending\_without\_reply: bool | None = None, reply\_markup: InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply | None = None, \*\*kwargs: Any) → *SendMessage*

Shortcut for method `aiogram.methods.send_message.SendMessage` will automatically fill method attributes:

- `chat_id`

Use this method to send text messages. On success, the sent `aiogram.types.message.Message` is returned.

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

#### Parameters

- **text** – Text of the message to be sent, 1-4096 characters after entities parsing
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **parse\_mode** – Mode for parsing entities in the message text. See [formatting options](#) for more details.
- **entities** – A JSON-serialized list of special entities that appear in message text, which can be specified instead of `parse_mode`
- **disable\_web\_page\_preview** – Disables link previews for links in this message
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.

- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method [aiogram.methods.send\\_message.SendMessage](#)

```
answer_pm(text: str, message_thread_id: int | None = None, parse_mode: str | None =  
    sentinel.UNSET_PARSE_MODE, entities: List[MessageEntity] | None = None,  
    disable_web_page_preview: bool | None = sentinel.UNSET_DISABLE_WEB_PAGE_PREVIEW,  
    disable_notification: bool | None = None, protect_content: bool | None =  
    sentinel.UNSET_PROTECT_CONTENT, reply_to_message_id: int | None = None,  
    allow_sending_without_reply: bool | None = None, reply_markup: InlineKeyboardMarkup |  
    ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply | None = None, **kwargs: Any)  
    → SendMessage
```

Shortcut for method [aiogram.methods.send\\_message.SendMessage](#) will automatically fill method attributes:

- **chat\_id**

Use this method to send text messages. On success, the sent [aiogram.types.message.Message](#) is returned.

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

#### Parameters

- **text** – Text of the message to be sent, 1-4096 characters after entities parsing
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **parse\_mode** – Mode for parsing entities in the message text. See [formatting options](#) for more details.
- **entities** – A JSON-serialized list of special entities that appear in message text, which can be specified instead of *parse\_mode*
- **disable\_web\_page\_preview** – Disables link previews for links in this message
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method [aiogram.methods.send\\_message.SendMessage](#)

**answer\_animation**(*animation*: [InputFile](#) | *str*, *message\_thread\_id*: *int* | *None* = *None*, *duration*: *int* | *None* = *None*, *width*: *int* | *None* = *None*, *height*: *int* | *None* = *None*, *thumbnail*: [InputFile](#) | *str* | *None* = *None*, *caption*: *str* | *None* = *None*, *parse\_mode*: *str* | *None* = *sentinel.UNSET\_PARSE\_MODE*, *caption\_entities*: *List*[[MessageEntity](#)] | *None* = *None*, *has\_spoiler*: *bool* | *None* = *None*, *disable\_notification*: *bool* | *None* = *None*, *protect\_content*: *bool* | *None* = *sentinel.UNSET\_PROTECT\_CONTENT*, *reply\_to\_message\_id*: *int* | *None* = *None*, *allow\_sending\_without\_reply*: *bool* | *None* = *None*, *reply\_markup*: [InlineKeyboardMarkup](#) | [ReplyKeyboardMarkup](#) | [ReplyKeyboardRemove](#) | [ForceReply](#) | *None* = *None*, *\*\*kwargs*: *Any*) → [SendAnimation](#)

Shortcut for method [aiogram.methods.send\\_animation.SendAnimation](#) will automatically fill method attributes:

- `chat_id`

Use this method to send animation files (GIF or H.264/MPEG-4 AVC video without sound). On success, the sent [aiogram.types.message.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** – 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. [More information on Sending Files](#) »
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **duration** – Duration of sent animation in seconds
- **width** – Animation width
- **height** – Animation height
- **thumbnail** – Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can't be reused and can be only uploaded as a new file, so you can pass 'attach://<file\_attach\_name>' if the thumbnail was uploaded using multipart/form-data under <file\_attach\_name>. [More information on Sending Files](#) »
- **caption** – Animation caption (may also be used when resending animation by `file_id`), 0-1024 characters after entities parsing
- **parse\_mode** – Mode for parsing entities in the animation caption. See [formatting options](#) for more details.
- **caption\_entities** – A JSON-serialized list of special entities that appear in the caption, which can be specified instead of `parse_mode`
- **has\_spoiler** – Pass `True` if the animation needs to be covered with a spoiler animation
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message

- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method [aiogram.methods.send\\_animation.SendAnimation](#)

**answer\_animation\_pm**(*animation*: [InputFile](#) | *str*, *message\_thread\_id*: *int* | *None* = *None*, *duration*: *int* | *None* = *None*, *width*: *int* | *None* = *None*, *height*: *int* | *None* = *None*, *thumbnail*: [InputFile](#) | *str* | *None* = *None*, *caption*: *str* | *None* = *None*, *parse\_mode*: *str* | *None* = *sentinel.UNSET\_PARSE\_MODE*, *caption\_entities*: *List*[[MessageEntity](#)] | *None* = *None*, *has\_spoiler*: *bool* | *None* = *None*, *disable\_notification*: *bool* | *None* = *None*, *protect\_content*: *bool* | *None* = *sentinel.UNSET\_PROTECT\_CONTENT*, *reply\_to\_message\_id*: *int* | *None* = *None*, *allow\_sending\_without\_reply*: *bool* | *None* = *None*, *reply\_markup*: [InlineKeyboardMarkup](#) | [ReplyKeyboardMarkup](#) | [ReplyKeyboardRemove](#) | [ForceReply](#) | *None* = *None*, *\*\*kwargs*: *Any*) → [SendAnimation](#)

Shortcut for method [aiogram.methods.send\\_animation.SendAnimation](#) will automatically fill method attributes:

- **chat\_id**

Use this method to send animation files (GIF or H.264/MPEG-4 AVC video without sound). On success, the sent [aiogram.types.message.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** – 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. [More information on Sending Files](#) »
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **duration** – Duration of sent animation in seconds
- **width** – Animation width
- **height** – Animation height
- **thumbnail** – Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can't be reused and can be only uploaded as a new file, so you can pass 'attach://<file\_attach\_name>' if the thumbnail was uploaded using multipart/form-data under <file\_attach\_name>. [More information on Sending Files](#) »
- **caption** – Animation caption (may also be used when resending animation by *file\_id*), 0-1024 characters after entities parsing
- **parse\_mode** – Mode for parsing entities in the animation caption. See [formatting options](#) for more details.



- **caption\_entities** – A JSON-serialized list of special entities that appear in the caption, which can be specified instead of *parse\_mode*
- **has\_spoiler** – Pass True if the animation needs to be covered with a spoiler animation
- **disable\_notification** – Sends the message *silently*. Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method `aiogram.methods.send_animation.SendAnimation`

**answer\_audio**(*audio*: `InputFile` | *str*, *message\_thread\_id*: *int* | *None* = *None*, *caption*: *str* | *None* = *None*, *parse\_mode*: *str* | *None* = *sentinel.UNSET\_PARSE\_MODE*, *caption\_entities*: *List[MessageEntity]* | *None* = *None*, *duration*: *int* | *None* = *None*, *performer*: *str* | *None* = *None*, *title*: *str* | *None* = *None*, *thumbnail*: `InputFile` | *str* | *None* = *None*, *disable\_notification*: *bool* | *None* = *None*, *protect\_content*: *bool* | *None* = *sentinel.UNSET\_PROTECT\_CONTENT*, *reply\_to\_message\_id*: *int* | *None* = *None*, *allow\_sending\_without\_reply*: *bool* | *None* = *None*, *reply\_markup*: `InlineKeyboardMarkup` | `ReplyKeyboardMarkup` | `ReplyKeyboardRemove` | `ForceReply` | *None* = *None*, *\*\*kwargs*: *Any*) → *SendAudio*

Shortcut for method `aiogram.methods.send_audio.SendAudio` will automatically fill method attributes:

- **chat\_id**

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 or .M4A format. On success, the sent `aiogram.types.message.Message` is returned. Bots can currently send audio files of up to 50 MB in size, this limit may be changed in the future. For sending voice messages, use the `aiogram.methods.send_voice.SendVoice` method instead.

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

#### Parameters

- **audio** – Audio file to send. Pass a *file\_id* as *String* to send an audio file that exists on the Telegram servers (recommended), pass an HTTP URL as a *String* for Telegram to get an audio file from the Internet, or upload a new one using multipart/form-data. *More information on Sending Files »*
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **caption** – Audio caption, 0-1024 characters after entities parsing
- **parse\_mode** – Mode for parsing entities in the audio caption. See *formatting options* for more details.
- **caption\_entities** – A JSON-serialized list of special entities that appear in the caption, which can be specified instead of *parse\_mode*
- **duration** – Duration of the audio in seconds
- **performer** – Performer

- **title** – Track name
- **thumbnail** – Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can't be reused and can be only uploaded as a new file, so you can pass 'attach://<file\_attach\_name>' if the thumbnail was uploaded using multipart/form-data under <file\_attach\_name>. [More information on Sending Files](#) »
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method [aiogram.methods.send\\_audio.SendAudio](#)

**answer\_audio\_pm**(audio: [InputFile](#) | str, message\_thread\_id: int | None = None, caption: str | None = None, parse\_mode: str | None = [sentinel.UNSET\\_PARSE\\_MODE](#), caption\_entities: List[[MessageEntity](#)] | None = None, duration: int | None = None, performer: str | None = None, title: str | None = None, thumbnail: [InputFile](#) | str | None = None, disable\_notification: bool | None = None, protect\_content: bool | None = [sentinel.UNSET\\_PROTECT\\_CONTENT](#), reply\_to\_message\_id: int | None = None, allow\_sending\_without\_reply: bool | None = None, reply\_markup: [InlineKeyboardMarkup](#) | [ReplyKeyboardMarkup](#) | [ReplyKeyboardRemove](#) | [ForceReply](#) | None = None, \*\*kwargs: Any) → [SendAudio](#)

Shortcut for method [aiogram.methods.send\\_audio.SendAudio](#) will automatically fill method attributes:

- chat\_id

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 or .M4A format. On success, the sent [aiogram.types.message.Message](#) is returned. Bots can currently send audio files of up to 50 MB in size, this limit may be changed in the future. For sending voice messages, use the [aiogram.methods.send\\_voice.SendVoice](#) method instead.

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

#### Parameters

- **audio** – Audio file to send. Pass a file\_id as String to send an audio file that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get an audio file from the Internet, or upload a new one using multipart/form-data. [More information on Sending Files](#) »
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **caption** – Audio caption, 0-1024 characters after entities parsing
- **parse\_mode** – Mode for parsing entities in the audio caption. See [formatting options](#) for more details.

- **caption\_entities** – A JSON-serialized list of special entities that appear in the caption, which can be specified instead of *parse\_mode*
- **duration** – Duration of the audio in seconds
- **performer** – Performer
- **title** – Track name
- **thumbnail** – Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can't be reused and can be only uploaded as a new file, so you can pass 'attach://<file\_attach\_name>' if the thumbnail was uploaded using multipart/form-data under <file\_attach\_name>. *More information on Sending Files*  
»
- **disable\_notification** – Sends the message *silently*. Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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**

instance of method *aiogram.methods.send\_audio.SendAudio*

**answer\_contact**(*phone\_number: str, first\_name: str, message\_thread\_id: int | None = None, last\_name: str | None = None, vcard: str | None = None, disable\_notification: bool | None = None, protect\_content: bool | None = sentinel.UNSET\_PROTECT\_CONTENT, reply\_to\_message\_id: int | None = None, allow\_sending\_without\_reply: bool | None = None, reply\_markup: InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply | None = None, \*\*kwargs: Any*) → *SendContact*

Shortcut for method *aiogram.methods.send\_contact.SendContact* will automatically fill method attributes:

- **chat\_id**

Use this method to send phone contacts. On success, the sent *aiogram.types.message.Message* is returned.

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

**Parameters**

- **phone\_number** – Contact's phone number
- **first\_name** – Contact's first name
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **last\_name** – Contact's last name
- **vcard** – Additional data about the contact in the form of a *vCard*, 0-2048 bytes

- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method [aiogram.methods.send\\_contact.SendContact](#)

**answer\_contact\_pm**(*phone\_number: str, first\_name: str, message\_thread\_id: int | None = None, last\_name: str | None = None, vcard: str | None = None, disable\_notification: bool | None = None, protect\_content: bool | None = sentinel.UNSET\_PROTECT\_CONTENT, reply\_to\_message\_id: int | None = None, allow\_sending\_without\_reply: bool | None = None, reply\_markup: InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply | None = None, \*\*kwargs: Any*) → [SendContact](#)

Shortcut for method [aiogram.methods.send\\_contact.SendContact](#) will automatically fill method attributes:

- **chat\_id**

Use this method to send phone contacts. On success, the sent [aiogram.types.message.Message](#) is returned.

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

#### Parameters

- **phone\_number** – Contact's phone number
- **first\_name** – Contact's first name
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **last\_name** – Contact's last name
- **vcard** – Additional data about the contact in the form of a [vCard](#), 0-2048 bytes
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method [aiogram.methods.send\\_contact.SendContact](#)

```
answer_document(document: InputFile | str, message_thread_id: int | None = None, thumbnail: InputFile | str | None = None, caption: str | None = None, parse_mode: str | None = sentinel.UNSET_PARSE_MODE, caption_entities: List[MessageEntity] | None = None, disable_content_type_detection: bool | None = None, disable_notification: bool | None = None, protect_content: bool | None = sentinel.UNSET_PROTECT_CONTENT, reply_to_message_id: int | None = None, allow_sending_without_reply: bool | None = None, reply_markup: InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply | None = None, **kwargs: Any) → SendDocument
```

Shortcut for method `aiogram.methods.send_document.SendDocument` will automatically fill method attributes:

- `chat_id`

Use this method to send general files. On success, the sent `aiogram.types.message.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** – File to send. Pass a `file_id` as String to send a file that exists on the Telegram servers (recommended), 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 information on Sending Files](#) »
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **thumbnail** – Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can't be reused and can be only uploaded as a new file, so you can pass 'attach://<file\_attach\_name>' if the thumbnail was uploaded using multipart/form-data under <file\_attach\_name>. [More information on Sending Files](#) »
- **caption** – Document caption (may also be used when resending documents by `file_id`), 0-1024 characters after entities parsing
- **parse\_mode** – Mode for parsing entities in the document caption. See [formatting options](#) for more details.
- **caption\_entities** – A JSON-serialized list of special entities that appear in the caption, which can be specified instead of `parse_mode`
- **disable\_content\_type\_detection** – Disables automatic server-side content type detection for files uploaded using multipart/form-data
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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**

instance of method `aiogram.methods.send_document.SendDocument`

```
answer_document_pm(document: InputFile | str, message_thread_id: int | None = None, thumbnail:
    InputFile | str | None = None, caption: str | None = None, parse_mode: str | None =
    sentinel.UNSET_PARSE_MODE, caption_entities: List[MessageEntity] | None =
    None, disable_content_type_detection: bool | None = None, disable_notification:
    bool | None = None, protect_content: bool | None =
    sentinel.UNSET_PROTECT_CONTENT, reply_to_message_id: int | None = None,
    allow_sending_without_reply: bool | None = None, reply_markup:
    InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove |
    ForceReply | None = None, **kwargs: Any) → SendDocument
```

Shortcut for method `aiogram.methods.send_document.SendDocument` will automatically fill method attributes:

- `chat_id`

Use this method to send general files. On success, the sent `aiogram.types.message.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** – File to send. Pass a `file_id` as String to send a file that exists on the Telegram servers (recommended), 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 information on Sending Files](#) »*
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **thumbnail** – Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can't be reused and can be only uploaded as a new file, so you can pass 'attach://<file\_attach\_name>' if the thumbnail was uploaded using multipart/form-data under <file\_attach\_name>. *[More information on Sending Files](#) »*
- **caption** – Document caption (may also be used when resending documents by `file_id`), 0-1024 characters after entities parsing
- **parse\_mode** – Mode for parsing entities in the document caption. See [formatting options](#) for more details.
- **caption\_entities** – A JSON-serialized list of special entities that appear in the caption, which can be specified instead of `parse_mode`
- **disable\_content\_type\_detection** – Disables automatic server-side content type detection for files uploaded using multipart/form-data
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found

- **reply\_markup** – 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

instance of method `aiogram.methods.send_document.SendDocument`

```
answer_game(game_short_name: str, message_thread_id: int | None = None, disable_notification: bool | None = None, protect_content: bool | None = sentinel.UNSET_PROTECT_CONTENT, reply_to_message_id: int | None = None, allow_sending_without_reply: bool | None = None, reply_markup: InlineKeyboardMarkup | None = None, **kwargs: Any) → SendGame
```

Shortcut for method `aiogram.methods.send_game.SendGame` will automatically fill method attributes:

- `chat_id`

Use this method to send a game. On success, the sent `aiogram.types.message.Message` is returned.

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

#### Parameters

- **game\_short\_name** – Short name of the game, serves as the unique identifier for the game. Set up your games via [@BotFather](#).
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method `aiogram.methods.send_game.SendGame`

```
answer_game_pm(game_short_name: str, message_thread_id: int | None = None, disable_notification: bool | None = None, protect_content: bool | None = sentinel.UNSET_PROTECT_CONTENT, reply_to_message_id: int | None = None, allow_sending_without_reply: bool | None = None, reply_markup: InlineKeyboardMarkup | None = None, **kwargs: Any) → SendGame
```

Shortcut for method `aiogram.methods.send_game.SendGame` will automatically fill method attributes:

- `chat_id`

Use this method to send a game. On success, the sent `aiogram.types.message.Message` is returned.

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

#### Parameters

- **game\_short\_name** – Short name of the game, serves as the unique identifier for the game. Set up your games via [@BotFather](#).
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only



- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method [aiogram.methods.send\\_game.SendGame](#)

**answer\_invoice**(title: str, description: str, payload: str, provider\_token: str, currency: str, prices: List[LabeledPrice], message\_thread\_id: int | None = None, max\_tip\_amount: int | None = None, suggested\_tip\_amounts: List[int] | None = None, start\_parameter: str | None = None, provider\_data: str | None = None, photo\_url: str | None = None, photo\_size: int | None = None, photo\_width: int | None = None, photo\_height: int | None = None, need\_name: bool | None = None, need\_phone\_number: bool | None = None, need\_email: bool | None = None, need\_shipping\_address: bool | None = None, send\_phone\_number\_to\_provider: bool | None = None, send\_email\_to\_provider: bool | None = None, is\_flexible: bool | None = None, disable\_notification: bool | None = None, protect\_content: bool | None = sentinel.UNSET\_PROTECT\_CONTENT, reply\_to\_message\_id: int | None = None, allow\_sending\_without\_reply: bool | None = None, reply\_markup: InlineKeyboardMarkup | None = None, \*\*kwargs: Any) → [SendInvoice](#)

Shortcut for method [aiogram.methods.send\\_invoice.SendInvoice](#) will automatically fill method attributes:

- **chat\_id**

Use this method to send invoices. On success, the sent [aiogram.types.message.Message](#) is returned.

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

#### Parameters

- **title** – Product name, 1-32 characters
- **description** – Product description, 1-255 characters
- **payload** – Bot-defined invoice payload, 1-128 bytes. This will not be displayed to the user, use for your internal processes.
- **provider\_token** – Payment provider token, obtained via [@BotFather](#)
- **currency** – Three-letter ISO 4217 currency code, see [more on currencies](#)
- **prices** – Price breakdown, a JSON-serialized list of components (e.g. product price, tax, discount, delivery cost, delivery tax, bonus, etc.)
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **max\_tip\_amount** – The maximum accepted amount for tips in the *smallest units* of the currency (integer, **not** float/double). For example, for a maximum tip of US\$ 1.45 pass `max_tip_amount = 145`. See the *exp* parameter in [currencies.json](#), it shows the number of digits past the decimal point for each currency (2 for the majority of currencies). Defaults to 0



- **suggested\_tip\_amounts** – A JSON-serialized array of suggested amounts of tips in the *smallest units* of the currency (integer, **not** float/double). At most 4 suggested tip amounts can be specified. The suggested tip amounts must be positive, passed in a strictly increased order and must not exceed *max\_tip\_amount*.
- **start\_parameter** – 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** – JSON-serialized data about the invoice, which will be shared with the payment provider. A detailed description of required fields should be provided by the payment provider.
- **photo\_url** – URL of the product photo for the invoice. Can be a photo of the goods or a marketing image for a service. People like it better when they see what they are paying for.
- **photo\_size** – Photo size in bytes
- **photo\_width** – Photo width
- **photo\_height** – Photo height
- **need\_name** – Pass True if you require the user's full name to complete the order
- **need\_phone\_number** – Pass True if you require the user's phone number to complete the order
- **need\_email** – Pass True if you require the user's email address to complete the order
- **need\_shipping\_address** – Pass True if you require the user's shipping address to complete the order
- **send\_phone\_number\_to\_provider** – Pass True if the user's phone number should be sent to provider
- **send\_email\_to\_provider** – Pass True if the user's email address should be sent to provider
- **is\_flexible** – Pass True if the final price depends on the shipping method
- **disable\_notification** – Sends the message *silently*. Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method `aiogram.methods.send_invoice.SendInvoice`

```
answer_invoice_pm(title: str, description: str, payload: str, provider_token: str, currency: str, prices:
    List[LabeledPrice], message_thread_id: int | None = None, max_tip_amount: int |
    None = None, suggested_tip_amounts: List[int] | None = None, start_parameter: str |
    None = None, provider_data: str | None = None, photo_url: str | None = None,
    photo_size: int | None = None, photo_width: int | None = None, photo_height: int |
    None = None, need_name: bool | None = None, need_phone_number: bool | None =
    None, need_email: bool | None = None, need_shipping_address: bool | None = None,
    send_phone_number_to_provider: bool | None = None, send_email_to_provider: bool
    | None = None, is_flexible: bool | None = None, disable_notification: bool | None =
    None, protect_content: bool | None = sentinel.UNSET_PROTECT_CONTENT,
    reply_to_message_id: int | None = None, allow_sending_without_reply: bool | None =
    None, reply_markup: InlineKeyboardMarkup | None = None, **kwargs: Any) →
    SendInvoice
```

Shortcut for method `aiogram.methods.send_invoice.SendInvoice` will automatically fill method attributes:

- `chat_id`

Use this method to send invoices. On success, the sent `aiogram.types.message.Message` is returned.

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

#### Parameters

- **title** – Product name, 1-32 characters
- **description** – Product description, 1-255 characters
- **payload** – Bot-defined invoice payload, 1-128 bytes. This will not be displayed to the user, use for your internal processes.
- **provider\_token** – Payment provider token, obtained via `@BotFather`
- **currency** – Three-letter ISO 4217 currency code, see [more on currencies](#)
- **prices** – Price breakdown, a JSON-serialized list of components (e.g. product price, tax, discount, delivery cost, delivery tax, bonus, etc.)
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **max\_tip\_amount** – The maximum accepted amount for tips in the *smallest units* of the currency (integer, **not** float/double). For example, for a maximum tip of US\$ 1.45 pass `max_tip_amount = 145`. See the `exp` parameter in [currencies.json](#), it shows the number of digits past the decimal point for each currency (2 for the majority of currencies). Defaults to 0
- **suggested\_tip\_amounts** – A JSON-serialized array of suggested amounts of tips in the *smallest units* of the currency (integer, **not** float/double). At most 4 suggested tip amounts can be specified. The suggested tip amounts must be positive, passed in a strictly increased order and must not exceed `max_tip_amount`.
- **start\_parameter** – 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** – JSON-serialized data about the invoice, which will be shared with the payment provider. A detailed description of required fields should be provided by the payment provider.

- **photo\_url** – URL of the product photo for the invoice. Can be a photo of the goods or a marketing image for a service. People like it better when they see what they are paying for.
- **photo\_size** – Photo size in bytes
- **photo\_width** – Photo width
- **photo\_height** – Photo height
- **need\_name** – Pass True if you require the user's full name to complete the order
- **need\_phone\_number** – Pass True if you require the user's phone number to complete the order
- **need\_email** – Pass True if you require the user's email address to complete the order
- **need\_shipping\_address** – Pass True if you require the user's shipping address to complete the order
- **send\_phone\_number\_to\_provider** – Pass True if the user's phone number should be sent to provider
- **send\_email\_to\_provider** – Pass True if the user's email address should be sent to provider
- **is\_flexible** – Pass True if the final price depends on the shipping method
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method [aiogram.methods.send\\_invoice.SendInvoice](#)

**answer\_location**(*latitude: float, longitude: float, message\_thread\_id: int | None = None, horizontal\_accuracy: float | None = None, live\_period: int | None = None, heading: int | None = None, proximity\_alert\_radius: int | None = None, disable\_notification: bool | None = None, protect\_content: bool | None = sentinel.UNSET\_PROTECT\_CONTENT, reply\_to\_message\_id: int | None = None, allow\_sending\_without\_reply: bool | None = None, reply\_markup: [InlineKeyboardMarkup](#) | [ReplyKeyboardMarkup](#) | [ReplyKeyboardRemove](#) | [ForceReply](#) | None = None, \*\*kwargs: Any*) → [SendLocation](#)

Shortcut for method [aiogram.methods.send\\_location.SendLocation](#) will automatically fill method attributes:

- **chat\_id**

Use this method to send point on the map. On success, the sent [aiogram.types.message.Message](#) is returned.

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

#### Parameters

- **latitude** – Latitude of the location
- **longitude** – Longitude of the location

- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **horizontal\_accuracy** – The radius of uncertainty for the location, measured in meters; 0-1500
- **live\_period** – Period in seconds for which the location will be updated (see [Live Locations](#), should be between 60 and 86400.
- **heading** – For live locations, a direction in which the user is moving, in degrees. Must be between 1 and 360 if specified.
- **proximity\_alert\_radius** – 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** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method [aiogram.methods.send\\_location.SendLocation](#)

**answer\_location\_pm**(latitude: float, longitude: float, message\_thread\_id: int | None = None, horizontal\_accuracy: float | None = None, live\_period: int | None = None, heading: int | None = None, proximity\_alert\_radius: int | None = None, disable\_notification: bool | None = None, protect\_content: bool | None = sentinel.UNSET\_PROTECT\_CONTENT, reply\_to\_message\_id: int | None = None, allow\_sending\_without\_reply: bool | None = None, reply\_markup: [InlineKeyboardMarkup](#) | [ReplyKeyboardMarkup](#) | [ReplyKeyboardRemove](#) | [ForceReply](#) | None = None, \*\*kwargs: Any) → [SendLocation](#)

Shortcut for method [aiogram.methods.send\\_location.SendLocation](#) will automatically fill method attributes:

- **chat\_id**

Use this method to send point on the map. On success, the sent [aiogram.types.message.Message](#) is returned.

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

#### Parameters

- **latitude** – Latitude of the location
- **longitude** – Longitude of the location
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **horizontal\_accuracy** – The radius of uncertainty for the location, measured in meters; 0-1500

- **live\_period** – Period in seconds for which the location will be updated (see [Live Locations](#), should be between 60 and 86400).
- **heading** – For live locations, a direction in which the user is moving, in degrees. Must be between 1 and 360 if specified.
- **proximity\_alert\_radius** – 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** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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**

instance of method `aiogram.methods.send_location.SendLocation`

**answer\_media\_group**(*media*: List[[InputMediaAudio](#) | [InputMediaDocument](#) | [InputMediaPhoto](#) | [InputMediaVideo](#)], *message\_thread\_id*: int | None = None, *disable\_notification*: bool | None = None, *protect\_content*: bool | None = `sentinel.UNSET_PROTECT_CONTENT`, *reply\_to\_message\_id*: int | None = None, *allow\_sending\_without\_reply*: bool | None = None, *\*\*kwargs*: Any) → [SendMediaGroup](#)

Shortcut for method `aiogram.methods.send_media_group.SendMediaGroup` will automatically fill method attributes:

- **chat\_id**

Use this method to send a group of photos, videos, documents or audios as an album. Documents and audio files can be only grouped 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** – A JSON-serialized array describing messages to be sent, must include 2-10 items
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **disable\_notification** – Sends messages [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent messages from forwarding and saving
- **reply\_to\_message\_id** – If the messages are a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found

**Returns**

instance of method `aiogram.methods.send_media_group.SendMediaGroup`

```
answer_media_group_pm(media: List[InputMediaAudio | InputMediaDocument | InputMediaPhoto |  
    InputMediaVideo], message_thread_id: int | None = None, disable_notification:  
    bool | None = None, protect_content: bool | None =  
    sentinel.UNSET_PROTECT_CONTENT, reply_to_message_id: int | None =  
    None, allow_sending_without_reply: bool | None = None, **kwargs: Any) →  
    SendMediaGroup
```

Shortcut for method `aiogram.methods.send_media_group.SendMediaGroup` will automatically fill method attributes:

- `chat_id`

Use this method to send a group of photos, videos, documents or audios as an album. Documents and audio files can be only grouped 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** – A JSON-serialized array describing messages to be sent, must include 2-10 items
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **disable\_notification** – Sends messages `silently`. Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent messages from forwarding and saving
- **reply\_to\_message\_id** – If the messages are a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found

#### Returns

instance of method `aiogram.methods.send_media_group.SendMediaGroup`

```
answer_photo(photo: InputFile | str, message_thread_id: int | None = None, caption: str | None = None,  
    parse_mode: str | None = sentinel.UNSET_PARSE_MODE, caption_entities:  
    List[MessageEntity] | None = None, has_spoiler: bool | None = None, disable_notification:  
    bool | None = None, protect_content: bool | None = sentinel.UNSET_PROTECT_CONTENT,  
    reply_to_message_id: int | None = None, allow_sending_without_reply: bool | None = None,  
    reply_markup: InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove |  
    ForceReply | None = None, **kwargs: Any) → SendPhoto
```

Shortcut for method `aiogram.methods.send_photo.SendPhoto` will automatically fill method attributes:

- `chat_id`

Use this method to send photos. On success, the sent `aiogram.types.message.Message` is returned.

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

#### Parameters

- **photo** – Photo to send. Pass a file\_id as String to send a photo that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get a photo from the Internet, or upload a new photo using multipart/form-data. The photo must be at most 10 MB in size. The photo's width and height must not exceed 10000 in total. Width and height ratio must be at most 20. *More information on Sending Files »*

- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **caption** – Photo caption (may also be used when resending photos by *file\_id*), 0-1024 characters after entities parsing
- **parse\_mode** – Mode for parsing entities in the photo caption. See [formatting options](#) for more details.
- **caption\_entities** – A JSON-serialized list of special entities that appear in the caption, which can be specified instead of *parse\_mode*
- **has\_spoiler** – Pass True if the photo needs to be covered with a spoiler animation
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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**

instance of method [aiogram.methods.send\\_photo.SendPhoto](#)

**answer\_photo\_pm**(*photo*: [InputFile](#) | *str*, *message\_thread\_id*: *int* | *None* = *None*, *caption*: *str* | *None* = *None*, *parse\_mode*: *str* | *None* = *sentinel.UNSET\_PARSE\_MODE*, *caption\_entities*: *List*[[MessageEntity](#)] | *None* = *None*, *has\_spoiler*: *bool* | *None* = *None*, *disable\_notification*: *bool* | *None* = *None*, *protect\_content*: *bool* | *None* = *sentinel.UNSET\_PROTECT\_CONTENT*, *reply\_to\_message\_id*: *int* | *None* = *None*, *allow\_sending\_without\_reply*: *bool* | *None* = *None*, *reply\_markup*: [InlineKeyboardMarkup](#) | [ReplyKeyboardMarkup](#) | [ReplyKeyboardRemove](#) | [ForceReply](#) | *None* = *None*, *\*\*kwargs*: *Any*) → [SendPhoto](#)

Shortcut for method [aiogram.methods.send\\_photo.SendPhoto](#) will automatically fill method attributes:

- **chat\_id**

Use this method to send photos. On success, the sent [aiogram.types.message.Message](#) is returned.

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

**Parameters**

- **photo** – Photo to send. Pass a *file\_id* as *String* to send a photo that exists on the Telegram servers (recommended), pass an HTTP URL as a *String* for Telegram to get a photo from the Internet, or upload a new photo using multipart/form-data. The photo must be at most 10 MB in size. The photo's width and height must not exceed 10000 in total. Width and height ratio must be at most 20. [More information on Sending Files](#) »
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **caption** – Photo caption (may also be used when resending photos by *file\_id*), 0-1024 characters after entities parsing



- **parse\_mode** – Mode for parsing entities in the photo caption. See [formatting options](#) for more details.
- **caption\_entities** – A JSON-serialized list of special entities that appear in the caption, which can be specified instead of *parse\_mode*
- **has\_spoiler** – Pass True if the photo needs to be covered with a spoiler animation
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method [aiogram.methods.send\\_photo.SendPhoto](#)

**answer\_poll**(*question: str, options: List[str], message\_thread\_id: int | None = None, is\_anonymous: bool | None = None, type: str | None = None, allows\_multiple\_answers: bool | None = None, correct\_option\_id: int | None = None, explanation: str | None = None, explanation\_parse\_mode: str | None = sentinel.UNSET\_PARSE\_MODE, explanation\_entities: List[MessageEntity] | None = None, open\_period: int | None = None, close\_date: datetime.datetime | datetime.timedelta | int | None = None, is\_closed: bool | None = None, disable\_notification: bool | None = None, protect\_content: bool | None = sentinel.UNSET\_PROTECT\_CONTENT, reply\_to\_message\_id: int | None = None, allow\_sending\_without\_reply: bool | None = None, reply\_markup: InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply | None = None, \*\*kwargs: Any*) → *SendPoll*

Shortcut for method [aiogram.methods.send\\_poll.SendPoll](#) will automatically fill method attributes:

- **chat\_id**

Use this method to send a native poll. On success, the sent [aiogram.types.message.Message](#) is returned.

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

#### Parameters

- **question** – Poll question, 1-300 characters
- **options** – A JSON-serialized list of answer options, 2-10 strings 1-100 characters each
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **is\_anonymous** – True, if the poll needs to be anonymous, defaults to True
- **type** – Poll type, ‘quiz’ or ‘regular’, defaults to ‘regular’
- **allows\_multiple\_answers** – True, if the poll allows multiple answers, ignored for polls in quiz mode, defaults to False
- **correct\_option\_id** – 0-based identifier of the correct answer option, required for polls in quiz mode



- **explanation** – 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** – Mode for parsing entities in the explanation. See [formatting options](#) for more details.
- **explanation\_entities** – A JSON-serialized list of special entities that appear in the poll explanation, which can be specified instead of *parse\_mode*
- **open\_period** – Amount of time in seconds the poll will be active after creation, 5-600. Can't be used together with *close\_date*.
- **close\_date** – 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** – Pass True if the poll needs to be immediately closed. This can be useful for poll preview.
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method [aiogram.methods.send\\_poll.SendPoll](#)

```
answer_poll_pm(question: str, options: List[str], message_thread_id: int | None = None, is_anonymous: bool | None = None, type: str | None = None, allows_multiple_answers: bool | None = None, correct_option_id: int | None = None, explanation: str | None = None, explanation_parse_mode: str | None = sentinel.UNSET_PARSE_MODE, explanation_entities: List[MessageEntity] | None = None, open_period: int | None = None, close_date: datetime.datetime | datetime.timedelta | int | None = None, is_closed: bool | None = None, disable_notification: bool | None = None, protect_content: bool | None = sentinel.UNSET_PROTECT_CONTENT, reply_to_message_id: int | None = None, allow_sending_without_reply: bool | None = None, reply_markup: InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply | None = None, **kwargs: Any) → SendPoll
```

Shortcut for method [aiogram.methods.send\\_poll.SendPoll](#) will automatically fill method attributes:

- **chat\_id**

Use this method to send a native poll. On success, the sent [aiogram.types.message.Message](#) is returned.

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

#### Parameters

- **question** – Poll question, 1-300 characters
- **options** – A JSON-serialized list of answer options, 2-10 strings 1-100 characters each

- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **is\_anonymous** – True, if the poll needs to be anonymous, defaults to True
- **type** – Poll type, ‘quiz’ or ‘regular’, defaults to ‘regular’
- **allows\_multiple\_answers** – True, if the poll allows multiple answers, ignored for polls in quiz mode, defaults to False
- **correct\_option\_id** – 0-based identifier of the correct answer option, required for polls in quiz mode
- **explanation** – 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** – Mode for parsing entities in the explanation. See [formatting options](#) for more details.
- **explanation\_entities** – A JSON-serialized list of special entities that appear in the poll explanation, which can be specified instead of *parse\_mode*
- **open\_period** – Amount of time in seconds the poll will be active after creation, 5-600. Can’t be used together with *close\_date*.
- **close\_date** – 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** – Pass True if the poll needs to be immediately closed. This can be useful for poll preview.
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method [aiogram.methods.send\\_poll.SendPoll](#)

**answer\_dice**(*message\_thread\_id*: int | None = None, *emoji*: str | None = None, *disable\_notification*: bool | None = None, *protect\_content*: bool | None = sentinel.UNSET\_PROTECT\_CONTENT, *reply\_to\_message\_id*: int | None = None, *allow\_sending\_without\_reply*: bool | None = None, *reply\_markup*: [InlineKeyboardMarkup](#) | [ReplyKeyboardMarkup](#) | [ReplyKeyboardRemove](#) | [ForceReply](#) | None = None, *\*\*kwargs*: Any) → [SendDice](#)

Shortcut for method [aiogram.methods.send\\_dice.SendDice](#) will automatically fill method attributes:

- **chat\_id**

Use this method to send an animated emoji that will display a random value. On success, the sent [aiogram.types.message.Message](#) is returned.

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

### Parameters

- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **emoji** – 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** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method [aiogram.methods.send\\_dice.SendDice](#)

```
answer_dice_pm(message_thread_id: int | None = None, emoji: str | None = None, disable_notification:
    bool | None = None, protect_content: bool | None =
    sentinel.UNSET_PROTECT_CONTENT, reply_to_message_id: int | None = None,
    allow_sending_without_reply: bool | None = None, reply_markup: InlineKeyboardMarkup
    | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply | None = None, **kwargs:
    Any) → SendDice
```

Shortcut for method [aiogram.methods.send\\_dice.SendDice](#) will automatically fill method attributes:

- **chat\_id**

Use this method to send an animated emoji that will display a random value. On success, the sent [aiogram.types.message.Message](#) is returned.

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

### Parameters

- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **emoji** – 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** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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**

instance of method `aiogram.methods.send_dice.SendDice`

```
answer_sticker(sticker: InputFile | str, message_thread_id: int | None = None, emoji: str | None = None,
                disable_notification: bool | None = None, protect_content: bool | None =
                sentinel.UNSET_PROTECT_CONTENT, reply_to_message_id: int | None = None,
                allow_sending_without_reply: bool | None = None, reply_markup: InlineKeyboardMarkup
                | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply | None = None, **kwargs:
                Any) → SendSticker
```

Shortcut for method `aiogram.methods.send_sticker.SendSticker` will automatically fill method attributes:

- `chat_id`

Use this method to send static .WEBP, `animated` .TGS, or `video` .WEBM stickers. On success, the sent `aiogram.types.message.Message` is returned.

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

**Parameters**

- **sticker** – Sticker to send. Pass a file\_id as String to send a file that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get a .WEBP sticker from the Internet, or upload a new .WEBP or .TGS sticker using multipart/form-data. *More information on Sending Files »*. Video stickers can only be sent by a file\_id. Animated stickers can't be sent via an HTTP URL.
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **emoji** – Emoji associated with the sticker; only for just uploaded stickers
- **disable\_notification** – Sends the message `silently`. Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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**

instance of method `aiogram.methods.send_sticker.SendSticker`

```
answer_sticker_pm(sticker: InputFile | str, message_thread_id: int | None = None, emoji: str | None =
None, disable_notification: bool | None = None, protect_content: bool | None =
sentinel.UNSET_PROTECT_CONTENT, reply_to_message_id: int | None = None,
allow_sending_without_reply: bool | None = None, reply_markup:
InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove |
ForceReply | None = None, **kwargs: Any) → SendSticker
```

Shortcut for method `aiogram.methods.send_sticker.SendSticker` will automatically fill method attributes:

- `chat_id`

Use this method to send static .WEBP, `animated` .TGS, or `video` .WEBM stickers. On success, the sent `aiogram.types.message.Message` is returned.

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

### Parameters

- **sticker** – Sticker to send. Pass a `file_id` as String to send a file that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get a .WEBP sticker from the Internet, or upload a new .WEBP or .TGS sticker using multipart/form-data. *More information on Sending Files* ». Video stickers can only be sent by a `file_id`. Animated stickers can't be sent via an HTTP URL.
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **emoji** – Emoji associated with the sticker; only for just uploaded stickers
- **disable\_notification** – Sends the message *silently*. Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method `aiogram.methods.send_sticker.SendSticker`

**answer\_venue**(latitude: float, longitude: float, title: str, address: str, message\_thread\_id: int | None = None, foursquare\_id: str | None = None, foursquare\_type: str | None = None, google\_place\_id: str | None = None, google\_place\_type: str | None = None, disable\_notification: bool | None = None, protect\_content: bool | None = sentinel.UNSET\_PROTECT\_CONTENT, reply\_to\_message\_id: int | None = None, allow\_sending\_without\_reply: bool | None = None, reply\_markup: InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply | None = None, \*\*kwargs: Any) → *SendVenue*

Shortcut for method `aiogram.methods.send_venue.SendVenue` will automatically fill method attributes:

- `chat_id`

Use this method to send information about a venue. On success, the sent `aiogram.types.message.Message` is returned.

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

### Parameters

- **latitude** – Latitude of the venue
- **longitude** – Longitude of the venue
- **title** – Name of the venue
- **address** – Address of the venue
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **foursquare\_id** – Foursquare identifier of the venue

- **foursquare\_type** – Foursquare type of the venue, if known. (For example, 'arts\_entertainment/default', 'arts\_entertainment/aquarium' or 'food/icecream'.)
- **google\_place\_id** – Google Places identifier of the venue
- **google\_place\_type** – Google Places type of the venue. (See [supported types](#).)
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method [aiogram.methods.send\\_venue.SendVenue](#)

```
answer_venue_pm(latitude: float, longitude: float, title: str, address: str, message_thread_id: int | None =  
    None, foursquare_id: str | None = None, foursquare_type: str | None = None,  
    google_place_id: str | None = None, google_place_type: str | None = None,  
    disable_notification: bool | None = None, protect_content: bool | None =  
    sentinel.UNSET_PROTECT_CONTENT, reply_to_message_id: int | None = None,  
    allow_sending_without_reply: bool | None = None, reply_markup:  
    InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply |  
    None = None, **kwargs: Any) → SendVenue
```

Shortcut for method [aiogram.methods.send\\_venue.SendVenue](#) will automatically fill method attributes:

- **chat\_id**

Use this method to send information about a venue. On success, the sent [aiogram.types.message.Message](#) is returned.

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

#### Parameters

- **latitude** – Latitude of the venue
- **longitude** – Longitude of the venue
- **title** – Name of the venue
- **address** – Address of the venue
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **foursquare\_id** – Foursquare identifier of the venue
- **foursquare\_type** – Foursquare type of the venue, if known. (For example, 'arts\_entertainment/default', 'arts\_entertainment/aquarium' or 'food/icecream'.)
- **google\_place\_id** – Google Places identifier of the venue
- **google\_place\_type** – Google Places type of the venue. (See [supported types](#).)

- **disable\_notification** – Sends the message *silently*. Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method `aiogram.methods.send_venue.SendVenue`

**answer\_video**(*video*: `InputFile` | *str*, *message\_thread\_id*: *int* | *None* = *None*, *duration*: *int* | *None* = *None*, *width*: *int* | *None* = *None*, *height*: *int* | *None* = *None*, *thumbnail*: `InputFile` | *str* | *None* = *None*, *caption*: *str* | *None* = *None*, *parse\_mode*: *str* | *None* = *sentinel.UNSET\_PARSE\_MODE*, *caption\_entities*: *List*[`MessageEntity`] | *None* = *None*, *has\_spoiler*: *bool* | *None* = *None*, *supports\_streaming*: *bool* | *None* = *None*, *disable\_notification*: *bool* | *None* = *None*, *protect\_content*: *bool* | *None* = *sentinel.UNSET\_PROTECT\_CONTENT*, *reply\_to\_message\_id*: *int* | *None* = *None*, *allow\_sending\_without\_reply*: *bool* | *None* = *None*, *reply\_markup*: `InlineKeyboardMarkup` | `ReplyKeyboardMarkup` | `ReplyKeyboardRemove` | `ForceReply` | *None* = *None*, *\*\*kwargs*: *Any*) → *SendVideo*

Shortcut for method `aiogram.methods.send_video.SendVideo` will automatically fill method attributes:

- **chat\_id**

Use this method to send video files, Telegram clients support MPEG4 videos (other formats may be sent as `aiogram.types.document.Document`). On success, the sent `aiogram.types.message.Message` is returned. Bots can currently send video files of up to 50 MB in size, this limit may be changed in the future.

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

#### Parameters

- **video** – Video to send. Pass a *file\_id* as *String* to send a video that exists on the Telegram servers (recommended), pass an HTTP URL as a *String* for Telegram to get a video from the Internet, or upload a new video using multipart/form-data. *More information on Sending Files* »
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **duration** – Duration of sent video in seconds
- **width** – Video width
- **height** – Video height
- **thumbnail** – Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can't be reused and can be only uploaded as a new file, so you can pass 'attach://<file\_attach\_name>' if the thumbnail was uploaded using multipart/form-data under <file\_attach\_name>. *More information on Sending Files* »



- **caption** – Video caption (may also be used when resending videos by *file\_id*), 0-1024 characters after entities parsing
- **parse\_mode** – Mode for parsing entities in the video caption. See [formatting options](#) for more details.
- **caption\_entities** – A JSON-serialized list of special entities that appear in the caption, which can be specified instead of *parse\_mode*
- **has\_spoiler** – Pass True if the video needs to be covered with a spoiler animation
- **supports\_streaming** – Pass True if the uploaded video is suitable for streaming
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method [aiogram.methods.send\\_video.SendVideo](#)

**answer\_video\_pm**(video: [InputFile](#) | str, message\_thread\_id: int | None = None, duration: int | None = None, width: int | None = None, height: int | None = None, thumbnail: [InputFile](#) | str | None = None, caption: str | None = None, parse\_mode: str | None = [sentinel.UNSET\\_PARSE\\_MODE](#), caption\_entities: List[[MessageEntity](#)] | None = None, has\_spoiler: bool | None = None, supports\_streaming: bool | None = None, disable\_notification: bool | None = None, protect\_content: bool | None = [sentinel.UNSET\\_PROTECT\\_CONTENT](#), reply\_to\_message\_id: int | None = None, allow\_sending\_without\_reply: bool | None = None, reply\_markup: [InlineKeyboardMarkup](#) | [ReplyKeyboardMarkup](#) | [ReplyKeyboardRemove](#) | [ForceReply](#) | None = None, \*\*kwargs: Any) → [SendVideo](#)

Shortcut for method [aiogram.methods.send\\_video.SendVideo](#) will automatically fill method attributes:

- **chat\_id**

Use this method to send video files, Telegram clients support MPEG4 videos (other formats may be sent as [aiogram.types.document.Document](#)). On success, the sent [aiogram.types.message.Message](#) is returned. Bots can currently send video files of up to 50 MB in size, this limit may be changed in the future.

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

#### Parameters

- **video** – Video to send. Pass a *file\_id* as String to send a video that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get a video from the Internet, or upload a new video using multipart/form-data. [More information on Sending Files](#) »
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **duration** – Duration of sent video in seconds



- **width** – Video width
- **height** – Video height
- **thumbnail** – Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can't be reused and can be only uploaded as a new file, so you can pass 'attach://<file\_attach\_name>' if the thumbnail was uploaded using multipart/form-data under <file\_attach\_name>. *More information on Sending Files* »
- **caption** – Video caption (may also be used when resending videos by *file\_id*), 0-1024 characters after entities parsing
- **parse\_mode** – Mode for parsing entities in the video caption. See *formatting options* for more details.
- **caption\_entities** – A JSON-serialized list of special entities that appear in the caption, which can be specified instead of *parse\_mode*
- **has\_spoiler** – Pass True if the video needs to be covered with a spoiler animation
- **supports\_streaming** – Pass True if the uploaded video is suitable for streaming
- **disable\_notification** – Sends the message *silently*. Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method `aiogram.methods.send_video.SendVideo`

**answer\_video\_note**(*video\_note*: `InputFile` | `str`, *message\_thread\_id*: `int` | `None` = `None`, *duration*: `int` | `None` = `None`, *length*: `int` | `None` = `None`, *thumbnail*: `InputFile` | `str` | `None` = `None`, *disable\_notification*: `bool` | `None` = `None`, *protect\_content*: `bool` | `None` = `sentinel.UNSET_PROTECT_CONTENT`, *reply\_to\_message\_id*: `int` | `None` = `None`, *allow\_sending\_without\_reply*: `bool` | `None` = `None`, *reply\_markup*: `InlineKeyboardMarkup` | `ReplyKeyboardMarkup` | `ReplyKeyboardRemove` | `ForceReply` | `None` = `None`, *\*\*kwargs*: `Any`) → `SendVideoNote`

Shortcut for method `aiogram.methods.send_video_note.SendVideoNote` will automatically fill method attributes:

- **chat\_id**

As of v.4.0, Telegram clients support rounded square MPEG4 videos of up to 1 minute long. Use this method to send video messages. On success, the sent `aiogram.types.message.Message` is returned.

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

#### Parameters

- **video\_note** – Video note to send. Pass a *file\_id* as String to send a video note that exists on the Telegram servers (recommended) or upload a new video using multipart/form-data.

*More information on Sending Files* ». Sending video notes by a URL is currently unsupported

- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **duration** – Duration of sent video in seconds
- **length** – Video width and height, i.e. diameter of the video message
- **thumbnail** – Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can't be reused and can be only uploaded as a new file, so you can pass 'attach://<file\_attach\_name>' if the thumbnail was uploaded using multipart/form-data under <file\_attach\_name>. *More information on Sending Files* »
- **disable\_notification** – Sends the message *silently*. Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method `aiogram.methods.send_video_note.SendVideoNote`

```
answer_video_note_pm(video_note: InputFile | str, message_thread_id: int | None = None, duration: int | None = None, length: int | None = None, thumbnail: InputFile | str | None = None, disable_notification: bool | None = None, protect_content: bool | None = sentinel.UNSET_PROTECT_CONTENT, reply_to_message_id: int | None = None, allow_sending_without_reply: bool | None = None, reply_markup: InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply | None = None, **kwargs: Any) → SendVideoNote
```

Shortcut for method `aiogram.methods.send_video_note.SendVideoNote` will automatically fill method attributes:

- **chat\_id**

As of v.4.0, Telegram clients support rounded square MPEG4 videos of up to 1 minute long. Use this method to send video messages. On success, the sent `aiogram.types.message.Message` is returned.

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

#### Parameters

- **video\_note** – Video note to send. Pass a file\_id as String to send a video note that exists on the Telegram servers (recommended) or upload a new video using multipart/form-data. *More information on Sending Files* ». Sending video notes by a URL is currently unsupported
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **duration** – Duration of sent video in seconds

- **length** – Video width and height, i.e. diameter of the video message
- **thumbnail** – Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can't be reused and can be only uploaded as a new file, so you can pass 'attach://<file\_attach\_name>' if the thumbnail was uploaded using multipart/form-data under <file\_attach\_name>. [More information on Sending Files](#) »
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method [aiogram.methods.send\\_video\\_note.SendVideoNote](#)

**answer\_voice**(voice: [InputFile](#) | str, message\_thread\_id: int | None = None, caption: str | None = None, parse\_mode: str | None = sentinel.UNSET\_PARSE\_MODE, caption\_entities: List[[MessageEntity](#)] | None = None, duration: int | None = None, disable\_notification: bool | None = None, protect\_content: bool | None = sentinel.UNSET\_PROTECT\_CONTENT, reply\_to\_message\_id: int | None = None, allow\_sending\_without\_reply: bool | None = None, reply\_markup: [InlineKeyboardMarkup](#) | [ReplyKeyboardMarkup](#) | [ReplyKeyboardRemove](#) | [ForceReply](#) | None = None, \*\*kwargs: Any) → [SendVoice](#)

Shortcut for method [aiogram.methods.send\\_voice.SendVoice](#) will automatically fill method attributes:

- chat\_id

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 [aiogram.types.audio.Audio](#) or [aiogram.types.document.Document](#)). On success, the sent [aiogram.types.message.Message](#) is returned. Bots can currently send voice messages of up to 50 MB in size, this limit may be changed in the future.

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

#### Parameters

- **voice** – Audio file to send. Pass a file\_id as String to send a file that exists on the Telegram servers (recommended), 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 information on Sending Files](#) »
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **caption** – Voice message caption, 0-1024 characters after entities parsing
- **parse\_mode** – Mode for parsing entities in the voice message caption. See [formatting options](#) for more details.

- **caption\_entities** – A JSON-serialized list of special entities that appear in the caption, which can be specified instead of *parse\_mode*
- **duration** – Duration of the voice message in seconds
- **disable\_notification** – Sends the message *silently*. Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method `aiogram.methods.send_voice.SendVoice`

**answer\_voice\_pm**(voice: `InputFile` | *str*, message\_thread\_id: *int* | *None* = *None*, caption: *str* | *None* = *None*, parse\_mode: *str* | *None* = *sentinel.UNSET\_PARSE\_MODE*, caption\_entities: *List*[`MessageEntity`] | *None* = *None*, duration: *int* | *None* = *None*, disable\_notification: *bool* | *None* = *None*, protect\_content: *bool* | *None* = *sentinel.UNSET\_PROTECT\_CONTENT*, reply\_to\_message\_id: *int* | *None* = *None*, allow\_sending\_without\_reply: *bool* | *None* = *None*, reply\_markup: `InlineKeyboardMarkup` | `ReplyKeyboardMarkup` | `ReplyKeyboardRemove` | `ForceReply` | *None* = *None*, *\*\*kwargs*: *Any*) → `SendVoice`

Shortcut for method `aiogram.methods.send_voice.SendVoice` will automatically fill method attributes:

- **chat\_id**

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 `aiogram.types.audio.Audio` or `aiogram.types.document.Document`). On success, the sent `aiogram.types.message.Message` is returned. Bots can currently send voice messages of up to 50 MB in size, this limit may be changed in the future.

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

#### Parameters

- **voice** – Audio file to send. Pass a *file\_id* as *String* to send a file that exists on the Telegram servers (recommended), 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 information on Sending Files »*
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **caption** – Voice message caption, 0-1024 characters after entities parsing
- **parse\_mode** – Mode for parsing entities in the voice message caption. See *formatting options* for more details.
- **caption\_entities** – A JSON-serialized list of special entities that appear in the caption, which can be specified instead of *parse\_mode*
- **duration** – Duration of the voice message in seconds

- **disable\_notification** – Sends the message *silently*. Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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**

instance of method *aiogram.methods.send\_voice.SendVoice*

**ChatLocation**

```
class aiogram.types.chat_location.ChatLocation(*, location: Location, address: str, **extra_data: Any)
```

Represents a location to which a chat is connected.

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

**location:** *Location*

The location to which the supergroup is connected. Can't be a live location.

**address:** *str*

Location address; 1-64 characters, as defined by the chat owner

**ChatMember**

```
class aiogram.types.chat_member.ChatMember(**extra_data: Any)
```

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

- *aiogram.types.chat\_member\_owner.ChatMemberOwner*
- *aiogram.types.chat\_member\_administrator.ChatMemberAdministrator*
- *aiogram.types.chat\_member\_member.ChatMemberMember*
- *aiogram.types.chat\_member\_restricted.ChatMemberRestricted*
- *aiogram.types.chat\_member\_left.ChatMemberLeft*
- *aiogram.types.chat\_member\_banned.ChatMemberBanned*

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

## ChatMemberAdministrator

```
class aiogram.types.chat_member_administrator.ChatMemberAdministrator(*, status: ~typing.Literal[<ChatMemberStatus.ADMINISTRATOR>] = 'administrator') = ChatMemberStatus.ADMINISTRATOR, user: ~aiogram.types.user.User, can_be_edited: bool, is_anonymous: bool, can_manage_chat: bool, can_delete_messages: bool, can_manage_video_chats: bool, can_restrict_members: bool, can_promote_members: bool, can_change_info: bool, can_invite_users: bool, can_post_messages: bool | None = None, can_edit_messages: bool | None = None, can_pin_messages: bool | None = None, can_manage_topics: bool | None = None, custom_title: str | None = None, **extra_data: ~typing.Any)
```

Represents a [chat member](#) that has some additional privileges.

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

**status:** `Literal[ChatMemberStatus.ADMINISTRATOR]`

The member's status in the chat, always 'administrator'

**user:** `User`

Information about the user

**can\_be\_edited:** `bool`

True, if the bot is allowed to edit administrator privileges of that user

**is\_anonymous:** `bool`

True, if the user's presence in the chat is hidden

**can\_manage\_chat:** `bool`

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\_delete\_messages:** `bool`

True, if the administrator can delete messages of other users

**can\_manage\_video\_chats:** `bool`

True, if the administrator can manage video chats

**can\_restrict\_members:** `bool`

True, if the administrator can restrict, ban or unban chat members

**can\_promote\_members:** `bool`

True, if the administrator can add new administrators with a subset of their own privileges or demote administrators that they have promoted, directly or indirectly (promoted by administrators that were appointed by the user)

**can\_change\_info:** `bool`

True, if the user is allowed to change the chat title, photo and other settings

**can\_invite\_users:** `bool`

True, if the user is allowed to invite new users to the chat

**can\_post\_messages:** `bool | None`

*Optional.* True, if the administrator can post in the channel; channels only

**can\_edit\_messages:** `bool | None`

*Optional.* True, if the administrator can edit messages of other users and can pin messages; channels only

**can\_pin\_messages:** `bool | None`

*Optional.* True, if the user is allowed to pin messages; groups and supergroups only

**can\_manage\_topics:** `bool | None`

*Optional.* True, if the user is allowed to create, rename, close, and reopen forum topics; supergroups only

**custom\_title:** `str | None`

*Optional.* Custom title for this user

## ChatMemberBanned

```
class aiogram.types.chat_member_banned.ChatMemberBanned(*, status: ~typing.Literal[<ChatMemberStatus.KICKED:
'kicked'>] = ChatMemberStatus.KICKED,
user: ~aiogram.types.user.User, until_date: ~datetime.datetime[~datetime.datetime],
**extra_data: ~typing.Any)
```

Represents a `chat member` that was banned in the chat and can't return to the chat or view chat messages.

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

**status:** `Literal[ChatMemberStatus.KICKED]`

The member's status in the chat, always 'kicked'

**user:** `User`

Information about the user

**until\_date:** `DateTime`

Date when restrictions will be lifted for this user; unix time. If 0, then the user is banned forever

### ChatMemberLeft

```
class aiogram.types.chat_member_left.ChatMemberLeft(*, status:
    ~typing.Literal[<ChatMemberStatus.LEFT:
    'left'>] = ChatMemberStatus.LEFT, user:
    ~aiogram.types.user.User, **extra_data:
    ~typing.Any)
```

Represents a `chat member` that isn't currently a member of the chat, but may join it themselves.

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

**status:** `Literal[ChatMemberStatus.LEFT]`

The member's status in the chat, always 'left'

**user:** `User`

Information about the user

### ChatMemberMember

```
class aiogram.types.chat_member_member.ChatMemberMember(*, status: ~typ-
    ing.Literal[<ChatMemberStatus.MEMBER:
    'member'>] =
    ChatMemberStatus.MEMBER, user:
    ~aiogram.types.user.User, **extra_data:
    ~typing.Any)
```

Represents a `chat member` that has no additional privileges or restrictions.

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

**status:** `Literal[ChatMemberStatus.MEMBER]`

The member's status in the chat, always 'member'

**user:** `User`

Information about the user

### ChatMemberOwner

```
class aiogram.types.chat_member_owner.ChatMemberOwner(*, status: ~typ-
    ing.Literal[<ChatMemberStatus.CREATOR:
    'creator'>] = ChatMemberStatus.CREATOR,
    user: ~aiogram.types.user.User,
    is_anonymous: bool, custom_title: str | None
    = None, **extra_data: ~typing.Any)
```

Represents a `chat member` that owns the chat and has all administrator privileges.

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

**status:** `Literal[ChatMemberStatus.CREATOR]`

The member's status in the chat, always 'creator'

**user:** `User`

Information about the user



**is\_anonymous:** `bool`

True, if the user's presence in the chat is hidden

**custom\_title:** `str | None`

*Optional.* Custom title for this user

## ChatMemberRestricted

```
class aiogram.types.chat_member_restricted.ChatMemberRestricted(*, status: ~typing.Literal[<ChatMemberStatus.RESTRICTED: 'restricted'>] = ChatMemberStatus.RESTRICTED, user: ~aiogram.types.user.User, is_member: bool, can_send_messages: bool, can_send_audios: bool, can_send_documents: bool, can_send_photos: bool, can_send_videos: bool, can_send_video_notes: bool, can_send_voice_notes: bool, can_send_polls: bool, can_send_other_messages: bool, can_add_web_page_previews: bool, can_change_info: bool, can_invite_users: bool, can_pin_messages: bool, can_manage_topics: bool, until_date: ~datetime.datetime[~datetime.datetime], **extra_data: ~typing.Any)
```

Represents a `chat member` that is under certain restrictions in the chat. Supergroups only.

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

**status:** `Literal[ChatMemberStatus.RESTRICTED]`

The member's status in the chat, always 'restricted'

**user:** `User`

Information about the user

**is\_member:** `bool`

True, if the user is a member of the chat at the moment of the request

**can\_send\_messages:** `bool`

True, if the user is allowed to send text messages, contacts, invoices, locations and venues

**can\_send\_audios:** `bool`

True, if the user is allowed to send audios

**can\_send\_documents:** `bool`

True, if the user is allowed to send documents

**can\_send\_photos:** `bool`

True, if the user is allowed to send photos

**can\_send\_videos:** `bool`

True, if the user is allowed to send videos

**can\_send\_video\_notes:** `bool`

True, if the user is allowed to send video notes

**can\_send\_voice\_notes:** `bool`

True, if the user is allowed to send voice notes

**can\_send\_polls:** `bool`

True, if the user is allowed to send polls

**can\_send\_other\_messages:** `bool`

True, if the user is allowed to send animations, games, stickers and use inline bots

**can\_add\_web\_page\_previews:** `bool`

True, if the user is allowed to add web page previews to their messages

**can\_change\_info:** `bool`

True, if the user is allowed to change the chat title, photo and other settings

**can\_invite\_users:** `bool`

True, if the user is allowed to invite new users to the chat

**can\_pin\_messages:** `bool`

True, if the user is allowed to pin messages

**can\_manage\_topics:** `bool`

True, if the user is allowed to create forum topics

**until\_date:** `DateTime`

Date when restrictions will be lifted for this user; unix time. If 0, then the user is restricted forever

## ChatMemberUpdated

```
class aiogram.types.chat_member_updated.ChatMemberUpdated(*, chat: Chat, from_user: User, date:
    datetime[datetime], old_chat_member:
        ChatMemberOwner |
        ChatMemberAdministrator |
        ChatMemberMember |
        ChatMemberRestricted |
        ChatMemberLeft | ChatMemberBanned,
    new_chat_member: ChatMemberOwner
    | ChatMemberAdministrator |
    ChatMemberMember |
    ChatMemberRestricted |
    ChatMemberLeft | ChatMemberBanned,
    invite_link: ChatInviteLink | None =
    None, via_chat_folder_invite_link: bool |
    None = None, **extra_data: Any)
```

This object represents changes in the status of a chat member.

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

**chat:** `Chat`

Chat the user belongs to

**from\_user:** `User`

Performer of the action, which resulted in the change

**date:** `DateTime`

Date the change was done in Unix time

**old\_chat\_member:** `ChatMemberOwner` | `ChatMemberAdministrator` | `ChatMemberMember` | `ChatMemberRestricted` | `ChatMemberLeft` | `ChatMemberBanned`

Previous information about the chat member

**new\_chat\_member:** `ChatMemberOwner` | `ChatMemberAdministrator` | `ChatMemberMember` | `ChatMemberRestricted` | `ChatMemberLeft` | `ChatMemberBanned`

New information about the chat member

**invite\_link:** `ChatInviteLink` | `None`

*Optional.* Chat invite link, which was used by the user to join the chat; for joining by invite link events only.

**via\_chat\_folder\_invite\_link:** `bool` | `None`

*Optional.* True, if the user joined the chat via a chat folder invite link

**answer**(*text*: str, *message\_thread\_id*: int | None = None, *parse\_mode*: str | None = sentinel.UNSET\_PARSE\_MODE, *entities*: List[`MessageEntity`] | None = None, *disable\_web\_page\_preview*: bool | None = sentinel.UNSET\_DISABLE\_WEB\_PAGE\_PREVIEW, *disable\_notification*: bool | None = None, *protect\_content*: bool | None = sentinel.UNSET\_PROTECT\_CONTENT, *reply\_to\_message\_id*: int | None = None, *allow\_sending\_without\_reply*: bool | None = None, *reply\_markup*: `InlineKeyboardMarkup` | `ReplyKeyboardMarkup` | `ReplyKeyboardRemove` | `ForceReply` | None = None, *\*\*kwargs*: Any) → `SendMessage`

Shortcut for method `aiogram.methods.send_message.SendMessage` will automatically fill method attributes:

- `chat_id`

Use this method to send text messages. On success, the sent `aiogram.types.message.Message` is returned.

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

#### Parameters

- **text** – Text of the message to be sent, 1-4096 characters after entities parsing
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **parse\_mode** – Mode for parsing entities in the message text. See [formatting options](#) for more details.
- **entities** – A JSON-serialized list of special entities that appear in message text, which can be specified instead of *parse\_mode*
- **disable\_web\_page\_preview** – Disables link previews for links in this message
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message

- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method [aiogram.methods.send\\_message.SendMessage](#)

**answer\_animation**(*animation*: [InputFile](#) | *str*, *message\_thread\_id*: *int* | *None* = *None*, *duration*: *int* | *None* = *None*, *width*: *int* | *None* = *None*, *height*: *int* | *None* = *None*, *thumbnail*: [InputFile](#) | *str* | *None* = *None*, *caption*: *str* | *None* = *None*, *parse\_mode*: *str* | *None* = *sentinel.UNSET\_PARSE\_MODE*, *caption\_entities*: *List*[[MessageEntity](#)] | *None* = *None*, *has\_spoiler*: *bool* | *None* = *None*, *disable\_notification*: *bool* | *None* = *None*, *protect\_content*: *bool* | *None* = *sentinel.UNSET\_PROTECT\_CONTENT*, *reply\_to\_message\_id*: *int* | *None* = *None*, *allow\_sending\_without\_reply*: *bool* | *None* = *None*, *reply\_markup*: [InlineKeyboardMarkup](#) | [ReplyKeyboardMarkup](#) | [ReplyKeyboardRemove](#) | [ForceReply](#) | *None* = *None*, *\*\*kwargs*: *Any*) → [SendAnimation](#)

Shortcut for method [aiogram.methods.send\\_animation.SendAnimation](#) will automatically fill method attributes:

- **chat\_id**

Use this method to send animation files (GIF or H.264/MPEG-4 AVC video without sound). On success, the sent [aiogram.types.message.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** – 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. [More information on Sending Files](#) »
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **duration** – Duration of sent animation in seconds
- **width** – Animation width
- **height** – Animation height
- **thumbnail** – Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can't be reused and can be only uploaded as a new file, so you can pass 'attach://<file\_attach\_name>' if the thumbnail was uploaded using multipart/form-data under <file\_attach\_name>. [More information on Sending Files](#) »
- **caption** – Animation caption (may also be used when resending animation by *file\_id*), 0-1024 characters after entities parsing
- **parse\_mode** – Mode for parsing entities in the animation caption. See [formatting options](#) for more details.
- **caption\_entities** – A JSON-serialized list of special entities that appear in the caption, which can be specified instead of *parse\_mode*

- **has\_spoiler** – Pass True if the animation needs to be covered with a spoiler animation
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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**

instance of method [aiogram.methods.send\\_animation.SendAnimation](#)

**answer\_audio**(*audio*: [InputFile](#) | *str*, *message\_thread\_id*: *int* | *None* = *None*, *caption*: *str* | *None* = *None*, *parse\_mode*: *str* | *None* = *sentinel.UNSET\_PARSE\_MODE*, *caption\_entities*: *List*[[MessageEntity](#)] | *None* = *None*, *duration*: *int* | *None* = *None*, *performer*: *str* | *None* = *None*, *title*: *str* | *None* = *None*, *thumbnail*: [InputFile](#) | *str* | *None* = *None*, *disable\_notification*: *bool* | *None* = *None*, *protect\_content*: *bool* | *None* = *sentinel.UNSET\_PROTECT\_CONTENT*, *reply\_to\_message\_id*: *int* | *None* = *None*, *allow\_sending\_without\_reply*: *bool* | *None* = *None*, *reply\_markup*: [InlineKeyboardMarkup](#) | [ReplyKeyboardMarkup](#) | [ReplyKeyboardRemove](#) | [ForceReply](#) | *None* = *None*, *\*\*kwargs*: *Any*) → [SendAudio](#)

Shortcut for method [aiogram.methods.send\\_audio.SendAudio](#) will automatically fill method attributes:

- **chat\_id**

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 or .M4A format. On success, the sent [aiogram.types.message.Message](#) is returned. Bots can currently send audio files of up to 50 MB in size, this limit may be changed in the future. For sending voice messages, use the [aiogram.methods.send\\_voice.SendVoice](#) method instead.

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

**Parameters**

- **audio** – Audio file to send. Pass a `file_id` as String to send an audio file that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get an audio file from the Internet, or upload a new one using multipart/form-data. [More information on Sending Files](#) »
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **caption** – Audio caption, 0-1024 characters after entities parsing
- **parse\_mode** – Mode for parsing entities in the audio caption. See [formatting options](#) for more details.
- **caption\_entities** – A JSON-serialized list of special entities that appear in the caption, which can be specified instead of *parse\_mode*
- **duration** – Duration of the audio in seconds
- **performer** – Performer
- **title** – Track name

- **thumbnail** – Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can't be reused and can be only uploaded as a new file, so you can pass 'attach://<file\_attach\_name>' if the thumbnail was uploaded using multipart/form-data under <file\_attach\_name>. *More information on Sending Files* »
- **disable\_notification** – Sends the message *silently*. Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method *aiogram.methods.send\_audio.SendAudio*

**answer\_contact**(*phone\_number: str, first\_name: str, message\_thread\_id: int | None = None, last\_name: str | None = None, vcard: str | None = None, disable\_notification: bool | None = None, protect\_content: bool | None = sentinel.UNSET\_PROTECT\_CONTENT, reply\_to\_message\_id: int | None = None, allow\_sending\_without\_reply: bool | None = None, reply\_markup: InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply | None = None, \*\*kwargs: Any*) → *SendContact*

Shortcut for method *aiogram.methods.send\_contact.SendContact* will automatically fill method attributes:

- **chat\_id**

Use this method to send phone contacts. On success, the sent *aiogram.types.message.Message* is returned.

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

#### Parameters

- **phone\_number** – Contact's phone number
- **first\_name** – Contact's first name
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **last\_name** – Contact's last name
- **vcard** – Additional data about the contact in the form of a *vCard*, 0-2048 bytes
- **disable\_notification** – Sends the message *silently*. Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found

- **reply\_markup** – 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

instance of method `aiogram.methods.send_contact.SendContact`

**answer\_document**(*document*: [InputFile](#) | *str*, *message\_thread\_id*: *int* | *None* = *None*, *thumbnail*: [InputFile](#) | *str* | *None* = *None*, *caption*: *str* | *None* = *None*, *parse\_mode*: *str* | *None* = *sentinel.UNSET\_PARSE\_MODE*, *caption\_entities*: *List*[[MessageEntity](#)] | *None* = *None*, *disable\_content\_type\_detection*: *bool* | *None* = *None*, *disable\_notification*: *bool* | *None* = *None*, *protect\_content*: *bool* | *None* = *sentinel.UNSET\_PROTECT\_CONTENT*, *reply\_to\_message\_id*: *int* | *None* = *None*, *allow\_sending\_without\_reply*: *bool* | *None* = *None*, *reply\_markup*: [InlineKeyboardMarkup](#) | [ReplyKeyboardMarkup](#) | [ReplyKeyboardRemove](#) | [ForceReply](#) | *None* = *None*, *\*\*kwargs*: *Any*) → *SendDocument*

Shortcut for method `aiogram.methods.send_document.SendDocument` will automatically fill method attributes:

- `chat_id`

Use this method to send general files. On success, the sent `aiogram.types.message.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** – File to send. Pass a `file_id` as `String` to send a file that exists on the Telegram servers (recommended), 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 information on Sending Files](#) »
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **thumbnail** – Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can't be reused and can be only uploaded as a new file, so you can pass 'attach://<file\_attach\_name>' if the thumbnail was uploaded using multipart/form-data under <file\_attach\_name>. [More information on Sending Files](#) »
- **caption** – Document caption (may also be used when resending documents by `file_id`), 0-1024 characters after entities parsing
- **parse\_mode** – Mode for parsing entities in the document caption. See [formatting options](#) for more details.
- **caption\_entities** – A JSON-serialized list of special entities that appear in the caption, which can be specified instead of `parse_mode`
- **disable\_content\_type\_detection** – Disables automatic server-side content type detection for files uploaded using multipart/form-data
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message



- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method [aiogram.methods.send\\_document.SendDocument](#)

**answer\_game**(*game\_short\_name: str, message\_thread\_id: int | None = None, disable\_notification: bool | None = None, protect\_content: bool | None = sentinel.UNSET\_PROTECT\_CONTENT, reply\_to\_message\_id: int | None = None, allow\_sending\_without\_reply: bool | None = None, reply\_markup: InlineKeyboardMarkup | None = None, \*\*kwargs: Any*) → [SendGame](#)

Shortcut for method [aiogram.methods.send\\_game.SendGame](#) will automatically fill method attributes:

- **chat\_id**

Use this method to send a game. On success, the sent [aiogram.types.message.Message](#) is returned.

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

#### Parameters

- **game\_short\_name** – Short name of the game, serves as the unique identifier for the game. Set up your games via [@BotFather](#).
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method [aiogram.methods.send\\_game.SendGame](#)

**answer\_invoice**(*title: str, description: str, payload: str, provider\_token: str, currency: str, prices: List[LabeledPrice], message\_thread\_id: int | None = None, max\_tip\_amount: int | None = None, suggested\_tip\_amounts: List[int] | None = None, start\_parameter: str | None = None, provider\_data: str | None = None, photo\_url: str | None = None, photo\_size: int | None = None, photo\_width: int | None = None, photo\_height: int | None = None, need\_name: bool | None = None, need\_phone\_number: bool | None = None, need\_email: bool | None = None, need\_shipping\_address: bool | None = None, send\_phone\_number\_to\_provider: bool | None = None, send\_email\_to\_provider: bool | None = None, is\_flexible: bool | None = None, disable\_notification: bool | None = None, protect\_content: bool | None = sentinel.UNSET\_PROTECT\_CONTENT, reply\_to\_message\_id: int | None = None, allow\_sending\_without\_reply: bool | None = None, reply\_markup: InlineKeyboardMarkup | None = None, \*\*kwargs: Any*) → [SendInvoice](#)

Shortcut for method [aiogram.methods.send\\_invoice.SendInvoice](#) will automatically fill method attributes:



- `chat_id`

Use this method to send invoices. On success, the sent `aiogram.types.message.Message` is returned.

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

#### Parameters

- **title** – Product name, 1-32 characters
- **description** – Product description, 1-255 characters
- **payload** – Bot-defined invoice payload, 1-128 bytes. This will not be displayed to the user, use for your internal processes.
- **provider\_token** – Payment provider token, obtained via `@BotFather`
- **currency** – Three-letter ISO 4217 currency code, see [more on currencies](#)
- **prices** – Price breakdown, a JSON-serialized list of components (e.g. product price, tax, discount, delivery cost, delivery tax, bonus, etc.)
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **max\_tip\_amount** – The maximum accepted amount for tips in the *smallest units* of the currency (integer, **not** float/double). For example, for a maximum tip of US\$ 1.45 pass `max_tip_amount = 145`. See the *exp* parameter in [currencies.json](#), it shows the number of digits past the decimal point for each currency (2 for the majority of currencies). Defaults to 0
- **suggested\_tip\_amounts** – A JSON-serialized array of suggested amounts of tips in the *smallest units* of the currency (integer, **not** float/double). At most 4 suggested tip amounts can be specified. The suggested tip amounts must be positive, passed in a strictly increased order and must not exceed *max\_tip\_amount*.
- **start\_parameter** – 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** – JSON-serialized data about the invoice, which will be shared with the payment provider. A detailed description of required fields should be provided by the payment provider.
- **photo\_url** – URL of the product photo for the invoice. Can be a photo of the goods or a marketing image for a service. People like it better when they see what they are paying for.
- **photo\_size** – Photo size in bytes
- **photo\_width** – Photo width
- **photo\_height** – Photo height
- **need\_name** – Pass True if you require the user's full name to complete the order
- **need\_phone\_number** – Pass True if you require the user's phone number to complete the order
- **need\_email** – Pass True if you require the user's email address to complete the order
- **need\_shipping\_address** – Pass True if you require the user's shipping address to complete the order

- **send\_phone\_number\_to\_provider** – Pass True if the user’s phone number should be sent to provider
- **send\_email\_to\_provider** – Pass True if the user’s email address should be sent to provider
- **is\_flexible** – Pass True if the final price depends on the shipping method
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method [aiogram.methods.send\\_invoice.SendInvoice](#)

**answer\_location**(*latitude: float, longitude: float, message\_thread\_id: int | None = None, horizontal\_accuracy: float | None = None, live\_period: int | None = None, heading: int | None = None, proximity\_alert\_radius: int | None = None, disable\_notification: bool | None = None, protect\_content: bool | None = sentinel.UNSET\_PROTECT\_CONTENT, reply\_to\_message\_id: int | None = None, allow\_sending\_without\_reply: bool | None = None, reply\_markup: [InlineKeyboardMarkup](#) | [ReplyKeyboardMarkup](#) | [ReplyKeyboardRemove](#) | [ForceReply](#) | None = None, \*\*kwargs: Any*) → [SendLocation](#)

Shortcut for method [aiogram.methods.send\\_location.SendLocation](#) will automatically fill method attributes:

- **chat\_id**

Use this method to send point on the map. On success, the sent [aiogram.types.message.Message](#) is returned.

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

#### Parameters

- **latitude** – Latitude of the location
- **longitude** – Longitude of the location
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **horizontal\_accuracy** – The radius of uncertainty for the location, measured in meters; 0-1500
- **live\_period** – Period in seconds for which the location will be updated (see [Live Locations](#), should be between 60 and 86400).
- **heading** – For live locations, a direction in which the user is moving, in degrees. Must be between 1 and 360 if specified.
- **proximity\_alert\_radius** – 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** – Sends the message *silently*. Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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**

instance of method `aiogram.methods.send_location.SendLocation`

**answer\_media\_group**(*media*: List[InputMediaAudio | InputMediaDocument | InputMediaPhoto | InputMediaVideo], *message\_thread\_id*: int | None = None, *disable\_notification*: bool | None = None, *protect\_content*: bool | None = sentinel.UNSET\_PROTECT\_CONTENT, *reply\_to\_message\_id*: int | None = None, *allow\_sending\_without\_reply*: bool | None = None, *\*\*kwargs*: Any) → *SendMediaGroup*

Shortcut for method `aiogram.methods.send_media_group.SendMediaGroup` will automatically fill method attributes:

- `chat_id`

Use this method to send a group of photos, videos, documents or audios as an album. Documents and audio files can be only grouped 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** – A JSON-serialized array describing messages to be sent, must include 2-10 items
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **disable\_notification** – Sends messages *silently*. Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent messages from forwarding and saving
- **reply\_to\_message\_id** – If the messages are a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found

**Returns**

instance of method `aiogram.methods.send_media_group.SendMediaGroup`

**answer\_photo**(*photo*: InputFile | str, *message\_thread\_id*: int | None = None, *caption*: str | None = None, *parse\_mode*: str | None = sentinel.UNSET\_PARSE\_MODE, *caption\_entities*: List[MessageEntity] | None = None, *has\_spoiler*: bool | None = None, *disable\_notification*: bool | None = None, *protect\_content*: bool | None = sentinel.UNSET\_PROTECT\_CONTENT, *reply\_to\_message\_id*: int | None = None, *allow\_sending\_without\_reply*: bool | None = None, *reply\_markup*: InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply | None = None, *\*\*kwargs*: Any) → *SendPhoto*

Shortcut for method `aiogram.methods.send_photo.SendPhoto` will automatically fill method attributes:

- `chat_id`

Use this method to send photos. On success, the sent `aiogram.types.message.Message` is returned.

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

#### Parameters

- **photo** – Photo to send. Pass a `file_id` as `String` to send a photo that exists on the Telegram servers (recommended), pass an `HTTP URL` as a `String` for Telegram to get a photo from the Internet, or upload a new photo using `multipart/form-data`. The photo must be at most 10 MB in size. The photo's width and height must not exceed 10000 in total. Width and height ratio must be at most 20. *More information on Sending Files »*
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **caption** – Photo caption (may also be used when resending photos by `file_id`), 0-1024 characters after entities parsing
- **parse\_mode** – Mode for parsing entities in the photo caption. See [formatting options](#) for more details.
- **caption\_entities** – A JSON-serialized list of special entities that appear in the caption, which can be specified instead of `parse_mode`
- **has\_spoiler** – Pass `True` if the photo needs to be covered with a spoiler animation
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass `True` if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method `aiogram.methods.send_photo.SendPhoto`

**answer\_poll**(*question: str, options: List[str], message\_thread\_id: int | None = None, is\_anonymous: bool | None = None, type: str | None = None, allows\_multiple\_answers: bool | None = None, correct\_option\_id: int | None = None, explanation: str | None = None, explanation\_parse\_mode: str | None = sentinel.UNSET\_PARSE\_MODE, explanation\_entities: List[MessageEntity] | None = None, open\_period: int | None = None, close\_date: datetime.datetime | datetime.timedelta | int | None = None, is\_closed: bool | None = None, disable\_notification: bool | None = None, protect\_content: bool | None = sentinel.UNSET\_PROTECT\_CONTENT, reply\_to\_message\_id: int | None = None, allow\_sending\_without\_reply: bool | None = None, reply\_markup: InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply | None = None, \*\*kwargs: Any*) → *SendPoll*

Shortcut for method `aiogram.methods.send_poll.SendPoll` will automatically fill method attributes:

- `chat_id`

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

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

#### Parameters

- **question** – Poll question, 1-300 characters
- **options** – A JSON-serialized list of answer options, 2-10 strings 1-100 characters each
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **is\_anonymous** – True, if the poll needs to be anonymous, defaults to True
- **type** – Poll type, ‘quiz’ or ‘regular’, defaults to ‘regular’
- **allows\_multiple\_answers** – True, if the poll allows multiple answers, ignored for polls in quiz mode, defaults to False
- **correct\_option\_id** – 0-based identifier of the correct answer option, required for polls in quiz mode
- **explanation** – 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** – Mode for parsing entities in the explanation. See [formatting options](#) for more details.
- **explanation\_entities** – A JSON-serialized list of special entities that appear in the poll explanation, which can be specified instead of `parse_mode`
- **open\_period** – Amount of time in seconds the poll will be active after creation, 5-600. Can’t be used together with `close_date`.
- **close\_date** – 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** – Pass True if the poll needs to be immediately closed. This can be useful for poll preview.
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method `aiogram.methods.send_poll.SendPoll`

```
answer_dice(message_thread_id: int | None = None, emoji: str | None = None, disable_notification: bool | None = None, protect_content: bool | None = sentinel.UNSET_PROTECT_CONTENT,
reply_to_message_id: int | None = None, allow_sending_without_reply: bool | None = None,
reply_markup: InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply | None = None, **kwargs: Any) → SendDice
```

Shortcut for method `aiogram.methods.send_dice.SendDice` will automatically fill method attributes:

- `chat_id`

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

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

#### Parameters

- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **emoji** – 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** – Sends the message `silently`. Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method `aiogram.methods.send_dice.SendDice`

```
answer_sticker(sticker: InputFile | str, message_thread_id: int | None = None, emoji: str | None = None,  
                disable_notification: bool | None = None, protect_content: bool | None =  
                sentinel.UNSET_PROTECT_CONTENT, reply_to_message_id: int | None = None,  
                allow_sending_without_reply: bool | None = None, reply_markup: InlineKeyboardMarkup  
                | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply | None = None, **kwargs:  
                Any) → SendSticker
```

Shortcut for method `aiogram.methods.send_sticker.SendSticker` will automatically fill method attributes:

- `chat_id`

Use this method to send static `.WEBP`, `animated .TGS`, or `video .WEBM` stickers. On success, the sent `aiogram.types.message.Message` is returned.

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

#### Parameters

- **sticker** – Sticker to send. Pass a `file_id` as `String` to send a file that exists on the Telegram servers (recommended), pass an `HTTP URL` as a `String` for Telegram to get a `.WEBP` sticker from the Internet, or upload a new `.WEBP` or `.TGS` sticker using `multipart/form-data`. *More information on Sending Files »*. Video stickers can only be sent by a `file_id`. Animated stickers can't be sent via an `HTTP URL`.
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **emoji** – Emoji associated with the sticker; only for just uploaded stickers

- **disable\_notification** – Sends the message *silently*. Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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**

instance of method `aiogram.methods.send_sticker.SendSticker`

**answer\_venue**(*latitude: float, longitude: float, title: str, address: str, message\_thread\_id: int | None = None, foursquare\_id: str | None = None, foursquare\_type: str | None = None, google\_place\_id: str | None = None, google\_place\_type: str | None = None, disable\_notification: bool | None = None, protect\_content: bool | None = sentinel.UNSET\_PROTECT\_CONTENT, reply\_to\_message\_id: int | None = None, allow\_sending\_without\_reply: bool | None = None, reply\_markup: InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply | None = None, \*\*kwargs: Any*) → *SendVenue*

Shortcut for method `aiogram.methods.send_venue.SendVenue` will automatically fill method attributes:

- **chat\_id**

Use this method to send information about a venue. On success, the sent `aiogram.types.message.Message` is returned.

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

**Parameters**

- **latitude** – Latitude of the venue
- **longitude** – Longitude of the venue
- **title** – Name of the venue
- **address** – Address of the venue
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **foursquare\_id** – Foursquare identifier of the venue
- **foursquare\_type** – Foursquare type of the venue, if known. (For example, ‘arts\_entertainment/default’, ‘arts\_entertainment/aquarium’ or ‘food/icecream’.)
- **google\_place\_id** – Google Places identifier of the venue
- **google\_place\_type** – Google Places type of the venue. (See *supported types*.)
- **disable\_notification** – Sends the message *silently*. Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found



- **reply\_markup** – 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

instance of method [aiogram.methods.send\\_venue.SendVenue](#)

**answer\_video**(*video*: [InputFile](#) | *str*, *message\_thread\_id*: *int* | *None* = *None*, *duration*: *int* | *None* = *None*, *width*: *int* | *None* = *None*, *height*: *int* | *None* = *None*, *thumbnail*: [InputFile](#) | *str* | *None* = *None*, *caption*: *str* | *None* = *None*, *parse\_mode*: *str* | *None* = *sentinel.UNSET\_PARSE\_MODE*, *caption\_entities*: *List*[[MessageEntity](#)] | *None* = *None*, *has\_spoiler*: *bool* | *None* = *None*, *supports\_streaming*: *bool* | *None* = *None*, *disable\_notification*: *bool* | *None* = *None*, *protect\_content*: *bool* | *None* = *sentinel.UNSET\_PROTECT\_CONTENT*, *reply\_to\_message\_id*: *int* | *None* = *None*, *allow\_sending\_without\_reply*: *bool* | *None* = *None*, *reply\_markup*: [InlineKeyboardMarkup](#) | [ReplyKeyboardMarkup](#) | [ReplyKeyboardRemove](#) | [ForceReply](#) | *None* = *None*, *\*\*kwargs*: *Any*) → [SendVideo](#)

Shortcut for method [aiogram.methods.send\\_video.SendVideo](#) will automatically fill method attributes:

- **chat\_id**

Use this method to send video files, Telegram clients support MPEG4 videos (other formats may be sent as [aiogram.types.document.Document](#)). On success, the sent [aiogram.types.message.Message](#) is returned. Bots can currently send video files of up to 50 MB in size, this limit may be changed in the future.

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

#### Parameters

- **video** – Video to send. Pass a *file\_id* as *String* to send a video that exists on the Telegram servers (recommended), pass an HTTP URL as a *String* for Telegram to get a video from the Internet, or upload a new video using multipart/form-data. [More information on Sending Files](#) »
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **duration** – Duration of sent video in seconds
- **width** – Video width
- **height** – Video height
- **thumbnail** – Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can't be reused and can be only uploaded as a new file, so you can pass 'attach://<file\_attach\_name>' if the thumbnail was uploaded using multipart/form-data under <file\_attach\_name>. [More information on Sending Files](#) »
- **caption** – Video caption (may also be used when resending videos by *file\_id*), 0-1024 characters after entities parsing
- **parse\_mode** – Mode for parsing entities in the video caption. See [formatting options](#) for more details.
- **caption\_entities** – A JSON-serialized list of special entities that appear in the caption, which can be specified instead of *parse\_mode*
- **has\_spoiler** – Pass *True* if the video needs to be covered with a spoiler animation



- **supports\_streaming** – Pass True if the uploaded video is suitable for streaming
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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**

instance of method [aiogram.methods.send\\_video.SendVideo](#)

**answer\_video\_note**(*video\_note*: [InputFile](#) | *str*, *message\_thread\_id*: *int* | *None* = *None*, *duration*: *int* | *None* = *None*, *length*: *int* | *None* = *None*, *thumbnail*: [InputFile](#) | *str* | *None* = *None*, *disable\_notification*: *bool* | *None* = *None*, *protect\_content*: *bool* | *None* = *sentinel.UNSET\_PROTECT\_CONTENT*, *reply\_to\_message\_id*: *int* | *None* = *None*, *allow\_sending\_without\_reply*: *bool* | *None* = *None*, *reply\_markup*: [InlineKeyboardMarkup](#) | [ReplyKeyboardMarkup](#) | [ReplyKeyboardRemove](#) | [ForceReply](#) | *None* = *None*, *\*\*kwargs*: *Any*) → [SendVideoNote](#)

Shortcut for method [aiogram.methods.send\\_video\\_note.SendVideoNote](#) will automatically fill method attributes:

- **chat\_id**

As of v.4.0, Telegram clients support rounded square MPEG4 videos of up to 1 minute long. Use this method to send video messages. On success, the sent [aiogram.types.message.Message](#) is returned.

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

**Parameters**

- **video\_note** – Video note to send. Pass a *file\_id* as *String* to send a video note that exists on the Telegram servers (recommended) or upload a new video using multipart/form-data. [More information on Sending Files](#) ». Sending video notes by a URL is currently unsupported
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **duration** – Duration of sent video in seconds
- **length** – Video width and height, i.e. diameter of the video message
- **thumbnail** – Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can't be reused and can be only uploaded as a new file, so you can pass 'attach://<file\_attach\_name>' if the thumbnail was uploaded using multipart/form-data under <file\_attach\_name>. [More information on Sending Files](#) »
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving

- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method [aiogram.methods.send\\_video\\_note.SendVideoNote](#)

**answer\_voice**(voice: [InputFile](#) | str, message\_thread\_id: int | None = None, caption: str | None = None, parse\_mode: str | None = sentinel.UNSET\_PARSE\_MODE, caption\_entities: List[[MessageEntity](#)] | None = None, duration: int | None = None, disable\_notification: bool | None = None, protect\_content: bool | None = sentinel.UNSET\_PROTECT\_CONTENT, reply\_to\_message\_id: int | None = None, allow\_sending\_without\_reply: bool | None = None, reply\_markup: [InlineKeyboardMarkup](#) | [ReplyKeyboardMarkup](#) | [ReplyKeyboardRemove](#) | [ForceReply](#) | None = None, \*\*kwargs: Any) → [SendVoice](#)

Shortcut for method [aiogram.methods.send\\_voice.SendVoice](#) will automatically fill method attributes:

- chat\_id

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 [aiogram.types.audio.Audio](#) or [aiogram.types.document.Document](#)). On success, the sent [aiogram.types.message.Message](#) is returned. Bots can currently send voice messages of up to 50 MB in size, this limit may be changed in the future.

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

#### Parameters

- **voice** – Audio file to send. Pass a file\_id as String to send a file that exists on the Telegram servers (recommended), 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 information on Sending Files](#) »
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **caption** – Voice message caption, 0-1024 characters after entities parsing
- **parse\_mode** – Mode for parsing entities in the voice message caption. See [formatting options](#) for more details.
- **caption\_entities** – A JSON-serialized list of special entities that appear in the caption, which can be specified instead of *parse\_mode*
- **duration** – Duration of the voice message in seconds
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found

- **reply\_markup** – 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**

instance of method [aiogram.methods.send\\_voice.SendVoice](#)

**ChatPermissions**

```
class aiogram.types.chat_permissions.ChatPermissions(*, can_send_messages: bool | None = None,
                                                    can_send_audios: bool | None = None,
                                                    can_send_documents: bool | None = None,
                                                    can_send_photos: bool | None = None,
                                                    can_send_videos: bool | None = None,
                                                    can_send_video_notes: bool | None = None,
                                                    can_send_voice_notes: bool | None = None,
                                                    can_send_polls: bool | None = None,
                                                    can_send_other_messages: bool | None =
None, can_add_web_page_previews: bool |
None = None, can_change_info: bool | None =
None, can_invite_users: bool | None = None,
can_pin_messages: bool | None = None,
can_manage_topics: bool | None = None,
**extra_data: Any)
```

Describes actions that a non-administrator user is allowed to take in a chat.

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

**can\_send\_messages: bool | None**

*Optional.* True, if the user is allowed to send text messages, contacts, invoices, locations and venues

**can\_send\_audios: bool | None**

*Optional.* True, if the user is allowed to send audios

**can\_send\_documents: bool | None**

*Optional.* True, if the user is allowed to send documents

**can\_send\_photos: bool | None**

*Optional.* True, if the user is allowed to send photos

**can\_send\_videos: bool | None**

*Optional.* True, if the user is allowed to send videos

**can\_send\_video\_notes: bool | None**

*Optional.* True, if the user is allowed to send video notes

**can\_send\_voice\_notes: bool | None**

*Optional.* True, if the user is allowed to send voice notes

**can\_send\_polls: bool | None**

*Optional.* True, if the user is allowed to send polls

**can\_send\_other\_messages: bool | None**

*Optional.* True, if the user is allowed to send animations, games, stickers and use inline bots

**can\_add\_web\_page\_previews:** `bool | None`

*Optional.* True, if the user is allowed to add web page previews to their messages

**can\_change\_info:** `bool | None`

*Optional.* True, if the user is allowed to change the chat title, photo and other settings. Ignored in public supergroups

**can\_invite\_users:** `bool | None`

*Optional.* True, if the user is allowed to invite new users to the chat

**can\_pin\_messages:** `bool | None`

*Optional.* True, if the user is allowed to pin messages. Ignored in public supergroups

**can\_manage\_topics:** `bool | None`

*Optional.* True, if the user is allowed to create forum topics. If omitted defaults to the value of `can_pin_messages`

## ChatPhoto

```
class aiogram.types.chat_photo.ChatPhoto(*, small_file_id: str, small_file_unique_id: str, big_file_id: str,
                                          big_file_unique_id: str, **extra_data: Any)
```

This object represents a chat photo.

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

**small\_file\_id:** `str`

File identifier of small (160x160) chat photo. This `file_id` can be used only for photo download and only for as long as the photo is not changed.

**small\_file\_unique\_id:** `str`

Unique file identifier of small (160x160) chat photo, which is supposed to be the same over time and for different bots. Can't be used to download or reuse the file.

**big\_file\_id:** `str`

File identifier of big (640x640) chat photo. This `file_id` can be used only for photo download and only for as long as the photo is not changed.

**big\_file\_unique\_id:** `str`

Unique file identifier of big (640x640) chat photo, which is supposed to be the same over time and for different bots. Can't be used to download or reuse the file.

## ChatShared

```
class aiogram.types.chat_shared.ChatShared(*, request_id: int, chat_id: int, **extra_data: Any)
```

This object contains information about the chat whose identifier was shared with the bot using a [`aiogram.types.keyboard\_button\_request\_chat.KeyboardButtonRequestChat`](#) button.

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

**request\_id:** `int`

Identifier of the request

**chat\_id: int**

Identifier of the shared chat. This number may have more than 32 significant bits and some programming languages may have difficulty/silent defects in interpreting it. But it has at most 52 significant bits, so a 64-bit integer or double-precision float type are safe for storing this identifier. The bot may not have access to the chat and could be unable to use this identifier, unless the chat is already known to the bot by some other means.

## Contact

```
class aiogram.types.contact.Contact(*, phone_number: str, first_name: str, last_name: str | None = None,
                                     user_id: int | None = None, vcard: str | None = None, **extra_data:
                                     Any)
```

This object represents a phone contact.

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

**phone\_number: str**

Contact's phone number

**first\_name: str**

Contact's first name

**last\_name: str | None**

*Optional.* Contact's last name

**user\_id: int | None**

*Optional.* Contact's user identifier in Telegram. This number may have more than 32 significant bits and some programming languages may have difficulty/silent defects in interpreting it. But it has at most 52 significant bits, so a 64-bit integer or double-precision float type are safe for storing this identifier.

**vcard: str | None**

*Optional.* Additional data about the contact in the form of a [vCard](#)

## Dice

```
class aiogram.types.dice.Dice(*, emoji: str, value: int, **extra_data: Any)
```

This object represents an animated emoji that displays a random value.

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

**emoji: str**

Emoji on which the dice throw animation is based

**value: int**

Value of the dice, 1-6 for “, “ and “ base emoji, 1-5 for “ and “ base emoji, 1-64 for “ base emoji

```
class aiogram.types.dice.DiceEmoji
```

DICE = ''

DART = ''

BASKETBALL = ''

FOOTBALL = ''

```
SLOT_MACHINE = ''  
BOWLING = ''
```

## Document

```
class aiogram.types.document.Document(*, file_id: str, file_unique_id: str, thumbnail: PhotoSize | None =  
    None, file_name: str | None = None, mime_type: str | None =  
    None, file_size: int | None = None, **extra_data: Any)
```

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

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

**file\_id: str**

Identifier for this file, which can be used to download or reuse the file

**file\_unique\_id: str**

Unique identifier for this file, which is supposed to be the same over time and for different bots. Can't be used to download or reuse the file.

**thumbnail: PhotoSize | None**

*Optional.* Document thumbnail as defined by sender

**file\_name: str | None**

*Optional.* Original filename as defined by sender

**mime\_type: str | None**

*Optional.* MIME type of the file as defined by sender

**file\_size: int | None**

*Optional.* File size in bytes. It can be bigger than  $2^{31}$  and some programming languages may have difficulty/silent defects in interpreting it. But it has at most 52 significant bits, so a signed 64-bit integer or double-precision float type are safe for storing this value.

## File

```
class aiogram.types.file.File(*, file_id: str, file_unique_id: str, file_size: int | None = None, file_path: str |  
    None = None, **extra_data: Any)
```

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>](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 [aiogram.methods.get\\_file.GetFile](#).

The maximum file size to download is 20 MB

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

**file\_id: str**

Identifier for this file, which can be used to download or reuse the file

**file\_unique\_id: str**

Unique identifier for this file, which is supposed to be the same over time and for different bots. Can't be used to download or reuse the file.

**file\_size:** `int` | `None`

*Optional.* File size in bytes. It can be bigger than  $2^{31}$  and some programming languages may have difficulty/silent defects in interpreting it. But it has at most 52 significant bits, so a signed 64-bit integer or double-precision float type are safe for storing this value.

**file\_path:** `str` | `None`

*Optional.* File path. Use `https://api.telegram.org/file/bot<token>/<file_path>` to get the file.

## ForceReply

```
class aiogram.types.force_reply.ForceReply(*, force_reply: Literal[True] = True,
                                           input_field_placeholder: str | None = None, selective: bool |
                                           None = None, **extra_data: Any)
```

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

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

- Explain the user how to send a command with parameters (e.g. `/newpoll question answer1 answer2`). May be appealing for hardcore users but lacks modern day polish.
- Guide the user through a step-by-step process. 'Please send me your question', 'Cool, now let's add the first answer option', 'Great. Keep adding answer options, then send `/done` when you're ready'.

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

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

**force\_reply:** `Literal[True]`

Shows reply interface to the user, as if they manually selected the bot's message and tapped 'Reply'

**input\_field\_placeholder:** `str` | `None`

*Optional.* The placeholder to be shown in the input field when the reply is active; 1-64 characters

**selective:** `bool` | `None`

*Optional.* Use this parameter if you want to force reply from specific users only. Targets: 1) users that are @mentioned in the *text* of the `aiogram.types.message.Message` object; 2) if the bot's message is a reply (has `reply_to_message_id`), sender of the original message.

## ForumTopic

```
class aiogram.types.forum_topic.ForumTopic(*, message_thread_id: int, name: str, icon_color: int,
                                           icon_custom_emoji_id: str | None = None, **extra_data:
                                           Any)
```

This object represents a forum topic.

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

**message\_thread\_id: int**

Unique identifier of the forum topic

**name: str**

Name of the topic

**icon\_color: int**

Color of the topic icon in RGB format

**icon\_custom\_emoji\_id: str | None**

*Optional.* Unique identifier of the custom emoji shown as the topic icon

## ForumTopicClosed

**class** aiogram.types.forum\_topic\_closed.**ForumTopicClosed**(\*\*extra\_data: Any)

This object represents a service message about a forum topic closed in the chat. Currently holds no information.

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

## ForumTopicCreated

**class** aiogram.types.forum\_topic\_created.**ForumTopicCreated**(\* , name: str, icon\_color: int, icon\_custom\_emoji\_id: str | None = None, \*\*extra\_data: Any)

This object represents a service message about a new forum topic created in the chat.

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

**name: str**

Name of the topic

**icon\_color: int**

Color of the topic icon in RGB format

**icon\_custom\_emoji\_id: str | None**

*Optional.* Unique identifier of the custom emoji shown as the topic icon

## ForumTopicEdited

**class** aiogram.types.forum\_topic\_edited.**ForumTopicEdited**(\* , name: str | None = None, icon\_custom\_emoji\_id: str | None = None, \*\*extra\_data: Any)

This object represents a service message about an edited forum topic.

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

**name: str | None**

*Optional.* New name of the topic, if it was edited

**icon\_custom\_emoji\_id: str | None**

*Optional.* New identifier of the custom emoji shown as the topic icon, if it was edited; an empty string if the icon was removed



## ForumTopicReopened

```
class aiogram.types.forum_topic_reopened.ForumTopicReopened(**extra_data: Any)
```

This object represents a service message about a forum topic reopened in the chat. Currently holds no information.

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

## GeneralForumTopicHidden

```
class aiogram.types.general_forum_topic_hidden.GeneralForumTopicHidden(**extra_data: Any)
```

This object represents a service message about General forum topic hidden in the chat. Currently holds no information.

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

## GeneralForumTopicUnhidden

```
class aiogram.types.general_forum_topic_unhidden.GeneralForumTopicUnhidden(**extra_data: Any)
```

This object represents a service message about General forum topic unhidden in the chat. Currently holds no information.

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

## InlineKeyboardButton

```
class aiogram.types.inline_keyboard_button.InlineKeyboardButton(*, text: str, url: str | None =
    None, callback_data: str | None = None, web_app: WebAppInfo
    | None = None, login_url: LoginUrl | None = None,
    switch_inline_query: str | None = None,
    switch_inline_query_current_chat: str | None = None,
    switch_inline_query_chosen_chat: SwitchInlineQueryChosenChat |
    None = None, callback_game: CallbackGame | None = None,
    pay: bool | None = None,
    **extra_data: Any)
```

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

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

**text:** **str**

Label text on the button

**url:** `str` | `None`

*Optional.* HTTP or tg:// URL to be opened when the button is pressed. Links `tg://user?id=<user_id>` can be used to mention a user by their ID without using a username, if this is allowed by their privacy settings.

**callback\_data:** `str` | `None`

*Optional.* Data to be sent in a `callback query` to the bot when button is pressed, 1-64 bytes

**web\_app:** `WebAppInfo` | `None`

*Optional.* Description of the `Web App` that will be launched when the user presses the button. The Web App will be able to send an arbitrary message on behalf of the user using the method `aiogram.methods.answer_web_app_query.AnswerWebAppQuery`. Available only in private chats between a user and the bot.

**login\_url:** `LoginUrl` | `None`

*Optional.* An HTTPS URL used to automatically authorize the user. Can be used as a replacement for the `Telegram Login Widget`.

**switch\_inline\_query:** `str` | `None`

*Optional.* If set, pressing the button will prompt the user to select one of their chats, open that chat and insert the bot's username and the specified inline query in the input field. May be empty, in which case just the bot's username will be inserted.

**switch\_inline\_query\_current\_chat:** `str` | `None`

*Optional.* If set, pressing the button will insert the bot's username and the specified inline query in the current chat's input field. May be empty, in which case only the bot's username will be inserted.

**switch\_inline\_query\_chosen\_chat:** `SwitchInlineQueryChosenChat` | `None`

*Optional.* If set, pressing the button will prompt the user to select one of their chats of the specified type, open that chat and insert the bot's username and the specified inline query in the input field

**callback\_game:** `CallbackGame` | `None`

*Optional.* Description of the game that will be launched when the user presses the button.

**pay:** `bool` | `None`

*Optional.* Specify True, to send a `Pay button`.

## InlineKeyboardMarkup

```
class aiogram.types.inline_keyboard_markup.InlineKeyboardMarkup(*, inline_keyboard:
    List[List[InlineKeyboardButton]],
    **extra_data: Any)
```

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*.

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

**inline\_keyboard:** `List[List[InlineKeyboardButton]]`

Array of button rows, each represented by an Array of `aiogram.types.inline_keyboard_button.InlineKeyboardButton` objects

## InputFile

**class** aiogram.types.input\_file.**InputFile**(filename: str | None = None, chunk\_size: int = 65536)

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.

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

**abstract async read**(bot: Bot) → AsyncGenerator[bytes, None]

**class** aiogram.types.input\_file.**BufferedInputFile**(file: bytes, filename: str, chunk\_size: int = 65536)

**classmethod from\_file**(path: str | Path, filename: str | None = None, chunk\_size: int = 65536) → *BufferedInputFile*

Create buffer from file

### Parameters

- **path** – Path to file
- **filename** – Filename to be propagated to telegram. By default, will be parsed from path
- **chunk\_size** – Uploading chunk size

### Returns

instance of *BufferedInputFile*

**async read**(bot: Bot) → AsyncGenerator[bytes, None]

**class** aiogram.types.input\_file.**FSInputFile**(path: str | Path, filename: str | None = None, chunk\_size: int = 65536)

**async read**(bot: Bot) → AsyncGenerator[bytes, None]

**class** aiogram.types.input\_file.**URLInputFile**(url: str, headers: Dict[str, Any] | None = None, filename: str | None = None, chunk\_size: int = 65536, timeout: int = 30, bot: Bot | None = None)

**async read**(bot: Bot) → AsyncGenerator[bytes, None]

## InputMedia

**class** aiogram.types.input\_media.**InputMedia**(\*\*extra\_data: Any)

This object represents the content of a media message to be sent. It should be one of

- *aiogram.types.input\_media\_animation.InputMediaAnimation*
- *aiogram.types.input\_media\_document.InputMediaDocument*
- *aiogram.types.input\_media\_audio.InputMediaAudio*
- *aiogram.types.input\_media\_photo.InputMediaPhoto*
- *aiogram.types.input\_media\_video.InputMediaVideo*

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

## InputMediaAnimation

```
class aiogram.types.input_media_animation.InputMediaAnimation(*, type: ~typing.Literal[<InputMediaType.ANIMATION: 'animation'>] = InputMediaType.ANIMATION, media: str | ~aiogram.types.input_file.InputFile, thumbnail: ~aiogram.types.input_file.InputFile | str | None = None, caption: str | None = None, parse_mode: str | None = sentinel.UNSET_PARSE_MODE, caption_entities: ~typing.List[~aiogram.types.message_entity.MessageEntity] | None = None, width: int | None = None, height: int | None = None, duration: int | None = None, has_spoiler: bool | None = None, **extra_data: ~typing.Any)
```

Represents an animation file (GIF or H.264/MPEG-4 AVC video without sound) to be sent.

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

**type:** `Literal[InputMediaType.ANIMATION]`

Type of the result, must be *animation*

**media:** `str | InputFile`

File to send. Pass a `file_id` to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet, or pass `'attach://<file_attach_name>'` to upload a new one using multipart/form-data under `<file_attach_name>` name. [More information on Sending Files »](#)

**thumbnail:** `InputFile | str | None`

*Optional.* Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can't be reused and can be only uploaded as a new file, so you can pass `'attach://<file_attach_name>'` if the thumbnail was uploaded using multipart/form-data under `<file_attach_name>`. [More information on Sending Files »](#)

**caption:** `str | None`

*Optional.* Caption of the animation to be sent, 0-1024 characters after entities parsing

**parse\_mode:** `str | None`

*Optional.* Mode for parsing entities in the animation caption. See [formatting options](#) for more details.

**caption\_entities:** `List[MessageEntity] | None`

*Optional.* List of special entities that appear in the caption, which can be specified instead of *parse\_mode*

**width:** `int | None`

*Optional.* Animation width

**height:** `int | None`

*Optional.* Animation height

**duration:** `int | None`

*Optional.* Animation duration in seconds

**has\_spoiler:** `bool | None`

*Optional.* Pass True if the animation needs to be covered with a spoiler animation

## InputMediaAudio

```
class aiogram.types.input_media_audio.InputMediaAudio(*, type: ~typing.Literal[<InlineQueryResultType.AUDIO: 'audio'>] = InputMediaType.AUDIO, media: str | ~aiogram.types.input_file.InputFile, thumbnail: ~aiogram.types.input_file.InputFile | str | None = None, caption: str | None = None, parse_mode: str | None = sentinel.UNSET_PARSE_MODE, caption_entities: ~typing.List[~aiogram.types.message_entity.MessageEntity] | None = None, duration: int | None = None, performer: str | None = None, title: str | None = None, **extra_data: ~typing.Any)
```

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

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

**type:** `Literal[InputMediaType.AUDIO]`

Type of the result, must be *audio*

**media:** `str | InputFile`

File to send. Pass a `file_id` to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet, or pass `'attach://<file_attach_name>'` to upload a new one using multipart/form-data under `<file_attach_name>` name. *More information on Sending Files »*

**thumbnail:** `InputFile | str | None`

*Optional.* Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can't be reused and can be only uploaded as a new file, so you can pass `'attach://<file_attach_name>'` if the thumbnail was uploaded using multipart/form-data under `<file_attach_name>`. *More information on Sending Files »*

**caption:** `str | None`

*Optional.* Caption of the audio to be sent, 0-1024 characters after entities parsing

**parse\_mode:** `str | None`

*Optional.* Mode for parsing entities in the audio caption. See [formatting options](#) for more details.

**caption\_entities:** `List[MessageEntity] | None`

*Optional.* List of special entities that appear in the caption, which can be specified instead of `parse_mode`

**duration:** `int | None`

*Optional.* Duration of the audio in seconds

**performer:** `str | None`

*Optional.* Performer of the audio

**title:** `str | None`

*Optional.* Title of the audio

## InputMediaDocument

```
class aiogram.types.input_media_document.InputMediaDocument(*, type: ~typing.Literal[<InlineQueryResultType.DOCUMENT: 'document'>] = InputMediaType.DOCUMENT, media: str | ~aiogram.types.input_file.InputFile, thumbnail: ~aiogram.types.input_file.InputFile | str | None = None, caption: str | None = None, parse_mode: str | None = sentinel.UNSET_PARSE_MODE, caption_entities: ~typing.List[~aiogram.types.message_entity.MessageEntity] | None = None, disable_content_type_detection: bool | None = None, **extra_data: ~typing.Any)
```

Represents a general file to be sent.

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

**type:** `Literal[InputMediaType.DOCUMENT]`

Type of the result, must be *document*

**media:** `str | InputFile`

File to send. Pass a `file_id` to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet, or pass `'attach://<file_attach_name>'` to upload a new one using multipart/form-data under `<file_attach_name>` name. *More information on Sending Files »*

**thumbnail:** `InputFile | str | None`

*Optional.* Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can't be reused and can be only uploaded as a new file, so you can pass `'attach://<file_attach_name>'` if the thumbnail was uploaded using multipart/form-data under `<file_attach_name>`. *More information on Sending Files »*

**caption:** `str | None`

*Optional.* Caption of the document to be sent, 0-1024 characters after entities parsing

**parse\_mode:** `str | None`

*Optional.* Mode for parsing entities in the document caption. See [formatting options](#) for more details.

**caption\_entities:** `List[MessageEntity] | None`

*Optional.* List of special entities that appear in the caption, which can be specified instead of *parse\_mode*

**disable\_content\_type\_detection:** `bool` | `None`

*Optional.* Disables automatic server-side content type detection for files uploaded using multipart/form-data. Always `True`, if the document is sent as part of an album.

## InputMediaPhoto

```
class aiogram.types.input_media_photo.InputMediaPhoto(*, type: ~typing.Literal[<InlineQueryResultType.PHOTO: 'photo'>] = InputMediaType.PHOTO, media: str | ~aiogram.types.input_file.InputFile, caption: str | None = None, parse_mode: str | None = sentinel.UNSET_PARSE_MODE, caption_entities: ~typing.List[~aiogram.types.message_entity.MessageEntity] | None = None, has_spoiler: bool | None = None, **extra_data: ~typing.Any)
```

Represents a photo to be sent.

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

**type:** `Literal[InputMediaType.PHOTO]`

Type of the result, must be *photo*

**media:** `str` | `InputFile`

File to send. Pass a `file_id` to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet, or pass `'attach://<file_attach_name>'` to upload a new one using multipart/form-data under `<file_attach_name>` name. [More information on Sending Files »](#)

**caption:** `str` | `None`

*Optional.* Caption of the photo to be sent, 0-1024 characters after entities parsing

**parse\_mode:** `str` | `None`

*Optional.* Mode for parsing entities in the photo caption. See [formatting options](#) for more details.

**caption\_entities:** `List[MessageEntity]` | `None`

*Optional.* List of special entities that appear in the caption, which can be specified instead of `parse_mode`

**has\_spoiler:** `bool` | `None`

*Optional.* Pass `True` if the photo needs to be covered with a spoiler animation

## InputMediaVideo

```
class aiogram.types.input_media_video.InputMediaVideo(*, type: ~typing.Literal[<InlineQueryResultType.VIDEO: 'video'>] = InputMediaType.VIDEO, media: str | ~aiogram.types.input_file.InputFile, thumbnail: ~aiogram.types.input_file.InputFile | str | None = None, caption: str | None = None, parse_mode: str | None = sentinel.UNSET_PARSE_MODE, caption_entities: ~typing.List[~aiogram.types.message_entity.MessageEntity] | None = None, width: int | None = None, height: int | None = None, duration: int | None = None, supports_streaming: bool | None = None, has_spoiler: bool | None = None, **extra_data: ~typing.Any)
```

Represents a video to be sent.

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

**type:** `Literal[InputMediaType.VIDEO]`

Type of the result, must be *video*

**media:** `str | InputFile`

File to send. Pass a file\_id to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet, or pass 'attach://<file\_attach\_name>' to upload a new one using multipart/form-data under <file\_attach\_name> name. *[More information on Sending Files »](#)*

**thumbnail:** `InputFile | str | None`

*Optional.* Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can't be reused and can be only uploaded as a new file, so you can pass 'attach://<file\_attach\_name>' if the thumbnail was uploaded using multipart/form-data under <file\_attach\_name>. *[More information on Sending Files »](#)*

**caption:** `str | None`

*Optional.* Caption of the video to be sent, 0-1024 characters after entities parsing

**parse\_mode:** `str | None`

*Optional.* Mode for parsing entities in the video caption. See [formatting options](#) for more details.

**caption\_entities:** `List[MessageEntity] | None`

*Optional.* List of special entities that appear in the caption, which can be specified instead of *parse\_mode*

**width:** `int | None`

*Optional.* Video width

**height:** `int | None`

*Optional.* Video height

**duration:** `int | None`

*Optional.* Video duration in seconds

**supports\_streaming:** `bool | None`

*Optional.* Pass True if the uploaded video is suitable for streaming



**has\_spoiler:** `bool` | `None`

*Optional.* Pass `True` if the video needs to be covered with a spoiler animation

## KeyboardButton

```
class aiogram.types.keyboard_button.KeyboardButton(*, text: str, request_user:
    KeyboardButtonRequestUser | None = None,
    request_chat: KeyboardButtonRequestChat |
    None = None, request_contact: bool | None =
    None, request_location: bool | None = None,
    request_poll: KeyboardButtonPollType | None =
    None, web_app: WebAppInfo | None = None,
    **extra_data: Any)
```

This object represents one button of the reply keyboard. For simple text buttons, *String* can be used instead of this object to specify the button text. The optional fields *web\_app*, *request\_user*, *request\_chat*, *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 display *unsupported message*.

**Note:** *request\_poll* option will only work in Telegram versions released after 23 January, 2020. Older clients will display *unsupported message*.

**Note:** *web\_app* option will only work in Telegram versions released after 16 April, 2022. Older clients will display *unsupported message*.

**Note:** *request\_user* and *request\_chat* options will only work in Telegram versions released after 3 February, 2023. Older clients will display *unsupported message*.

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

**text:** `str`

Text of the button. If none of the optional fields are used, it will be sent as a message when the button is pressed

**request\_user:** `KeyboardButtonRequestUser` | `None`

*Optional.* If specified, pressing the button will open a list of suitable users. Tapping on any user will send their identifier to the bot in a 'user\_shared' service message. Available in private chats only.

**request\_chat:** `KeyboardButtonRequestChat` | `None`

*Optional.* If specified, pressing the button will open a list of suitable chats. Tapping on a chat will send its identifier to the bot in a 'chat\_shared' service message. Available in private chats only.

**request\_contact:** `bool` | `None`

*Optional.* If `True`, the user's phone number will be sent as a contact when the button is pressed. Available in private chats only.

**request\_location:** `bool` | `None`

*Optional.* If `True`, the user's current location will be sent when the button is pressed. Available in private chats only.

**request\_poll:** `KeyboardButtonPollType` | `None`

*Optional.* If specified, the user will be asked to create a poll and send it to the bot when the button is pressed. Available in private chats only.

**web\_app:** `WebAppInfo` | `None`

*Optional.* If specified, the described *Web App* will be launched when the button is pressed. The Web App will be able to send a 'web\_app\_data' service message. Available in private chats only.

## KeyboardButtonPollType

```
class aiogram.types.keyboard_button_poll_type.KeyboardButtonPollType(*, type: str | None = None,
                                                                    **extra_data: Any)
```

This object represents type of a poll, which is allowed to be created and sent when the corresponding button is pressed.

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

**type:** `str | None`

*Optional.* If *quiz* is passed, the user will be allowed to create only polls in the quiz mode. If *regular* is passed, only regular polls will be allowed. Otherwise, the user will be allowed to create a poll of any type.

## KeyboardButtonRequestChat

```
class aiogram.types.keyboard_button_request_chat.KeyboardButtonRequestChat(*, request_id: int,
                                                                            chat_is_channel:
                                                                            bool,
                                                                            chat_is_forum:
                                                                            bool | None =
                                                                            None,
                                                                            chat_has_username:
                                                                            bool | None =
                                                                            None,
                                                                            chat_is_created:
                                                                            bool | None =
                                                                            None,
                                                                            user_administrator_rights:
                                                                            ChatAdministra-
                                                                            torRights | None =
                                                                            None,
                                                                            bot_administrator_rights:
                                                                            ChatAdministra-
                                                                            torRights | None =
                                                                            None,
                                                                            bot_is_member:
                                                                            bool | None =
                                                                            None,
                                                                            **extra_data:
                                                                            Any)
```

This object defines the criteria used to request a suitable chat. The identifier of the selected chat will be shared with the bot when the corresponding button is pressed. [More about requesting chats](#) »

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

**request\_id:** `int`

Signed 32-bit identifier of the request, which will be received back in the `aiogram.types.chat_shared.ChatShared` object. Must be unique within the message

**chat\_is\_channel:** `bool`

Pass True to request a channel chat, pass False to request a group or a supergroup chat.

**chat\_is\_forum:** `bool` | `None`

*Optional.* Pass `True` to request a forum supergroup, pass `False` to request a non-forum chat. If not specified, no additional restrictions are applied.

**chat\_has\_username:** `bool` | `None`

*Optional.* Pass `True` to request a supergroup or a channel with a username, pass `False` to request a chat without a username. If not specified, no additional restrictions are applied.

**chat\_is\_created:** `bool` | `None`

*Optional.* Pass `True` to request a chat owned by the user. Otherwise, no additional restrictions are applied.

**user\_administrator\_rights:** `ChatAdministratorRights` | `None`

*Optional.* A JSON-serialized object listing the required administrator rights of the user in the chat. The rights must be a superset of `bot_administrator_rights`. If not specified, no additional restrictions are applied.

**bot\_administrator\_rights:** `ChatAdministratorRights` | `None`

*Optional.* A JSON-serialized object listing the required administrator rights of the bot in the chat. The rights must be a subset of `user_administrator_rights`. If not specified, no additional restrictions are applied.

**bot\_is\_member:** `bool` | `None`

*Optional.* Pass `True` to request a chat with the bot as a member. Otherwise, no additional restrictions are applied.

## KeyboardButtonRequestUser

```
class aiogram.types.keyboard_button_request_user.KeyboardButtonRequestUser(*, request_id: int,
                                                                           user_is_bot: bool |
                                                                           None = None,
                                                                           user_is_premium:
                                                                           bool | None =
                                                                           None,
                                                                           **extra_data:
                                                                           Any)
```

This object defines the criteria used to request a suitable user. The identifier of the selected user will be shared with the bot when the corresponding button is pressed. [More about requesting users](#) »

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

**request\_id:** `int`

Signed 32-bit identifier of the request, which will be received back in the `aiogram.types.user_shared.UserShared` object. Must be unique within the message

**user\_is\_bot:** `bool` | `None`

*Optional.* Pass `True` to request a bot, pass `False` to request a regular user. If not specified, no additional restrictions are applied.

**user\_is\_premium:** `bool` | `None`

*Optional.* Pass `True` to request a premium user, pass `False` to request a non-premium user. If not specified, no additional restrictions are applied.

## Location

```
class aiogram.types.location.Location(*, longitude: float, latitude: float, horizontal_accuracy: float |
    None = None, live_period: int | None = None, heading: int | None
    = None, proximity_alert_radius: int | None = None, **extra_data:
    Any)
```

This object represents a point on the map.

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

**longitude: float**

Longitude as defined by sender

**latitude: float**

Latitude as defined by sender

**horizontal\_accuracy: float | None**

*Optional.* The radius of uncertainty for the location, measured in meters; 0-1500

**live\_period: int | None**

*Optional.* Time relative to the message sending date, during which the location can be updated; in seconds. For active live locations only.

**heading: int | None**

*Optional.* The direction in which user is moving, in degrees; 1-360. For active live locations only.

**proximity\_alert\_radius: int | None**

*Optional.* The maximum distance for proximity alerts about approaching another chat member, in meters. For sent live locations only.

## LoginUrl

```
class aiogram.types.login_url.LoginUrl(*, url: str, forward_text: str | None = None, bot_username: str |
    None = None, request_write_access: bool | None = None,
    **extra_data: Any)
```

This object represents a parameter of the inline keyboard button used to automatically authorize a user. Serves as a great replacement for the [Telegram Login Widget](#) when the user is coming from Telegram. All the user needs to do is tap/click a button and confirm that they want to log in: Telegram apps support these buttons as of [version 5.7](#).

Sample bot: [@discussbot](#)

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

**url: str**

An HTTPS URL to be opened with user authorization data added to the query string when the button is pressed. If the user refuses to provide authorization data, the original URL without information about the user will be opened. The data added is the same as described in [Receiving authorization data](#).

**forward\_text: str | None**

*Optional.* New text of the button in forwarded messages.

**bot\_username: str | None**

*Optional.* Username of a bot, which will be used for user authorization. See [Setting up a bot](#) for more details. If not specified, the current bot's username will be assumed. The *url*'s domain must be the same as the domain linked with the bot. See [Linking your domain to the bot](#) for more details.

**request\_write\_access:** `bool` | `None`

*Optional.* Pass `True` to request the permission for your bot to send messages to the user.

## MenuButton

```
class aiogram.types.menu_button.MenuButton(*, type: str, text: str | None = None, web_app: WebAppInfo
                                           | None = None, **extra_data: Any)
```

This object describes the bot's menu button in a private chat. It should be one of

- `aiogram.types.menu_button_commands.MenuButtonCommands`
- `aiogram.types.menu_button_web_app.MenuButtonWebApp`
- `aiogram.types.menu_button_default.MenuButtonDefault`

If a menu button other than `aiogram.types.menu_button_default.MenuButtonDefault` is set for a private chat, then it is applied in the chat. Otherwise the default menu button is applied. By default, the menu button opens the list of bot commands.

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

**type:** `str`

Type of the button

**text:** `str` | `None`

*Optional.* Text on the button

**web\_app:** `WebAppInfo` | `None`

*Optional.* Description of the Web App that will be launched when the user presses the button. The Web App will be able to send an arbitrary message on behalf of the user using the method `aiogram.methods.answer_web_app_query.AnswerWebAppQuery`.

## MenuButtonCommands

```
class aiogram.types.menu_button_commands.MenuButtonCommands(*, type: ~typing.Literal[<MenuButtonType.COMMANDS:
                                                                                   'commands'>] =
                                                                                   MenuButtonType.COMMANDS, text:
                                                                                   str | None = None, web_app:
                                                                                   ~aiogram.types.web_app_info.WebAppInfo
                                                                                   | None = None, **extra_data:
                                                                                   ~typing.Any)
```

Represents a menu button, which opens the bot's list of commands.

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

**type:** `Literal[MenuButtonType.COMMANDS]`

Type of the button, must be *commands*

## MenuButtonDefault

```
class aiogram.types.menu_button_default.MenuButtonDefault(*, type: ~typing.Literal[<BotCommandScopeType.DEFAULT: 'default'>] = MenuButtonType.DEFAULT, text: str | None = None, web_app: ~aiogram.types.web_app_info.WebAppInfo | None = None, **extra_data: ~typing.Any)
```

Describes that no specific value for the menu button was set.

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

**type:** `Literal[MenuButtonType.DEFAULT]`

Type of the button, must be *default*

## MenuButtonWebApp

```
class aiogram.types.menu_button_web_app.MenuButtonWebApp(*, type: ~typing.Literal[<MenuButtonType.WEB_APP: 'web_app'>] = MenuButtonType.WEB_APP, text: str, web_app: ~aiogram.types.web_app_info.WebAppInfo, **extra_data: ~typing.Any)
```

Represents a menu button, which launches a [Web App](#).

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

**type:** `Literal[MenuButtonType.WEB_APP]`

Type of the button, must be *web\_app*

**text:** `str`

Text on the button

**web\_app:** [WebAppInfo](#)

Description of the Web App that will be launched when the user presses the button. The Web App will be able to send an arbitrary message on behalf of the user using the method [aiogram.methods.answer\\_web\\_app\\_query.AnswerWebAppQuery](#).

## Message

```

class aiogram.types.message.Message(*, message_id: int, date: datetime[datetime], chat: Chat,
    message_thread_id: int | None = None, from_user: User | None =
    None, sender_chat: Chat | None = None, forward_from: User | None
    = None, forward_from_chat: Chat | None = None,
    forward_from_message_id: int | None = None, forward_signature: str
    | None = None, forward_sender_name: str | None = None,
    forward_date: datetime[datetime] | None = None, is_topic_message:
    bool | None = None, is_automatic_forward: bool | None = None,
    reply_to_message: Message | None = None, via_bot: User | None =
    None, edit_date: int | None = None, has_protected_content: bool |
    None = None, media_group_id: str | None = None, author_signature:
    str | None = None, text: str | None = None, entities:
    List[MessageEntity] | None = None, animation: Animation | None =
    None, audio: Audio | None = None, document: Document | None =
    None, photo: List[PhotoSize] | None = None, sticker: Sticker | None
    = None, story: Story | None = None, video: Video | None = None,
    video_note: VideoNote | None = None, voice: Voice | None = None,
    caption: str | None = None, caption_entities: List[MessageEntity] |
    None = None, has_media_spoiler: bool | None = None, contact:
    Contact | None = None, dice: Dice | None = None, game: Game |
    None = None, poll: Poll | None = None, venue: Venue | None = None,
    location: Location | None = None, new_chat_members: List[User] |
    None = None, left_chat_member: User | None = None,
    new_chat_title: str | None = None, new_chat_photo: List[PhotoSize]
    | None = None, delete_chat_photo: bool | None = None,
    group_chat_created: bool | None = None, supergroup_chat_created:
    bool | None = None, channel_chat_created: bool | None = None,
    message_auto_delete_timer_changed:
    MessageAutoDeleteTimerChanged | None = None,
    migrate_to_chat_id: int | None = None, migrate_from_chat_id: int |
    None = None, pinned_message: Message | None = None, invoice:
    Invoice | None = None, successful_payment: SuccessfulPayment |
    None = None, user_shared: UserShared | None = None, chat_shared:
    ChatShared | None = None, connected_website: str | None = None,
    write_access_allowed: WriteAccessAllowed | None = None,
    passport_data: PassportData | None = None,
    proximity_alert_triggered: ProximityAlertTriggered | None = None,
    forum_topic_created: ForumTopicCreated | None = None,
    forum_topic_edited: ForumTopicEdited | None = None,
    forum_topic_closed: ForumTopicClosed | None = None,
    forum_topic_reopened: ForumTopicReopened | None = None,
    general_forum_topic_hidden: GeneralForumTopicHidden | None =
    None, general_forum_topic_unhidden: GeneralForumTopicUnhidden
    | None = None, video_chat_scheduled: VideoChatScheduled | None =
    None, video_chat_started: VideoChatStarted | None = None,
    video_chat_ended: VideoChatEnded | None = None,
    video_chat_participants_invited: VideoChatParticipantsInvited |
    None = None, web_app_data: WebAppData | None = None,
    reply_markup: InlineKeyboardMarkup | None = None, **extra_data:
    Any)

```

This object represents a message.

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

**message\_id:** int

Unique message identifier inside this chat

**date:** `DateTime`

Date the message was sent in Unix time

**chat:** `Chat`

Conversation the message belongs to

**message\_thread\_id:** `int | None`

*Optional.* Unique identifier of a message thread to which the message belongs; for supergroups only

**from\_user:** `User | None`

*Optional.* Sender of the message; empty for messages sent to channels. For backward compatibility, the field contains a fake sender user in non-channel chats, if the message was sent on behalf of a chat.

**sender\_chat:** `Chat | None`

*Optional.* Sender of the message, sent on behalf of a chat. For example, the channel itself for channel posts, the supergroup itself for messages from anonymous group administrators, the linked channel for messages automatically forwarded to the discussion group. For backward compatibility, the field *from* contains a fake sender user in non-channel chats, if the message was sent on behalf of a chat.

**forward\_from:** `User | None`

*Optional.* For forwarded messages, sender of the original message

**forward\_from\_chat:** `Chat | None`

*Optional.* For messages forwarded from channels or from anonymous administrators, information about the original sender chat

**forward\_from\_message\_id:** `int | None`

*Optional.* For messages forwarded from channels, identifier of the original message in the channel

**forward\_signature:** `str | None`

*Optional.* For forwarded messages that were originally sent in channels or by an anonymous chat administrator, signature of the message sender if present

**forward\_sender\_name:** `str | None`

*Optional.* Sender's name for messages forwarded from users who disallow adding a link to their account in forwarded messages

**forward\_date:** `DateTime | None`

*Optional.* For forwarded messages, date the original message was sent in Unix time

**is\_topic\_message:** `bool | None`

*Optional.* True, if the message is sent to a forum topic

**is\_automatic\_forward:** `bool | None`

*Optional.* True, if the message is a channel post that was automatically forwarded to the connected discussion group

**reply\_to\_message:** `Message | None`

*Optional.* For replies, the original message. Note that the Message object in this field will not contain further *reply\_to\_message* fields even if it itself is a reply.

**via\_bot:** `User | None`

*Optional.* Bot through which the message was sent



**edit\_date:** `int` | `None`

*Optional.* Date the message was last edited in Unix time

**has\_protected\_content:** `bool` | `None`

*Optional.* True, if the message can't be forwarded

**media\_group\_id:** `str` | `None`

*Optional.* The unique identifier of a media message group this message belongs to

**author\_signature:** `str` | `None`

*Optional.* Signature of the post author for messages in channels, or the custom title of an anonymous group administrator

**text:** `str` | `None`

*Optional.* For text messages, the actual UTF-8 text of the message

**entities:** `List[MessageEntity]` | `None`

*Optional.* For text messages, special entities like usernames, URLs, bot commands, etc. that appear in the text

**animation:** `Animation` | `None`

*Optional.* Message is an animation, information about the animation. For backward compatibility, when this field is set, the *document* field will also be set

**audio:** `Audio` | `None`

*Optional.* Message is an audio file, information about the file

**document:** `Document` | `None`

*Optional.* Message is a general file, information about the file

**photo:** `List[PhotoSize]` | `None`

*Optional.* Message is a photo, available sizes of the photo

**sticker:** `Sticker` | `None`

*Optional.* Message is a sticker, information about the sticker

**story:** `Story` | `None`

*Optional.* Message is a forwarded story

**video:** `Video` | `None`

*Optional.* Message is a video, information about the video

**video\_note:** `VideoNote` | `None`

*Optional.* Message is a *video note*, information about the video message

**voice:** `Voice` | `None`

*Optional.* Message is a voice message, information about the file

**caption:** `str` | `None`

*Optional.* Caption for the animation, audio, document, photo, video or voice

**caption\_entities:** `List[MessageEntity]` | `None`

*Optional.* For messages with a caption, special entities like usernames, URLs, bot commands, etc. that appear in the caption

**has\_media\_spoiler:** `bool` | `None`

*Optional.* True, if the message media is covered by a spoiler animation

**contact:** [Contact](#) | None

*Optional.* Message is a shared contact, information about the contact

**dice:** [Dice](#) | None

*Optional.* Message is a dice with random value

**game:** [Game](#) | None

*Optional.* Message is a game, information about the game. [More about games »](#)

**poll:** [Poll](#) | None

*Optional.* Message is a native poll, information about the poll

**venue:** [Venue](#) | None

*Optional.* Message is a venue, information about the venue. For backward compatibility, when this field is set, the *location* field will also be set

**location:** [Location](#) | None

*Optional.* Message is a shared location, information about the location

**new\_chat\_members:** [List\[User\]](#) | None

*Optional.* New members that were added to the group or supergroup and information about them (the bot itself may be one of these members)

**left\_chat\_member:** [User](#) | None

*Optional.* A member was removed from the group, information about them (this member may be the bot itself)

**new\_chat\_title:** [str](#) | None

*Optional.* A chat title was changed to this value

**new\_chat\_photo:** [List\[PhotoSize\]](#) | None

*Optional.* A chat photo was change to this value

**delete\_chat\_photo:** [bool](#) | None

*Optional.* Service message: the chat photo was deleted

**group\_chat\_created:** [bool](#) | None

*Optional.* Service message: the group has been created

**supergroup\_chat\_created:** [bool](#) | None

*Optional.* Service message: the supergroup has been created. This field can't be received in a message coming through updates, because bot can't be a member of a supergroup when it is created. It can only be found in *reply\_to\_message* if someone replies to a very first message in a directly created supergroup.

**channel\_chat\_created:** [bool](#) | None

*Optional.* Service message: the channel has been created. This field can't be received in a message coming through updates, because bot can't be a member of a channel when it is created. It can only be found in *reply\_to\_message* if someone replies to a very first message in a channel.

**message\_auto\_delete\_timer\_changed:** [MessageAutoDeleteTimerChanged](#) | None

*Optional.* Service message: auto-delete timer settings changed in the chat

**migrate\_to\_chat\_id:** [int](#) | None

*Optional.* The group has been migrated to a supergroup with the specified identifier. This number may have more than 32 significant bits and some programming languages may have difficulty/silent defects in interpreting it. But it has at most 52 significant bits, so a signed 64-bit integer or double-precision float type are safe for storing this identifier.

**migrate\_from\_chat\_id:** `int` | `None`

*Optional.* The supergroup has been migrated from a group with the specified identifier. This number may have more than 32 significant bits and some programming languages may have difficulty/silent defects in interpreting it. But it has at most 52 significant bits, so a signed 64-bit integer or double-precision float type are safe for storing this identifier.

**pinned\_message:** `Message` | `None`

*Optional.* Specified message was pinned. Note that the `Message` object in this field will not contain further `reply_to_message` fields even if it is itself a reply.

**invoice:** `Invoice` | `None`

*Optional.* Message is an invoice for a [payment](#), information about the invoice. [More about payments »](#)

**successful\_payment:** `SuccessfulPayment` | `None`

*Optional.* Message is a service message about a successful payment, information about the payment. [More about payments »](#)

**user\_shared:** `UserShared` | `None`

*Optional.* Service message: a user was shared with the bot

**chat\_shared:** `ChatShared` | `None`

*Optional.* Service message: a chat was shared with the bot

**connected\_website:** `str` | `None`

*Optional.* The domain name of the website on which the user has logged in. [More about Telegram Login »](#)

**write\_access\_allowed:** `WriteAccessAllowed` | `None`

*Optional.* Service message: the user allowed the bot added to the attachment menu to write messages

**passport\_data:** `PassportData` | `None`

*Optional.* Telegram Passport data

**proximity\_alert\_triggered:** `ProximityAlertTriggered` | `None`

*Optional.* Service message. A user in the chat triggered another user's proximity alert while sharing Live Location.

**forum\_topic\_created:** `ForumTopicCreated` | `None`

*Optional.* Service message: forum topic created

**forum\_topic\_edited:** `ForumTopicEdited` | `None`

*Optional.* Service message: forum topic edited

**forum\_topic\_closed:** `ForumTopicClosed` | `None`

*Optional.* Service message: forum topic closed

**forum\_topic\_reopened:** `ForumTopicReopened` | `None`

*Optional.* Service message: forum topic reopened

**general\_forum\_topic\_hidden:** `GeneralForumTopicHidden` | `None`

*Optional.* Service message: the 'General' forum topic hidden

**general\_forum\_topic\_unhidden:** `GeneralForumTopicUnhidden` | `None`

*Optional.* Service message: the 'General' forum topic unhidden

**video\_chat\_scheduled:** `VideoChatScheduled` | `None`

*Optional.* Service message: video chat scheduled

**video\_chat\_started:** [VideoChatStarted](#) | None

*Optional.* Service message: video chat started

**video\_chat\_ended:** [VideoChatEnded](#) | None

*Optional.* Service message: video chat ended

**video\_chat\_participants\_invited:** [VideoChatParticipantsInvited](#) | None

*Optional.* Service message: new participants invited to a video chat

**web\_app\_data:** [WebAppData](#) | None

*Optional.* Service message: data sent by a Web App

**reply\_markup:** [InlineKeyboardMarkup](#) | None

*Optional.* Inline keyboard attached to the message. `login_url` buttons are represented as ordinary url buttons.

**property content\_type:** str

**property html\_text:** str

**property md\_text:** str

**reply\_animation**(*animation:* [InputFile](#) | str, *duration:* int | None = None, *width:* int | None = None, *height:* int | None = None, *thumbnail:* [InputFile](#) | str | None = None, *caption:* str | None = None, *parse\_mode:* str | None = sentinel.UNSET\_PARSE\_MODE, *caption\_entities:* List[[MessageEntity](#)] | None = None, *has\_spoiler:* bool | None = None, *disable\_notification:* bool | None = None, *protect\_content:* bool | None = sentinel.UNSET\_PROTECT\_CONTENT, *allow\_sending\_without\_reply:* bool | None = None, *reply\_markup:* [InlineKeyboardMarkup](#) | [ReplyKeyboardMarkup](#) | [ReplyKeyboardRemove](#) | [ForceReply](#) | None = None, *\*\*kwargs:* Any) → [SendAnimation](#)

Shortcut for method [aiogram.methods.send\\_animation.SendAnimation](#) will automatically fill method attributes:

- `chat_id`
- `message_thread_id`
- `reply_to_message_id`

Use this method to send animation files (GIF or H.264/MPEG-4 AVC video without sound). On success, the sent [aiogram.types.message.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** – 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. [More information on Sending Files](#) »
- **duration** – Duration of sent animation in seconds
- **width** – Animation width
- **height** – Animation height
- **thumbnail** – Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320. Ignored if the file is not

uploaded using multipart/form-data. Thumbnails can't be reused and can be only uploaded as a new file, so you can pass 'attach://<file\_attach\_name>' if the thumbnail was uploaded using multipart/form-data under <file\_attach\_name>. [More information on Sending Files](#) »

- **caption** – Animation caption (may also be used when resending animation by *file\_id*), 0-1024 characters after entities parsing
- **parse\_mode** – Mode for parsing entities in the animation caption. See [formatting options](#) for more details.
- **caption\_entities** – A JSON-serialized list of special entities that appear in the caption, which can be specified instead of *parse\_mode*
- **has\_spoiler** – Pass True if the animation needs to be covered with a spoiler animation
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method [aiogram.methods.send\\_animation.SendAnimation](#)

**answer\_animation**(*animation*: [InputFile](#) | *str*, *duration*: *int* | *None* = *None*, *width*: *int* | *None* = *None*, *height*: *int* | *None* = *None*, *thumbnail*: [InputFile](#) | *str* | *None* = *None*, *caption*: *str* | *None* = *None*, *parse\_mode*: *str* | *None* = *sentinel.UNSET\_PARSE\_MODE*, *caption\_entities*: *List*[[MessageEntity](#)] | *None* = *None*, *has\_spoiler*: *bool* | *None* = *None*, *disable\_notification*: *bool* | *None* = *None*, *protect\_content*: *bool* | *None* = *sentinel.UNSET\_PROTECT\_CONTENT*, *reply\_to\_message\_id*: *int* | *None* = *None*, *allow\_sending\_without\_reply*: *bool* | *None* = *None*, *reply\_markup*: [InlineKeyboardMarkup](#) | [ReplyKeyboardMarkup](#) | [ReplyKeyboardRemove](#) | [ForceReply](#) | *None* = *None*, *\*\*kwargs*: *Any*) → [SendAnimation](#)

Shortcut for method [aiogram.methods.send\\_animation.SendAnimation](#) will automatically fill method attributes:

- `chat_id`
- `message_thread_id`

Use this method to send animation files (GIF or H.264/MPEG-4 AVC video without sound). On success, the sent [aiogram.types.message.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** – 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. [More information on Sending Files](#) »
- **duration** – Duration of sent animation in seconds
- **width** – Animation width

- **height** – Animation height
- **thumbnail** – Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can't be reused and can be only uploaded as a new file, so you can pass 'attach://<file\_attach\_name>' if the thumbnail was uploaded using multipart/form-data under <file\_attach\_name>. [More information on Sending Files](#) »
- **caption** – Animation caption (may also be used when resending animation by *file\_id*), 0-1024 characters after entities parsing
- **parse\_mode** – Mode for parsing entities in the animation caption. See [formatting options](#) for more details.
- **caption\_entities** – A JSON-serialized list of special entities that appear in the caption, which can be specified instead of *parse\_mode*
- **has\_spoiler** – Pass True if the animation needs to be covered with a spoiler animation
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method [aiogram.methods.send\\_animation.SendAnimation](#)

**reply\_audio**(*audio*: [InputFile](#) | *str*, *caption*: *str* | *None* = *None*, *parse\_mode*: *str* | *None* = *sentinel.UNSET\_PARSE\_MODE*, *caption\_entities*: *List*[[MessageEntity](#)] | *None* = *None*, *duration*: *int* | *None* = *None*, *performer*: *str* | *None* = *None*, *title*: *str* | *None* = *None*, *thumbnail*: [InputFile](#) | *str* | *None* = *None*, *disable\_notification*: *bool* | *None* = *None*, *protect\_content*: *bool* | *None* = *sentinel.UNSET\_PROTECT\_CONTENT*, *allow\_sending\_without\_reply*: *bool* | *None* = *None*, *reply\_markup*: [InlineKeyboardMarkup](#) | [ReplyKeyboardMarkup](#) | [ReplyKeyboardRemove](#) | [ForceReply](#) | *None* = *None*, *\*\*kwargs*: *Any*) → [SendAudio](#)

Shortcut for method [aiogram.methods.send\\_audio.SendAudio](#) will automatically fill method attributes:

- *chat\_id*
- *message\_thread\_id*
- *reply\_to\_message\_id*

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 or .M4A format. On success, the sent [aiogram.types.message.Message](#) is returned. Bots can currently send audio files of up to 50 MB in size, this limit may be changed in the future. For sending voice messages, use the [aiogram.methods.send\\_voice.SendVoice](#) method instead.

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

#### Parameters

- **audio** – Audio file to send. Pass a `file_id` as `String` to send an audio file that exists on the Telegram servers (recommended), pass an HTTP URL as a `String` for Telegram to get an audio file from the Internet, or upload a new one using multipart/form-data. [More information on Sending Files](#) »
- **caption** – Audio caption, 0-1024 characters after entities parsing
- **parse\_mode** – Mode for parsing entities in the audio caption. See [formatting options](#) for more details.
- **caption\_entities** – A JSON-serialized list of special entities that appear in the caption, which can be specified instead of `parse_mode`
- **duration** – Duration of the audio in seconds
- **performer** – Performer
- **title** – Track name
- **thumbnail** – Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can't be reused and can be only uploaded as a new file, so you can pass 'attach://<file\_attach\_name>' if the thumbnail was uploaded using multipart/form-data under <file\_attach\_name>. [More information on Sending Files](#) »
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **allow\_sending\_without\_reply** – Pass `True` if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method `aiogram.methods.send_audio.SendAudio`

```
answer_audio(audio: InputFile | str, caption: str | None = None, parse_mode: str | None =
    sentinel.UNSET_PARSE_MODE, caption_entities: List[MessageEntity] | None = None,
    duration: int | None = None, performer: str | None = None, title: str | None = None,
    thumbnail: InputFile | str | None = None, disable_notification: bool | None = None,
    protect_content: bool | None = sentinel.UNSET_PROTECT_CONTENT,
    reply_to_message_id: int | None = None, allow_sending_without_reply: bool | None = None,
    reply_markup: InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove |
    ForceReply | None = None, **kwargs: Any) → SendAudio
```

Shortcut for method `aiogram.methods.send_audio.SendAudio` will automatically fill method attributes:

- `chat_id`
- `message_thread_id`

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 or .M4A format. On success, the sent `aiogram.types.message.Message` is returned. Bots can currently send audio files of up to 50 MB in size, this limit may be changed in the future. For sending voice messages, use the `aiogram.methods.send_voice.SendVoice` method instead.



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

### Parameters

- **audio** – Audio file to send. Pass a `file_id` as `String` to send an audio file that exists on the Telegram servers (recommended), pass an `HTTP URL` as a `String` for Telegram to get an audio file from the Internet, or upload a new one using `multipart/form-data`. [More information on Sending Files](#) »
- **caption** – Audio caption, 0-1024 characters after entities parsing
- **parse\_mode** – Mode for parsing entities in the audio caption. See [formatting options](#) for more details.
- **caption\_entities** – A JSON-serialized list of special entities that appear in the caption, which can be specified instead of `parse_mode`
- **duration** – Duration of the audio in seconds
- **performer** – Performer
- **title** – Track name
- **thumbnail** – Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in `JPEG` format and less than 200 kB in size. A thumbnail's width and height should not exceed 320. Ignored if the file is not uploaded using `multipart/form-data`. Thumbnails can't be reused and can be only uploaded as a new file, so you can pass `'attach://<file_attach_name>'` if the thumbnail was uploaded using `multipart/form-data` under `<file_attach_name>`. [More information on Sending Files](#) »
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass `True` if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method [aiogram.methods.send\\_audio.SendAudio](#)

**reply\_contact**(*phone\_number: str, first\_name: str, last\_name: str | None = None, vcard: str | None = None, disable\_notification: bool | None = None, protect\_content: bool | None = sentinel.UNSET\_PROTECT\_CONTENT, allow\_sending\_without\_reply: bool | None = None, reply\_markup: InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply | None = None, \*\*kwargs: Any*) → [SendContact](#)

Shortcut for method [aiogram.methods.send\\_contact.SendContact](#) will automatically fill method attributes:

- `chat_id`
- `message_thread_id`
- `reply_to_message_id`



Use this method to send phone contacts. On success, the sent `aiogram.types.message.Message` is returned.

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

#### Parameters

- **phone\_number** – Contact’s phone number
- **first\_name** – Contact’s first name
- **last\_name** – Contact’s last name
- **vcard** – Additional data about the contact in the form of a `vCard`, 0-2048 bytes
- **disable\_notification** – Sends the message `silently`. Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **allow\_sending\_without\_reply** – Pass `True` if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method `aiogram.methods.send_contact.SendContact`

```
answer_contact(phone_number: str, first_name: str, last_name: str | None = None, vcard: str | None = None, disable_notification: bool | None = None, protect_content: bool | None = sentinel.UNSET_PROTECT_CONTENT, reply_to_message_id: int | None = None, allow_sending_without_reply: bool | None = None, reply_markup: InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply | None = None, **kwargs: Any) → SendContact
```

Shortcut for method `aiogram.methods.send_contact.SendContact` will automatically fill method attributes:

- `chat_id`
- `message_thread_id`

Use this method to send phone contacts. On success, the sent `aiogram.types.message.Message` is returned.

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

#### Parameters

- **phone\_number** – Contact’s phone number
- **first\_name** – Contact’s first name
- **last\_name** – Contact’s last name
- **vcard** – Additional data about the contact in the form of a `vCard`, 0-2048 bytes
- **disable\_notification** – Sends the message `silently`. Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message

- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method `aiogram.methods.send_contact.SendContact`

**reply\_document**(*document*: [InputFile](#) | *str*, *thumbnail*: [InputFile](#) | *str* | *None* = *None*, *caption*: *str* | *None* = *None*, *parse\_mode*: *str* | *None* = *sentinel.UNSET\_PARSE\_MODE*, *caption\_entities*: *List*[[MessageEntity](#)] | *None* = *None*, *disable\_content\_type\_detection*: *bool* | *None* = *None*, *disable\_notification*: *bool* | *None* = *None*, *protect\_content*: *bool* | *None* = *sentinel.UNSET\_PROTECT\_CONTENT*, *allow\_sending\_without\_reply*: *bool* | *None* = *None*, *reply\_markup*: [InlineKeyboardMarkup](#) | [ReplyKeyboardMarkup](#) | [ReplyKeyboardRemove](#) | [ForceReply](#) | *None* = *None*, *\*\*kwargs*: *Any*) → [SendDocument](#)

Shortcut for method `aiogram.methods.send_document.SendDocument` will automatically fill method attributes:

- `chat_id`
- `message_thread_id`
- `reply_to_message_id`

Use this method to send general files. On success, the sent `aiogram.types.message.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** – File to send. Pass a `file_id` as String to send a file that exists on the Telegram servers (recommended), 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 information on Sending Files](#) »
- **thumbnail** – Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can't be reused and can be only uploaded as a new file, so you can pass 'attach://<file\_attach\_name>' if the thumbnail was uploaded using multipart/form-data under <file\_attach\_name>. [More information on Sending Files](#) »
- **caption** – Document caption (may also be used when resending documents by `file_id`), 0-1024 characters after entities parsing
- **parse\_mode** – Mode for parsing entities in the document caption. See [formatting options](#) for more details.
- **caption\_entities** – A JSON-serialized list of special entities that appear in the caption, which can be specified instead of `parse_mode`
- **disable\_content\_type\_detection** – Disables automatic server-side content type detection for files uploaded using multipart/form-data
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.

- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method `aiogram.methods.send_document.SendDocument`

**answer\_document**(*document*: `InputFile` | `str`, *thumbnail*: `InputFile` | `str` | `None` = `None`, *caption*: `str` | `None` = `None`, *parse\_mode*: `str` | `None` = `sentinel.UNSET_PARSE_MODE`, *caption\_entities*: `List[MessageEntity]` | `None` = `None`, *disable\_content\_type\_detection*: `bool` | `None` = `None`, *disable\_notification*: `bool` | `None` = `None`, *protect\_content*: `bool` | `None` = `sentinel.UNSET_PROTECT_CONTENT`, *reply\_to\_message\_id*: `int` | `None` = `None`, *allow\_sending\_without\_reply*: `bool` | `None` = `None`, *reply\_markup*: `InlineKeyboardMarkup` | `ReplyKeyboardMarkup` | `ReplyKeyboardRemove` | `ForceReply` | `None` = `None`, *\*\*kwargs*: `Any`) → `SendDocument`

Shortcut for method `aiogram.methods.send_document.SendDocument` will automatically fill method attributes:

- `chat_id`
- `message_thread_id`

Use this method to send general files. On success, the sent `aiogram.types.message.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** – File to send. Pass a `file_id` as `String` to send a file that exists on the Telegram servers (recommended), 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 information on Sending Files](#) »
- **thumbnail** – Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can't be reused and can be only uploaded as a new file, so you can pass 'attach://<file\_attach\_name>' if the thumbnail was uploaded using multipart/form-data under <file\_attach\_name>. [More information on Sending Files](#) »
- **caption** – Document caption (may also be used when resending documents by `file_id`), 0-1024 characters after entities parsing
- **parse\_mode** – Mode for parsing entities in the document caption. See [formatting options](#) for more details.
- **caption\_entities** – A JSON-serialized list of special entities that appear in the caption, which can be specified instead of `parse_mode`
- **disable\_content\_type\_detection** – Disables automatic server-side content type detection for files uploaded using multipart/form-data
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.

- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method [aiogram.methods.send\\_document.SendDocument](#)

**reply\_game**(*game\_short\_name: str, disable\_notification: bool | None = None, protect\_content: bool | None = sentinel.UNSET\_PROTECT\_CONTENT, allow\_sending\_without\_reply: bool | None = None, reply\_markup: InlineKeyboardMarkup | None = None, \*\*kwargs: Any*) → [SendGame](#)

Shortcut for method [aiogram.methods.send\\_game.SendGame](#) will automatically fill method attributes:

- chat\_id
- message\_thread\_id
- reply\_to\_message\_id

Use this method to send a game. On success, the sent [aiogram.types.message.Message](#) is returned.

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

#### Parameters

- **game\_short\_name** – Short name of the game, serves as the unique identifier for the game. Set up your games via [@BotFather](#).
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method [aiogram.methods.send\\_game.SendGame](#)

**answer\_game**(*game\_short\_name: str, disable\_notification: bool | None = None, protect\_content: bool | None = sentinel.UNSET\_PROTECT\_CONTENT, reply\_to\_message\_id: int | None = None, allow\_sending\_without\_reply: bool | None = None, reply\_markup: InlineKeyboardMarkup | None = None, \*\*kwargs: Any*) → [SendGame](#)

Shortcut for method [aiogram.methods.send\\_game.SendGame](#) will automatically fill method attributes:

- chat\_id
- message\_thread\_id

Use this method to send a game. On success, the sent [aiogram.types.message.Message](#) is returned.

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

#### Parameters

- **game\_short\_name** – Short name of the game, serves as the unique identifier for the game. Set up your games via [@BotFather](#).
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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**

instance of method [aiogram.methods.send\\_game.SendGame](#)

**reply\_invoice**(title: str, description: str, payload: str, provider\_token: str, currency: str, prices: List[LabeledPrice], max\_tip\_amount: int | None = None, suggested\_tip\_amounts: List[int] | None = None, start\_parameter: str | None = None, provider\_data: str | None = None, photo\_url: str | None = None, photo\_size: int | None = None, photo\_width: int | None = None, photo\_height: int | None = None, need\_name: bool | None = None, need\_phone\_number: bool | None = None, need\_email: bool | None = None, need\_shipping\_address: bool | None = None, send\_phone\_number\_to\_provider: bool | None = None, send\_email\_to\_provider: bool | None = None, is\_flexible: bool | None = None, disable\_notification: bool | None = None, protect\_content: bool | None = sentinel.UNSET\_PROTECT\_CONTENT, allow\_sending\_without\_reply: bool | None = None, reply\_markup: InlineKeyboardMarkup | None = None, \*\*kwargs: Any) → [SendInvoice](#)

Shortcut for method [aiogram.methods.send\\_invoice.SendInvoice](#) will automatically fill method attributes:

- chat\_id
- message\_thread\_id
- reply\_to\_message\_id

Use this method to send invoices. On success, the sent [aiogram.types.message.Message](#) is returned.

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

**Parameters**

- **title** – Product name, 1-32 characters
- **description** – Product description, 1-255 characters
- **payload** – Bot-defined invoice payload, 1-128 bytes. This will not be displayed to the user, use for your internal processes.
- **provider\_token** – Payment provider token, obtained via [@BotFather](#)
- **currency** – Three-letter ISO 4217 currency code, see [more on currencies](#)
- **prices** – Price breakdown, a JSON-serialized list of components (e.g. product price, tax, discount, delivery cost, delivery tax, bonus, etc.)
- **max\_tip\_amount** – The maximum accepted amount for tips in the *smallest units* of the currency (integer, **not** float/double). For example, for a maximum tip of US\$ 1.45 pass max\_tip\_amount = 145. See the *exp* parameter in [currencies.json](#), it shows the number

of digits past the decimal point for each currency (2 for the majority of currencies). Defaults to 0

- **suggested\_tip\_amounts** – A JSON-serialized array of suggested amounts of tips in the *smallest units* of the currency (integer, **not** float/double). At most 4 suggested tip amounts can be specified. The suggested tip amounts must be positive, passed in a strictly increased order and must not exceed *max\_tip\_amount*.
- **start\_parameter** – 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** – JSON-serialized data about the invoice, which will be shared with the payment provider. A detailed description of required fields should be provided by the payment provider.
- **photo\_url** – URL of the product photo for the invoice. Can be a photo of the goods or a marketing image for a service. People like it better when they see what they are paying for.
- **photo\_size** – Photo size in bytes
- **photo\_width** – Photo width
- **photo\_height** – Photo height
- **need\_name** – Pass True if you require the user's full name to complete the order
- **need\_phone\_number** – Pass True if you require the user's phone number to complete the order
- **need\_email** – Pass True if you require the user's email address to complete the order
- **need\_shipping\_address** – Pass True if you require the user's shipping address to complete the order
- **send\_phone\_number\_to\_provider** – Pass True if the user's phone number should be sent to provider
- **send\_email\_to\_provider** – Pass True if the user's email address should be sent to provider
- **is\_flexible** – Pass True if the final price depends on the shipping method
- **disable\_notification** – Sends the message *silently*. Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method `aiogram.methods.send_invoice.SendInvoice`

```

answer_invoice(title: str, description: str, payload: str, provider_token: str, currency: str, prices:
    List[LabeledPrice], max_tip_amount: int | None = None, suggested_tip_amounts: List[int]
    | None = None, start_parameter: str | None = None, provider_data: str | None = None,
    photo_url: str | None = None, photo_size: int | None = None, photo_width: int | None =
    None, photo_height: int | None = None, need_name: bool | None = None,
    need_phone_number: bool | None = None, need_email: bool | None = None,
    need_shipping_address: bool | None = None, send_phone_number_to_provider: bool |
    None = None, send_email_to_provider: bool | None = None, is_flexible: bool | None =
    None, disable_notification: bool | None = None, protect_content: bool | None =
    sentinel.UNSET_PROTECT_CONTENT, reply_to_message_id: int | None = None,
    allow_sending_without_reply: bool | None = None, reply_markup: InlineKeyboardMarkup
    | None = None, **kwargs: Any) → SendInvoice

```

Shortcut for method `aiogram.methods.send_invoice.SendInvoice` will automatically fill method attributes:

- `chat_id`
- `message_thread_id`

Use this method to send invoices. On success, the sent `aiogram.types.message.Message` is returned.

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

#### Parameters

- **title** – Product name, 1-32 characters
- **description** – Product description, 1-255 characters
- **payload** – Bot-defined invoice payload, 1-128 bytes. This will not be displayed to the user, use for your internal processes.
- **provider\_token** – Payment provider token, obtained via `@BotFather`
- **currency** – Three-letter ISO 4217 currency code, see [more on currencies](#)
- **prices** – Price breakdown, a JSON-serialized list of components (e.g. product price, tax, discount, delivery cost, delivery tax, bonus, etc.)
- **max\_tip\_amount** – The maximum accepted amount for tips in the *smallest units* of the currency (integer, **not** float/double). For example, for a maximum tip of US\$ 1.45 pass `max_tip_amount = 145`. See the `exp` parameter in `currencies.json`, it shows the number of digits past the decimal point for each currency (2 for the majority of currencies). Defaults to 0
- **suggested\_tip\_amounts** – A JSON-serialized array of suggested amounts of tips in the *smallest units* of the currency (integer, **not** float/double). At most 4 suggested tip amounts can be specified. The suggested tip amounts must be positive, passed in a strictly increased order and must not exceed `max_tip_amount`.
- **start\_parameter** – 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** – JSON-serialized data about the invoice, which will be shared with the payment provider. A detailed description of required fields should be provided by the payment provider.
- **photo\_url** – URL of the product photo for the invoice. Can be a photo of the goods or a marketing image for a service. People like it better when they see what they are paying for.



- **photo\_size** – Photo size in bytes
- **photo\_width** – Photo width
- **photo\_height** – Photo height
- **need\_name** – Pass True if you require the user’s full name to complete the order
- **need\_phone\_number** – Pass True if you require the user’s phone number to complete the order
- **need\_email** – Pass True if you require the user’s email address to complete the order
- **need\_shipping\_address** – Pass True if you require the user’s shipping address to complete the order
- **send\_phone\_number\_to\_provider** – Pass True if the user’s phone number should be sent to provider
- **send\_email\_to\_provider** – Pass True if the user’s email address should be sent to provider
- **is\_flexible** – Pass True if the final price depends on the shipping method
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method [aiogram.methods.send\\_invoice.SendInvoice](#)

**reply\_location**(*latitude: float, longitude: float, horizontal\_accuracy: float | None = None, live\_period: int | None = None, heading: int | None = None, proximity\_alert\_radius: int | None = None, disable\_notification: bool | None = None, protect\_content: bool | None = sentinel.UNSET\_PROTECT\_CONTENT, allow\_sending\_without\_reply: bool | None = None, reply\_markup: [InlineKeyboardMarkup](#) | [ReplyKeyboardMarkup](#) | [ReplyKeyboardRemove](#) | [ForceReply](#) | None = None, \*\*kwargs: Any*) → [SendLocation](#)

Shortcut for method [aiogram.methods.send\\_location.SendLocation](#) will automatically fill method attributes:

- **chat\_id**
- **message\_thread\_id**
- **reply\_to\_message\_id**

Use this method to send point on the map. On success, the sent [aiogram.types.message.Message](#) is returned.

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

#### Parameters

- **latitude** – Latitude of the location
- **longitude** – Longitude of the location



- **horizontal\_accuracy** – The radius of uncertainty for the location, measured in meters; 0-1500
- **live\_period** – Period in seconds for which the location will be updated (see [Live Locations](#), should be between 60 and 86400).
- **heading** – For live locations, a direction in which the user is moving, in degrees. Must be between 1 and 360 if specified.
- **proximity\_alert\_radius** – 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** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method [aiogram.methods.send\\_location.SendLocation](#)

**answer\_location**(*latitude: float, longitude: float, horizontal\_accuracy: float | None = None, live\_period: int | None = None, heading: int | None = None, proximity\_alert\_radius: int | None = None, disable\_notification: bool | None = None, protect\_content: bool | None = sentinel.UNSET\_PROTECT\_CONTENT, reply\_to\_message\_id: int | None = None, allow\_sending\_without\_reply: bool | None = None, reply\_markup: [InlineKeyboardMarkup](#) | [ReplyKeyboardMarkup](#) | [ReplyKeyboardRemove](#) | [ForceReply](#) | None = None, \*\*kwargs: Any*) → [SendLocation](#)

Shortcut for method [aiogram.methods.send\\_location.SendLocation](#) will automatically fill method attributes:

- `chat_id`
- `message_thread_id`

Use this method to send point on the map. On success, the sent [aiogram.types.message.Message](#) is returned.

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

#### Parameters

- **latitude** – Latitude of the location
- **longitude** – Longitude of the location
- **horizontal\_accuracy** – The radius of uncertainty for the location, measured in meters; 0-1500
- **live\_period** – Period in seconds for which the location will be updated (see [Live Locations](#), should be between 60 and 86400).
- **heading** – For live locations, a direction in which the user is moving, in degrees. Must be between 1 and 360 if specified.

- **proximity\_alert\_radius** – 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** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method [aiogram.methods.send\\_location.SendLocation](#)

**reply\_media\_group**(*media: List[InputMediaAudio | InputMediaDocument | InputMediaPhoto | InputMediaVideo]*, *disable\_notification: bool | None = None*, *protect\_content: bool | None = sentinel.UNSET\_PROTECT\_CONTENT*, *allow\_sending\_without\_reply: bool | None = None*, *\*\*kwargs: Any*) → [SendMediaGroup](#)

Shortcut for method [aiogram.methods.send\\_media\\_group.SendMediaGroup](#) will automatically fill method attributes:

- `chat_id`
- `message_thread_id`
- `reply_to_message_id`

Use this method to send a group of photos, videos, documents or audios as an album. Documents and audio files can be only grouped 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** – A JSON-serialized array describing messages to be sent, must include 2-10 items
- **disable\_notification** – Sends messages [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent messages from forwarding and saving
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found

#### Returns

instance of method [aiogram.methods.send\\_media\\_group.SendMediaGroup](#)

**answer\_media\_group**(*media: List[InputMediaAudio | InputMediaDocument | InputMediaPhoto | InputMediaVideo]*, *disable\_notification: bool | None = None*, *protect\_content: bool | None = sentinel.UNSET\_PROTECT\_CONTENT*, *reply\_to\_message\_id: int | None = None*, *allow\_sending\_without\_reply: bool | None = None*, *\*\*kwargs: Any*) → [SendMediaGroup](#)

Shortcut for method [aiogram.methods.send\\_media\\_group.SendMediaGroup](#) will automatically fill method attributes:

- `chat_id`
- `message_thread_id`

Use this method to send a group of photos, videos, documents or audios as an album. Documents and audio files can be only grouped 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** – A JSON-serialized array describing messages to be sent, must include 2-10 items
- **disable\_notification** – Sends messages `silently`. Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent messages from forwarding and saving
- **reply\_to\_message\_id** – If the messages are a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass `True` if the message should be sent even if the specified replied-to message is not found

#### Returns

instance of method `aiogram.methods.send_media_group.SendMediaGroup`

```
reply(text: str, parse_mode: str | None = sentinel.UNSET_PARSE_MODE, entities: List[MessageEntity] | None = None, disable_web_page_preview: bool | None = sentinel.UNSET_DISABLE_WEB_PAGE_PREVIEW, disable_notification: bool | None = None, protect_content: bool | None = sentinel.UNSET_PROTECT_CONTENT, allow_sending_without_reply: bool | None = None, reply_markup: InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply | None = None, **kwargs: Any) → SendMessage
```

Shortcut for method `aiogram.methods.send_message.SendMessage` will automatically fill method attributes:

- `chat_id`
- `message_thread_id`
- `reply_to_message_id`

Use this method to send text messages. On success, the sent `aiogram.types.message.Message` is returned.

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

#### Parameters

- **text** – Text of the message to be sent, 1-4096 characters after entities parsing
- **parse\_mode** – Mode for parsing entities in the message text. See [formatting options](#) for more details.
- **entities** – A JSON-serialized list of special entities that appear in message text, which can be specified instead of `parse_mode`
- **disable\_web\_page\_preview** – Disables link previews for links in this message
- **disable\_notification** – Sends the message `silently`. Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving

- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method [aiogram.methods.send\\_message.SendMessage](#)

**answer**(*text: str, parse\_mode: str | None = sentinel.UNSET\_PARSE\_MODE, entities: List[MessageEntity] | None = None, disable\_web\_page\_preview: bool | None = sentinel.UNSET\_DISABLE\_WEB\_PAGE\_PREVIEW, disable\_notification: bool | None = None, protect\_content: bool | None = sentinel.UNSET\_PROTECT\_CONTENT, reply\_to\_message\_id: int | None = None, allow\_sending\_without\_reply: bool | None = None, reply\_markup: InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply | None = None, \*\*kwargs: Any*) → [SendMessage](#)

Shortcut for method [aiogram.methods.send\\_message.SendMessage](#) will automatically fill method attributes:

- `chat_id`
- `message_thread_id`

Use this method to send text messages. On success, the sent [aiogram.types.message.Message](#) is returned.

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

#### Parameters

- **text** – Text of the message to be sent, 1-4096 characters after entities parsing
- **parse\_mode** – Mode for parsing entities in the message text. See [formatting options](#) for more details.
- **entities** – A JSON-serialized list of special entities that appear in message text, which can be specified instead of *parse\_mode*
- **disable\_web\_page\_preview** – Disables link previews for links in this message
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method [aiogram.methods.send\\_message.SendMessage](#)

**reply\_photo**(*photo: InputFile | str, caption: str | None = None, parse\_mode: str | None = sentinel.UNSET\_PARSE\_MODE, caption\_entities: List[MessageEntity] | None = None, has\_spoiler: bool | None = None, disable\_notification: bool | None = None, protect\_content: bool | None = sentinel.UNSET\_PROTECT\_CONTENT, allow\_sending\_without\_reply: bool | None = None, reply\_markup: InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply | None = None, \*\*kwargs: Any*) → [SendPhoto](#)

Shortcut for method `aiogram.methods.send_photo.SendPhoto` will automatically fill method attributes:

- `chat_id`
- `message_thread_id`
- `reply_to_message_id`

Use this method to send photos. On success, the sent `aiogram.types.message.Message` is returned.

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

#### Parameters

- **photo** – Photo to send. Pass a `file_id` as String to send a photo that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get a photo from the Internet, or upload a new photo using multipart/form-data. The photo must be at most 10 MB in size. The photo's width and height must not exceed 10000 in total. Width and height ratio must be at most 20. *More information on Sending Files »*
- **caption** – Photo caption (may also be used when resending photos by `file_id`), 0-1024 characters after entities parsing
- **parse\_mode** – Mode for parsing entities in the photo caption. See [formatting options](#) for more details.
- **caption\_entities** – A JSON-serialized list of special entities that appear in the caption, which can be specified instead of `parse_mode`
- **has\_spoiler** – Pass True if the photo needs to be covered with a spoiler animation
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method `aiogram.methods.send_photo.SendPhoto`

```
answer_photo(photo: InputFile | str, caption: str | None = None, parse_mode: str | None =
    sentinel.UNSET_PARSE_MODE, caption_entities: List[MessageEntity] | None = None,
    has_spoiler: bool | None = None, disable_notification: bool | None = None, protect_content:
    bool | None = sentinel.UNSET_PROTECT_CONTENT, reply_to_message_id: int | None =
    None, allow_sending_without_reply: bool | None = None, reply_markup:
    InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply |
    None = None, **kwargs: Any) → SendPhoto
```

Shortcut for method `aiogram.methods.send_photo.SendPhoto` will automatically fill method attributes:

- `chat_id`
- `message_thread_id`

Use this method to send photos. On success, the sent `aiogram.types.message.Message` is returned.

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

#### Parameters

- **photo** – Photo to send. Pass a `file_id` as `String` to send a photo that exists on the Telegram servers (recommended), pass an `HTTP URL` as a `String` for Telegram to get a photo from the Internet, or upload a new photo using `multipart/form-data`. The photo must be at most 10 MB in size. The photo's width and height must not exceed 10000 in total. Width and height ratio must be at most 20. *More information on Sending Files »*
- **caption** – Photo caption (may also be used when resending photos by `file_id`), 0-1024 characters after entities parsing
- **parse\_mode** – Mode for parsing entities in the photo caption. See [formatting options](#) for more details.
- **caption\_entities** – A JSON-serialized list of special entities that appear in the caption, which can be specified instead of `parse_mode`
- **has\_spoiler** – Pass `True` if the photo needs to be covered with a spoiler animation
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass `True` if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method `aiogram.methods.send_photo.SendPhoto`

**reply\_poll**(*question: str, options: List[str], is\_anonymous: bool | None = None, type: str | None = None, allows\_multiple\_answers: bool | None = None, correct\_option\_id: int | None = None, explanation: str | None = None, explanation\_parse\_mode: str | None = sentinel.UNSET\_PARSE\_MODE, explanation\_entities: List[MessageEntity] | None = None, open\_period: int | None = None, close\_date: datetime.datetime | datetime.timedelta | int | None = None, is\_closed: bool | None = None, disable\_notification: bool | None = None, protect\_content: bool | None = sentinel.UNSET\_PROTECT\_CONTENT, allow\_sending\_without\_reply: bool | None = None, reply\_markup: InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply | None = None, \*\*kwargs: Any*) → *SendPoll*

Shortcut for method `aiogram.methods.send_poll.SendPoll` will automatically fill method attributes:

- `chat_id`
- `message_thread_id`
- `reply_to_message_id`

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

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

#### Parameters

- **question** – Poll question, 1-300 characters
- **options** – A JSON-serialized list of answer options, 2-10 strings 1-100 characters each
- **is\_anonymous** – True, if the poll needs to be anonymous, defaults to True
- **type** – Poll type, ‘quiz’ or ‘regular’, defaults to ‘regular’
- **allows\_multiple\_answers** – True, if the poll allows multiple answers, ignored for polls in quiz mode, defaults to False
- **correct\_option\_id** – 0-based identifier of the correct answer option, required for polls in quiz mode
- **explanation** – 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** – Mode for parsing entities in the explanation. See [formatting options](#) for more details.
- **explanation\_entities** – A JSON-serialized list of special entities that appear in the poll explanation, which can be specified instead of *parse\_mode*
- **open\_period** – Amount of time in seconds the poll will be active after creation, 5-600. Can’t be used together with *close\_date*.
- **close\_date** – 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** – Pass True if the poll needs to be immediately closed. This can be useful for poll preview.
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method `aiogram.methods.send_poll.SendPoll`

```
answer_poll(question: str, options: List[str], is_anonymous: bool | None = None, type: str | None = None,
allows_multiple_answers: bool | None = None, correct_option_id: int | None = None,
explanation: str | None = None, explanation_parse_mode: str | None =
sentinel.UNSET_PARSE_MODE, explanation_entities: List[MessageEntity] | None = None,
open_period: int | None = None, close_date: datetime.datetime | datetime.timedelta | int | None
= None, is_closed: bool | None = None, disable_notification: bool | None = None,
protect_content: bool | None = sentinel.UNSET_PROTECT_CONTENT, reply_to_message_id:
int | None = None, allow_sending_without_reply: bool | None = None, reply_markup:
InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply | None
= None, **kwargs: Any) → SendPoll
```

Shortcut for method `aiogram.methods.send_poll.SendPoll` will automatically fill method attributes:

- `chat_id`



- `message_thread_id`

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

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

#### Parameters

- **question** – Poll question, 1-300 characters
- **options** – A JSON-serialized list of answer options, 2-10 strings 1-100 characters each
- **is\_anonymous** – True, if the poll needs to be anonymous, defaults to True
- **type** – Poll type, ‘quiz’ or ‘regular’, defaults to ‘regular’
- **allows\_multiple\_answers** – True, if the poll allows multiple answers, ignored for polls in quiz mode, defaults to False
- **correct\_option\_id** – 0-based identifier of the correct answer option, required for polls in quiz mode
- **explanation** – 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** – Mode for parsing entities in the explanation. See [formatting options](#) for more details.
- **explanation\_entities** – A JSON-serialized list of special entities that appear in the poll explanation, which can be specified instead of `parse_mode`
- **open\_period** – Amount of time in seconds the poll will be active after creation, 5-600. Can’t be used together with `close_date`.
- **close\_date** – 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** – Pass True if the poll needs to be immediately closed. This can be useful for poll preview.
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method `aiogram.methods.send_poll.SendPoll`

**reply\_dice**(*emoji*: *str* | *None* = *None*, *disable\_notification*: *bool* | *None* = *None*, *protect\_content*: *bool* | *None* = *sentinel.UNSET\_PROTECT\_CONTENT*, *allow\_sending\_without\_reply*: *bool* | *None* = *None*, *reply\_markup*: [InlineKeyboardMarkup](#) | [ReplyKeyboardMarkup](#) | [ReplyKeyboardRemove](#) | [ForceReply](#) | *None* = *None*, *\*\*kwargs*: *Any*) → [SendDice](#)

Shortcut for method `aiogram.methods.send_dice.SendDice` will automatically fill method attributes:



- `chat_id`
- `message_thread_id`
- `reply_to_message_id`

Use this method to send an animated emoji that will display a random value. On success, the sent [`aiogram.types.message.Message`](#) is returned.

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

#### Parameters

- **emoji** – 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** – Sends the message [`silently`](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding
- **allow\_sending\_without\_reply** – Pass `True` if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method [`aiogram.methods.send\_dice.SendDice`](#)

**answer\_dice**(*emoji: str | None = None, disable\_notification: bool | None = None, protect\_content: bool | None = sentinel.UNSET\_PROTECT\_CONTENT, reply\_to\_message\_id: int | None = None, allow\_sending\_without\_reply: bool | None = None, reply\_markup: InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply | None = None, \*\*kwargs: Any*) → [`SendDice`](#)

Shortcut for method [`aiogram.methods.send\_dice.SendDice`](#) will automatically fill method attributes:

- `chat_id`
- `message_thread_id`

Use this method to send an animated emoji that will display a random value. On success, the sent [`aiogram.types.message.Message`](#) is returned.

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

#### Parameters

- **emoji** – 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** – Sends the message [`silently`](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass `True` if the message should be sent even if the specified replied-to message is not found

- **reply\_markup** – 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

instance of method [aiogram.methods.send\\_dice.SendDice](#)

**reply\_sticker**(*sticker: InputFile | str, emoji: str | None = None, disable\_notification: bool | None = None, protect\_content: bool | None = sentinel.UNSET\_PROTECT\_CONTENT, allow\_sending\_without\_reply: bool | None = None, reply\_markup: InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply | None = None, \*\*kwargs: Any*) → *SendSticker*

Shortcut for method [aiogram.methods.send\\_sticker.SendSticker](#) will automatically fill method attributes:

- chat\_id
- message\_thread\_id
- reply\_to\_message\_id

Use this method to send static [.WEBP](#), [animated .TGS](#), or [video .WEBM](#) stickers. On success, the sent [aiogram.types.message.Message](#) is returned.

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

#### Parameters

- **sticker** – Sticker to send. Pass a file\_id as String to send a file that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get a [.WEBP](#) sticker from the Internet, or upload a new [.WEBP](#) or [.TGS](#) sticker using multipart/form-data. [More information on Sending Files »](#). Video stickers can only be sent by a file\_id. Animated stickers can't be sent via an HTTP URL.
- **emoji** – Emoji associated with the sticker; only for just uploaded stickers
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method [aiogram.methods.send\\_sticker.SendSticker](#)

**answer\_sticker**(*sticker: InputFile | str, emoji: str | None = None, disable\_notification: bool | None = None, protect\_content: bool | None = sentinel.UNSET\_PROTECT\_CONTENT, reply\_to\_message\_id: int | None = None, allow\_sending\_without\_reply: bool | None = None, reply\_markup: InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply | None = None, \*\*kwargs: Any*) → *SendSticker*

Shortcut for method [aiogram.methods.send\\_sticker.SendSticker](#) will automatically fill method attributes:

- chat\_id
- message\_thread\_id

Use this method to send static .WEBP, *animated* .TGS, or *video* .WEBM stickers. On success, the sent *aiogram.types.message.Message* is returned.

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

#### Parameters

- **sticker** – Sticker to send. Pass a `file_id` as `String` to send a file that exists on the Telegram servers (recommended), pass an `HTTP URL` as a `String` for Telegram to get a .WEBP sticker from the Internet, or upload a new .WEBP or .TGS sticker using `multipart/form-data`. *More information on Sending Files* ». Video stickers can only be sent by a `file_id`. Animated stickers can't be sent via an `HTTP URL`.
- **emoji** – Emoji associated with the sticker; only for just uploaded stickers
- **disable\_notification** – Sends the message *silently*. Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass `True` if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method *aiogram.methods.send\_sticker.SendSticker*

**reply\_venue**(*latitude: float, longitude: float, title: str, address: str, foursquare\_id: str | None = None, foursquare\_type: str | None = None, google\_place\_id: str | None = None, google\_place\_type: str | None = None, disable\_notification: bool | None = None, protect\_content: bool | None = sentinel.UNSET\_PROTECT\_CONTENT, allow\_sending\_without\_reply: bool | None = None, reply\_markup: InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply | None = None, \*\*kwargs: Any*) → *SendVenue*

Shortcut for method *aiogram.methods.send\_venue.SendVenue* will automatically fill method attributes:

- `chat_id`
- `message_thread_id`
- `reply_to_message_id`

Use this method to send information about a venue. On success, the sent *aiogram.types.message.Message* is returned.

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

#### Parameters

- **latitude** – Latitude of the venue
- **longitude** – Longitude of the venue
- **title** – Name of the venue
- **address** – Address of the venue
- **foursquare\_id** – Foursquare identifier of the venue

- **foursquare\_type** – Foursquare type of the venue, if known. (For example, ‘arts\_entertainment/default’, ‘arts\_entertainment/aquarium’ or ‘food/icecream’.)
- **google\_place\_id** – Google Places identifier of the venue
- **google\_place\_type** – Google Places type of the venue. (See [supported types](#).)
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method [aiogram.methods.send\\_venue.SendVenue](#)

**answer\_venue**(latitude: float, longitude: float, title: str, address: str, foursquare\_id: str | None = None, foursquare\_type: str | None = None, google\_place\_id: str | None = None, google\_place\_type: str | None = None, disable\_notification: bool | None = None, protect\_content: bool | None = sentinel.UNSET\_PROTECT\_CONTENT, reply\_to\_message\_id: int | None = None, allow\_sending\_without\_reply: bool | None = None, reply\_markup: [InlineKeyboardMarkup](#) | [ReplyKeyboardMarkup](#) | [ReplyKeyboardRemove](#) | [ForceReply](#) | None = None, \*\*kwargs: Any) → [SendVenue](#)

Shortcut for method [aiogram.methods.send\\_venue.SendVenue](#) will automatically fill method attributes:

- chat\_id
- message\_thread\_id

Use this method to send information about a venue. On success, the sent [aiogram.types.message.Message](#) is returned.

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

#### Parameters

- **latitude** – Latitude of the venue
- **longitude** – Longitude of the venue
- **title** – Name of the venue
- **address** – Address of the venue
- **foursquare\_id** – Foursquare identifier of the venue
- **foursquare\_type** – Foursquare type of the venue, if known. (For example, ‘arts\_entertainment/default’, ‘arts\_entertainment/aquarium’ or ‘food/icecream’.)
- **google\_place\_id** – Google Places identifier of the venue
- **google\_place\_type** – Google Places type of the venue. (See [supported types](#).)
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message

- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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**

instance of method [aiogram.methods.send\\_venue.SendVenue](#)

**reply\_video**(video: [InputFile](#) | str, duration: int | None = None, width: int | None = None, height: int | None = None, thumbnail: [InputFile](#) | str | None = None, caption: str | None = None, parse\_mode: str | None = sentinel.UNSET\_PARSE\_MODE, caption\_entities: List[[MessageEntity](#)] | None = None, has\_spoiler: bool | None = None, supports\_streaming: bool | None = None, disable\_notification: bool | None = None, protect\_content: bool | None = sentinel.UNSET\_PROTECT\_CONTENT, allow\_sending\_without\_reply: bool | None = None, reply\_markup: [InlineKeyboardMarkup](#) | [ReplyKeyboardMarkup](#) | [ReplyKeyboardRemove](#) | [ForceReply](#) | None = None, \*\*kwargs: Any) → [SendVideo](#)

Shortcut for method [aiogram.methods.send\\_video.SendVideo](#) will automatically fill method attributes:

- chat\_id
- message\_thread\_id
- reply\_to\_message\_id

Use this method to send video files, Telegram clients support MPEG4 videos (other formats may be sent as [aiogram.types.document.Document](#)). On success, the sent [aiogram.types.message.Message](#) is returned. Bots can currently send video files of up to 50 MB in size, this limit may be changed in the future.

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

**Parameters**

- **video** – Video to send. Pass a file\_id as String to send a video that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get a video from the Internet, or upload a new video using multipart/form-data. [More information on Sending Files](#) »
- **duration** – Duration of sent video in seconds
- **width** – Video width
- **height** – Video height
- **thumbnail** – Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can't be reused and can be only uploaded as a new file, so you can pass 'attach://<file\_attach\_name>' if the thumbnail was uploaded using multipart/form-data under <file\_attach\_name>. [More information on Sending Files](#) »
- **caption** – Video caption (may also be used when resending videos by *file\_id*), 0-1024 characters after entities parsing
- **parse\_mode** – Mode for parsing entities in the video caption. See [formatting options](#) for more details.

- **caption\_entities** – A JSON-serialized list of special entities that appear in the caption, which can be specified instead of *parse\_mode*
- **has\_spoiler** – Pass True if the video needs to be covered with a spoiler animation
- **supports\_streaming** – Pass True if the uploaded video is suitable for streaming
- **disable\_notification** – Sends the message *silently*. Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method [`aiogram.methods.send\_video.SendVideo`](#)

**answer\_video**(*video*: [`InputFile`](#) | *str*, *duration*: *int* | *None* = *None*, *width*: *int* | *None* = *None*, *height*: *int* | *None* = *None*, *thumbnail*: [`InputFile`](#) | *str* | *None* = *None*, *caption*: *str* | *None* = *None*, *parse\_mode*: *str* | *None* = *sentinel.UNSET\_PARSE\_MODE*, *caption\_entities*: *List*[[`MessageEntity`](#)] | *None* = *None*, *has\_spoiler*: *bool* | *None* = *None*, *supports\_streaming*: *bool* | *None* = *None*, *disable\_notification*: *bool* | *None* = *None*, *protect\_content*: *bool* | *None* = *sentinel.UNSET\_PROTECT\_CONTENT*, *reply\_to\_message\_id*: *int* | *None* = *None*, *allow\_sending\_without\_reply*: *bool* | *None* = *None*, *reply\_markup*: [`InlineKeyboardMarkup`](#) | [`ReplyKeyboardMarkup`](#) | [`ReplyKeyboardRemove`](#) | [`ForceReply`](#) | *None* = *None*, *\*\*kwargs*: *Any*) → [`SendVideo`](#)

Shortcut for method [`aiogram.methods.send\_video.SendVideo`](#) will automatically fill method attributes:

- `chat_id`
- `message_thread_id`

Use this method to send video files, Telegram clients support MPEG4 videos (other formats may be sent as [`aiogram.types.document.Document`](#)). On success, the sent [`aiogram.types.message.Message`](#) is returned. Bots can currently send video files of up to 50 MB in size, this limit may be changed in the future.

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

#### Parameters

- **video** – Video to send. Pass a `file_id` as String to send a video that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get a video from the Internet, or upload a new video using multipart/form-data. [More information on Sending Files »](#)
- **duration** – Duration of sent video in seconds
- **width** – Video width
- **height** – Video height
- **thumbnail** – Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can't be reused and can be only uploaded as a new file, so you can pass 'attach://<file\_attach\_name>' if the thumbnail was uploaded

using multipart/form-data under <file\_attach\_name>. [More information on Sending Files](#) »

- **caption** – Video caption (may also be used when resending videos by *file\_id*), 0-1024 characters after entities parsing
- **parse\_mode** – Mode for parsing entities in the video caption. See [formatting options](#) for more details.
- **caption\_entities** – A JSON-serialized list of special entities that appear in the caption, which can be specified instead of *parse\_mode*
- **has\_spoiler** – Pass True if the video needs to be covered with a spoiler animation
- **supports\_streaming** – Pass True if the uploaded video is suitable for streaming
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method [aiogram.methods.send\\_video.SendVideo](#)

**reply\_video\_note**(*video\_note*: [InputFile](#) | *str* | *None* = *None*, *duration*: *int* | *None* = *None*, *length*: *int* | *None* = *None*, *thumbnail*: [InputFile](#) | *str* | *None* = *None*, *disable\_notification*: *bool* | *None* = *None*, *protect\_content*: *bool* | *None* = *sentinel.UNSET\_PROTECT\_CONTENT*, *allow\_sending\_without\_reply*: *bool* | *None* = *None*, *reply\_markup*: [InlineKeyboardMarkup](#) | [ReplyKeyboardMarkup](#) | [ReplyKeyboardRemove](#) | [ForceReply](#) | *None* = *None*, *\*\*kwargs*: *Any*) → [SendVideoNote](#)

Shortcut for method [aiogram.methods.send\\_video\\_note.SendVideoNote](#) will automatically fill method attributes:

- *chat\_id*
- *message\_thread\_id*
- *reply\_to\_message\_id*

As of v.4.0, Telegram clients support rounded square MPEG4 videos of up to 1 minute long. Use this method to send video messages. On success, the sent [aiogram.types.message.Message](#) is returned.

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

#### Parameters

- **video\_note** – Video note to send. Pass a *file\_id* as *String* to send a video note that exists on the Telegram servers (recommended) or upload a new video using multipart/form-data. [More information on Sending Files](#) ». Sending video notes by a URL is currently unsupported
- **duration** – Duration of sent video in seconds
- **length** – Video width and height, i.e. diameter of the video message



- **thumbnail** – Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can't be reused and can be only uploaded as a new file, so you can pass 'attach://<file\_attach\_name>' if the thumbnail was uploaded using multipart/form-data under <file\_attach\_name>. [More information on Sending Files](#) »
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method [aiogram.methods.send\\_video\\_note.SendVideoNote](#)

**answer\_video\_note**(*video\_note*: [InputFile](#) | *str*, *duration*: *int* | *None* = *None*, *length*: *int* | *None* = *None*, *thumbnail*: [InputFile](#) | *str* | *None* = *None*, *disable\_notification*: *bool* | *None* = *None*, *protect\_content*: *bool* | *None* = *sentinel.UNSET\_PROTECT\_CONTENT*, *reply\_to\_message\_id*: *int* | *None* = *None*, *allow\_sending\_without\_reply*: *bool* | *None* = *None*, *reply\_markup*: [InlineKeyboardMarkup](#) | [ReplyKeyboardMarkup](#) | [ReplyKeyboardRemove](#) | [ForceReply](#) | *None* = *None*, *\*\*kwargs*: *Any*) → [SendVideoNote](#)

Shortcut for method [aiogram.methods.send\\_video\\_note.SendVideoNote](#) will automatically fill method attributes:

- `chat_id`
- `message_thread_id`

As of v.4.0, Telegram clients support rounded square MPEG4 videos of up to 1 minute long. Use this method to send video messages. On success, the sent [aiogram.types.message.Message](#) is returned.

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

#### Parameters

- **video\_note** – Video note to send. Pass a `file_id` as `String` to send a video note that exists on the Telegram servers (recommended) or upload a new video using multipart/form-data. [More information on Sending Files](#) ». Sending video notes by a URL is currently unsupported
- **duration** – Duration of sent video in seconds
- **length** – Video width and height, i.e. diameter of the video message
- **thumbnail** – Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can't be reused and can be only uploaded as a new file, so you can pass 'attach://<file\_attach\_name>' if the thumbnail was uploaded using multipart/form-data under <file\_attach\_name>. [More information on Sending Files](#) »



- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method [aiogram.methods.send\\_video\\_note.SendVideoNote](#)

**reply\_voice**(voice: [InputFile](#) | str, caption: str | None = None, parse\_mode: str | None = [sentinel.UNSET\\_PARSE\\_MODE](#), caption\_entities: List[[MessageEntity](#)] | None = None, duration: int | None = None, disable\_notification: bool | None = None, protect\_content: bool | None = [sentinel.UNSET\\_PROTECT\\_CONTENT](#), allow\_sending\_without\_reply: bool | None = None, reply\_markup: [InlineKeyboardMarkup](#) | [ReplyKeyboardMarkup](#) | [ReplyKeyboardRemove](#) | [ForceReply](#) | None = None, \*\*kwargs: Any) → [SendVoice](#)

Shortcut for method [aiogram.methods.send\\_voice.SendVoice](#) will automatically fill method attributes:

- chat\_id
- message\_thread\_id
- reply\_to\_message\_id

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 [aiogram.types.audio.Audio](#) or [aiogram.types.document.Document](#)). On success, the sent [aiogram.types.message.Message](#) is returned. Bots can currently send voice messages of up to 50 MB in size, this limit may be changed in the future.

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

#### Parameters

- **voice** – Audio file to send. Pass a file\_id as String to send a file that exists on the Telegram servers (recommended), 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 information on Sending Files](#) »
- **caption** – Voice message caption, 0-1024 characters after entities parsing
- **parse\_mode** – Mode for parsing entities in the voice message caption. See [formatting options](#) for more details.
- **caption\_entities** – A JSON-serialized list of special entities that appear in the caption, which can be specified instead of *parse\_mode*
- **duration** – Duration of the voice message in seconds
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found

- **reply\_markup** – 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

instance of method [aiogram.methods.send\\_voice.SendVoice](#)

**answer\_voice**(voice: [InputFile](#) | str, caption: str | None = None, parse\_mode: str | None = sentinel.UNSET\_PARSE\_MODE, caption\_entities: List[[MessageEntity](#)] | None = None, duration: int | None = None, disable\_notification: bool | None = None, protect\_content: bool | None = sentinel.UNSET\_PROTECT\_CONTENT, reply\_to\_message\_id: int | None = None, allow\_sending\_without\_reply: bool | None = None, reply\_markup: [InlineKeyboardMarkup](#) | [ReplyKeyboardMarkup](#) | [ReplyKeyboardRemove](#) | [ForceReply](#) | None = None, \*\*kwargs: Any) → [SendVoice](#)

Shortcut for method [aiogram.methods.send\\_voice.SendVoice](#) will automatically fill method attributes:

- chat\_id
- message\_thread\_id

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 [aiogram.types.audio.Audio](#) or [aiogram.types.document.Document](#)). On success, the sent [aiogram.types.message.Message](#) is returned. Bots can currently send voice messages of up to 50 MB in size, this limit may be changed in the future.

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

#### Parameters

- **voice** – Audio file to send. Pass a file\_id as String to send a file that exists on the Telegram servers (recommended), 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 information on Sending Files](#) »
- **caption** – Voice message caption, 0-1024 characters after entities parsing
- **parse\_mode** – Mode for parsing entities in the voice message caption. See [formatting options](#) for more details.
- **caption\_entities** – A JSON-serialized list of special entities that appear in the caption, which can be specified instead of *parse\_mode*
- **duration** – Duration of the voice message in seconds
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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

instance of method [aiogram.methods.send\\_voice.SendVoice](#)

**send\_copy**(*chat\_id*: str | int, *disable\_notification*: bool | None = None, *reply\_to\_message\_id*: int | None = None, *reply\_markup*: InlineKeyboardMarkup | ReplyKeyboardMarkup | None = None, *allow\_sending\_without\_reply*: bool | None = None, *message\_thread\_id*: int | None = None) → ForwardMessage | SendAnimation | SendAudio | SendContact | SendDocument | SendLocation | SendMessage | SendPhoto | SendPoll | SendDice | SendSticker | SendVenue | SendVideo | SendVideoNote | SendVoice

Send copy of a message.

Is similar to `aiogram.client.bot.Bot.copy_message()` but returning the sent message instead of `aiogram.types.message_id.MessageId`

---

**Note:** This method doesn't use the API method named `copyMessage` and historically implemented before the similar method is added to API

---

#### Parameters

- **chat\_id** –
- **disable\_notification** –
- **reply\_to\_message\_id** –
- **reply\_markup** –
- **allow\_sending\_without\_reply** –
- **message\_thread\_id** –

#### Returns

**copy\_to**(*chat\_id*: int | str, *message\_thread\_id*: int | None = None, *caption*: str | None = None, *parse\_mode*: str | None = sentinel.UNSET\_PARSE\_MODE, *caption\_entities*: List[MessageEntity] | None = None, *disable\_notification*: bool | None = None, *protect\_content*: bool | None = sentinel.UNSET\_PROTECT\_CONTENT, *reply\_to\_message\_id*: int | None = None, *allow\_sending\_without\_reply*: bool | None = None, *reply\_markup*: InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply | None = None, *\*\*kwargs*: Any) → CopyMessage

Shortcut for method `aiogram.methods.copy_message.CopyMessage` will automatically fill method attributes:

- **from\_chat\_id**
- **message\_id**

Use this method to copy messages of any kind. Service messages and invoice messages can't be copied. A quiz `aiogram.methods.poll.Poll` can be copied only if the value of the field `correct_option_id` is known to the bot. The method is analogous to the method `aiogram.methods.forward_message.ForwardMessage`, but the copied message doesn't have a link to the original message. Returns the `aiogram.types.message_id.MessageId` of the sent message on success.

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

#### Parameters

- **chat\_id** – Unique identifier for the target chat or username of the target channel (in the format @channelusername)
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only

- **caption** – New caption for media, 0-1024 characters after entities parsing. If not specified, the original caption is kept
- **parse\_mode** – Mode for parsing entities in the new caption. See [formatting options](#) for more details.
- **caption\_entities** – A JSON-serialized list of special entities that appear in the new caption, which can be specified instead of *parse\_mode*
- **disable\_notification** – Sends the message [silently](#). Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the sent message from forwarding and saving
- **reply\_to\_message\_id** – If the message is a reply, ID of the original message
- **allow\_sending\_without\_reply** – Pass True if the message should be sent even if the specified replied-to message is not found
- **reply\_markup** – 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**

instance of method [aiogram.methods.copy\\_message.CopyMessage](#)

**edit\_text**(*text: str, inline\_message\_id: str | None = None, parse\_mode: str | None = sentinel.UNSET\_PARSE\_MODE, entities: List[MessageEntity] | None = None, disable\_web\_page\_preview: bool | None = sentinel.UNSET\_DISABLE\_WEB\_PAGE\_PREVIEW, reply\_markup: InlineKeyboardMarkup | None = None, \*\*kwargs: Any*) → [EditMessageText](#)

Shortcut for method [aiogram.methods.edit\\_message\\_text.EditMessageText](#) will automatically fill method attributes:

- `chat_id`
- `message_id`

Use this method to edit text and [game](#) messages. On success, if the edited message is not an inline message, the edited [aiogram.types.message.Message](#) is returned, otherwise True is returned.

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

**Parameters**

- **text** – New text of the message, 1-4096 characters after entities parsing
- **inline\_message\_id** – Required if *chat\_id* and *message\_id* are not specified. Identifier of the inline message
- **parse\_mode** – Mode for parsing entities in the message text. See [formatting options](#) for more details.
- **entities** – A JSON-serialized list of special entities that appear in message text, which can be specified instead of *parse\_mode*
- **disable\_web\_page\_preview** – Disables link previews for links in this message
- **reply\_markup** – A JSON-serialized object for an [inline keyboard](#).

**Returns**

instance of method [aiogram.methods.edit\\_message\\_text.EditMessageText](#)

**forward**(*chat\_id*: int | str, *message\_thread\_id*: int | None = None, *disable\_notification*: bool | None = None, *protect\_content*: bool | None = sentinel.UNSET\_PROTECT\_CONTENT, *\*\*kwargs*: Any) → *ForwardMessage*

Shortcut for method `aiogram.methods.forward_message.ForwardMessage` will automatically fill method attributes:

- `from_chat_id`
- `message_id`

Use this method to forward messages of any kind. Service messages can't be forwarded. On success, the sent `aiogram.types.message.Message` is returned.

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

#### Parameters

- **chat\_id** – Unique identifier for the target chat or username of the target channel (in the format @channelusername)
- **message\_thread\_id** – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
- **disable\_notification** – Sends the message `silently`. Users will receive a notification with no sound.
- **protect\_content** – Protects the contents of the forwarded message from forwarding and saving

#### Returns

instance of method `aiogram.methods.forward_message.ForwardMessage`

**edit\_media**(*media*: InputMediaAnimation | InputMediaDocument | InputMediaAudio | InputMediaPhoto | InputMediaVideo, *inline\_message\_id*: str | None = None, *reply\_markup*: InlineKeyboardMarkup | None = None, *\*\*kwargs*: Any) → *EditMessageMedia*

Shortcut for method `aiogram.methods.edit_message_media.EditMessageMedia` will automatically fill method attributes:

- `chat_id`
- `message_id`

Use this method to edit animation, audio, document, photo, or video messages. If a message is part of a message album, then it can be edited only to an audio for audio albums, only to a document for document albums and to a photo or a video otherwise. When an inline message is edited, a new file can't be uploaded; use a previously uploaded file via its `file_id` or specify a URL. On success, if the edited message is not an inline message, the edited `aiogram.types.message.Message` is returned, otherwise `True` is returned.

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

#### Parameters

- **media** – A JSON-serialized object for a new media content of the message
- **inline\_message\_id** – Required if `chat_id` and `message_id` are not specified. Identifier of the inline message
- **reply\_markup** – A JSON-serialized object for a new `inline keyboard`.

#### Returns

instance of method `aiogram.methods.edit_message_media.EditMessageMedia`

**edit\_reply\_markup**(*inline\_message\_id*: str | None = None, *reply\_markup*: InlineKeyboardMarkup | None = None, *\*\*kwargs*: Any) → *EditMessageReplyMarkup*

Shortcut for method `aiogram.methods.edit_message_reply_markup.EditMessageReplyMarkup` will automatically fill method attributes:

- `chat_id`
- `message_id`

Use this method to edit only the reply markup of messages. On success, if the edited message is not an inline message, the edited `aiogram.types.message.Message` is returned, otherwise True is returned.

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

#### Parameters

- **inline\_message\_id** – Required if `chat_id` and `message_id` are not specified. Identifier of the inline message
- **reply\_markup** – A JSON-serialized object for an inline keyboard.

#### Returns

instance of method `aiogram.methods.edit_message_reply_markup.EditMessageReplyMarkup`

**delete\_reply\_markup**(*inline\_message\_id*: str | None = None, *\*\*kwargs*: Any) → *EditMessageReplyMarkup*

Shortcut for method `aiogram.methods.edit_message_reply_markup.EditMessageReplyMarkup` will automatically fill method attributes:

- `chat_id`
- `message_id`
- `reply_markup`

Use this method to edit only the reply markup of messages. On success, if the edited message is not an inline message, the edited `aiogram.types.message.Message` is returned, otherwise True is returned.

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

#### Parameters

**inline\_message\_id** – Required if `chat_id` and `message_id` are not specified. Identifier of the inline message

#### Returns

instance of method `aiogram.methods.edit_message_reply_markup.EditMessageReplyMarkup`

**edit\_live\_location**(*latitude*: float, *longitude*: float, *inline\_message\_id*: str | None = None, *horizontal\_accuracy*: float | None = None, *heading*: int | None = None, *proximity\_alert\_radius*: int | None = None, *reply\_markup*: InlineKeyboardMarkup | None = None, *\*\*kwargs*: Any) → *EditMessageLiveLocation*

Shortcut for method `aiogram.methods.edit_message_live_location.EditMessageLiveLocation` will automatically fill method attributes:

- `chat_id`
- `message_id`

Use this method to edit live location messages. A location can be edited until its *live\_period* expires or editing is explicitly disabled by a call to `aiogram.methods.stop_message_live_location.StopMessageLiveLocation`. On success, if the edited message is not an inline message, the edited `aiogram.types.message.Message` is returned, otherwise True is returned.

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

#### Parameters

- **latitude** – Latitude of new location
- **longitude** – Longitude of new location
- **inline\_message\_id** – Required if *chat\_id* and *message\_id* are not specified. Identifier of the inline message
- **horizontal\_accuracy** – The radius of uncertainty for the location, measured in meters; 0-1500
- **heading** – Direction in which the user is moving, in degrees. Must be between 1 and 360 if specified.
- **proximity\_alert\_radius** – The maximum distance for proximity alerts about approaching another chat member, in meters. Must be between 1 and 100000 if specified.
- **reply\_markup** – A JSON-serialized object for a new inline keyboard.

#### Returns

instance of method `aiogram.methods.edit_message_live_location.EditMessageLiveLocation`

**stop\_live\_location**(*inline\_message\_id*: str | None = None, *reply\_markup*: InlineKeyboardMarkup | None = None, *\*\*kwargs*: Any) → `StopMessageLiveLocation`

Shortcut for method `aiogram.methods.stop_message_live_location.StopMessageLiveLocation` will automatically fill method attributes:

- *chat\_id*
- *message\_id*

Use this method to stop updating a live location message before *live\_period* expires. On success, if the message is not an inline message, the edited `aiogram.types.message.Message` is returned, otherwise True is returned.

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

#### Parameters

- **inline\_message\_id** – Required if *chat\_id* and *message\_id* are not specified. Identifier of the inline message
- **reply\_markup** – A JSON-serialized object for a new inline keyboard.

#### Returns

instance of method `aiogram.methods.stop_message_live_location.StopMessageLiveLocation`

**edit\_caption**(*inline\_message\_id*: str | None = None, *caption*: str | None = None, *parse\_mode*: str | None = *sentinel.UNSET\_PARSE\_MODE*, *caption\_entities*: List[MessageEntity] | None = None, *reply\_markup*: InlineKeyboardMarkup | None = None, *\*\*kwargs*: Any) → `EditMessageCaption`

Shortcut for method `aiogram.methods.edit_message_caption.EditMessageCaption` will automatically fill method attributes:



- `chat_id`
- `message_id`

Use this method to edit captions of messages. On success, if the edited message is not an inline message, the edited `aiogram.types.message.Message` is returned, otherwise `True` is returned.

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

#### Parameters

- **`inline_message_id`** – Required if `chat_id` and `message_id` are not specified. Identifier of the inline message
- **`caption`** – New caption of the message, 0-1024 characters after entities parsing
- **`parse_mode`** – Mode for parsing entities in the message caption. See [formatting options](#) for more details.
- **`caption_entities`** – A JSON-serialized list of special entities that appear in the caption, which can be specified instead of `parse_mode`
- **`reply_markup`** – A JSON-serialized object for an [inline keyboard](#).

#### Returns

instance of method `aiogram.methods.edit_message_caption.EditMessageCaption`

**`delete(**kwargs: Any) → DeleteMessage`**

Shortcut for method `aiogram.methods.delete_message.DeleteMessage` will automatically fill method attributes:

- `chat_id`
- `message_id`

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.
- Service messages about a supergroup, channel, or forum topic creation can't be deleted.
- A dice message in a private chat can only be deleted if it was sent more than 24 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.

Returns `True` on success.

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

#### Returns

instance of method `aiogram.methods.delete_message.DeleteMessage`

**`pin(disable_notification: bool | None = None, **kwargs: Any) → PinChatMessage`**

Shortcut for method `aiogram.methods.pin_chat_message.PinChatMessage` will automatically fill method attributes:

- `chat_id`



- `message_id`

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’ administrator right in a supergroup or ‘can\_edit\_messages’ administrator right in a channel. Returns True on success.

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

#### Parameters

**disable\_notification** – Pass True if it is not necessary to send a notification to all chat members about the new pinned message. Notifications are always disabled in channels and private chats.

#### Returns

instance of method `aiogram.methods.pin_chat_message.PinChatMessage`

**unpin**(\*\*kwargs: Any) → *UnpinChatMessage*

Shortcut for method `aiogram.methods.unpin_chat_message.UnpinChatMessage` will automatically fill method attributes:

- `chat_id`
- `message_id`

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’ administrator right in a supergroup or ‘can\_edit\_messages’ administrator right in a channel. Returns True on success.

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

#### Returns

instance of method `aiogram.methods.unpin_chat_message.UnpinChatMessage`

**get\_url**(force\_private: bool = False) → str | None

Returns message URL. Cannot be used in private (one-to-one) chats. If chat has a username, returns URL like [https://t.me/username/message\\_id](https://t.me/username/message_id) Otherwise (or if {force\_private} flag is set), returns [https://t.me/c/shifted\\_chat\\_id/message\\_id](https://t.me/c/shifted_chat_id/message_id)

#### Parameters

**force\_private** – if set, a private URL is returned even for a public chat

#### Returns

string with full message URL

## MessageAutoDeleteTimerChanged

```
class aiogram.types.message_auto_delete_timer_changed.MessageAutoDeleteTimerChanged(*,
                                                                                       mes-
                                                                                       sage_auto_delete_time:
                                                                                       int,
                                                                                       **ex-
                                                                                       tra_data:
                                                                                       Any)
```

This object represents a service message about a change in auto-delete timer settings.

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

**message\_auto\_delete\_time: int**

New auto-delete time for messages in the chat; in seconds

## MessageEntity

```
class aiogram.types.message_entity.MessageEntity(*, type: str, offset: int, length: int, url: str | None =
None, user: User | None = None, language: str |
None = None, custom_emoji_id: str | None = None,
**extra_data: Any)
```

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

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

**type: str**

Type of the entity. Currently, can be 'mention' (@username), 'hashtag' (#hashtag), 'cashtag' (\$USD), 'bot\_command' (/start@jobs\_bot), 'url' (<https://telegram.org>), 'email' (do-not-reply@telegram.org), 'phone\_number' (+1-212-555-0123), 'bold' (**bold text**), 'italic' (*italic text*), 'underline' (underlined text), 'strikethrough' (strikethrough text), 'spoiler' (spoiler message), 'code' (monowidth string), 'pre' (monowidth block), 'text\_link' (for clickable text URLs), 'text\_mention' (for users [without usernames](#)), 'custom\_emoji' (for inline custom emoji stickers)

**offset: int**

Offset in [UTF-16 code units](#) to the start of the entity

**length: int**

Length of the entity in [UTF-16 code units](#)

**url: str | None**

*Optional.* For 'text\_link' only, URL that will be opened after user taps on the text

**user: User | None**

*Optional.* For 'text\_mention' only, the mentioned user

**language: str | None**

*Optional.* For 'pre' only, the programming language of the entity text

**custom\_emoji\_id: str | None**

*Optional.* For 'custom\_emoji' only, unique identifier of the custom emoji. Use [aiogram.methods.get\\_custom\\_emoji\\_stickers.GetCustomEmojiStickers](#) to get full information about the sticker

**extract\_from(text: str) → str**

## MessageId

```
class aiogram.types.message_id.MessageId(*, message_id: int, **extra_data: Any)
```

This object represents a unique message identifier.

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

**message\_id: int**

Unique message identifier

## PhotoSize

```
class aiogram.types.photo_size.PhotoSize(*, file_id: str, file_unique_id: str, width: int, height: int,
                                          file_size: int | None = None, **extra_data: Any)
```

This object represents one size of a photo or a `file` / `aiogram.methods.sticker.Sticker` thumbnail.

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

**file\_id:** `str`

Identifier for this file, which can be used to download or reuse the file

**file\_unique\_id:** `str`

Unique identifier for this file, which is supposed to be the same over time and for different bots. Can't be used to download or reuse the file.

**width:** `int`

Photo width

**height:** `int`

Photo height

**file\_size:** `int | None`

*Optional.* File size in bytes

## Poll

```
class aiogram.types.poll.Poll(*, id: str, question: str, options: List[PollOption], total_voter_count: int,
                              is_closed: bool, is_anonymous: bool, type: str, allows_multiple_answers:
                              bool, correct_option_id: int | None = None, explanation: str | None = None,
                              explanation_entities: List[MessageEntity] | None = None, open_period: int |
                              None = None, close_date: datetime[datetime] | None = None, **extra_data:
                              Any)
```

This object contains information about a poll.

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

**id:** `str`

Unique poll identifier

**question:** `str`

Poll question, 1-300 characters

**options:** `List[PollOption]`

List of poll options

**total\_voter\_count:** `int`

Total number of users that voted in the poll

**is\_closed:** `bool`

True, if the poll is closed

**is\_anonymous:** `bool`

True, if the poll is anonymous

**type:** `str`

Poll type, currently can be 'regular' or 'quiz'

**allows\_multiple\_answers:** `bool`

True, if the poll allows multiple answers

**correct\_option\_id:** `int | None`

*Optional.* 0-based identifier of the correct answer option. Available only for polls in the quiz mode, which are closed, or was sent (not forwarded) by the bot or to the private chat with the bot.

**explanation:** `str | None`

*Optional.* 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

**explanation\_entities:** `List[MessageEntity] | None`

*Optional.* Special entities like usernames, URLs, bot commands, etc. that appear in the *explanation*

**open\_period:** `int | None`

*Optional.* Amount of time in seconds the poll will be active after creation

**close\_date:** `DateTime | None`

*Optional.* Point in time (Unix timestamp) when the poll will be automatically closed

## PollAnswer

```
class aiogram.types.poll_answer.PollAnswer(*, poll_id: str, option_ids: List[int], voter_chat: Chat | None
                                           = None, user: User | None = None, **extra_data: Any)
```

This object represents an answer of a user in a non-anonymous poll.

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

**poll\_id:** `str`

Unique poll identifier

**option\_ids:** `List[int]`

0-based identifiers of chosen answer options. May be empty if the vote was retracted.

**voter\_chat:** `Chat | None`

*Optional.* The chat that changed the answer to the poll, if the voter is anonymous

**user:** `User | None`

*Optional.* The user that changed the answer to the poll, if the voter isn't anonymous

## PollOption

```
class aiogram.types.poll_option.PollOption(*, text: str, voter_count: int, **extra_data: Any)
```

This object contains information about one answer option in a poll.

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

**text:** `str`

Option text, 1-100 characters

**voter\_count:** `int`

Number of users that voted for this option

## ProximityAlertTriggered

```
class aiogram.types.proximity_alert_triggered.ProximityAlertTriggered(*, traveler: User,
                                                                    watcher: User, distance:
                                                                    int, **extra_data: Any)
```

This object represents the content of a service message, sent whenever a user in the chat triggers a proximity alert set by another user.

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

**traveler:** *User*

User that triggered the alert

**watcher:** *User*

User that set the alert

**distance:** *int*

The distance between the users

## ReplyKeyboardMarkup

```
class aiogram.types.reply_keyboard_markup.ReplyKeyboardMarkup(*, keyboard:
                                                                List[List[KeyboardButton]],
                                                                is_persistent: bool | None = None,
                                                                resize_keyboard: bool | None =
                                                                None, one_time_keyboard: bool |
                                                                None = None,
                                                                input_field_placeholder: str | None
                                                                = None, selective: bool | None =
                                                                None, **extra_data: Any)
```

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

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

**keyboard:** *List[List[KeyboardButton]]*

Array of button rows, each represented by an Array of *aiogram.types.keyboard\_button.KeyboardButton* objects

**is\_persistent:** *bool | None*

*Optional.* Requests clients to always show the keyboard when the regular keyboard is hidden. Defaults to *false*, in which case the custom keyboard can be hidden and opened with a keyboard icon.

**resize\_keyboard:** *bool | None*

*Optional.* Requests clients to resize the keyboard vertically for optimal fit (e.g., make the keyboard smaller if there are just two rows of buttons). Defaults to *false*, in which case the custom keyboard is always of the same height as the app's standard keyboard.

**one\_time\_keyboard:** *bool | None*

*Optional.* Requests clients to hide the keyboard as soon as it's been used. The keyboard will still be available, but clients will automatically display the usual letter-keyboard in the chat - the user can press a special button in the input field to see the custom keyboard again. Defaults to *false*.

**input\_field\_placeholder:** *str | None*

*Optional.* The placeholder to be shown in the input field when the keyboard is active; 1-64 characters

**selective:** `bool | None`

*Optional.* Use this parameter if you want to show the keyboard to specific users only. Targets: 1) users that are @mentioned in the *text* of the `aiogram.types.message.Message` object; 2) if the bot's message is a reply (has `reply_to_message_id`), sender of the original message.

## ReplyKeyboardRemove

```
class aiogram.types.reply_keyboard_remove.ReplyKeyboardRemove(*, remove_keyboard: Literal[True]
                                                             = True, selective: bool | None =
                                                             None, **extra_data: Any)
```

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 `aiogram.types.reply_keyboard_markup.ReplyKeyboardMarkup`).

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

**remove\_keyboard:** `Literal[True]`

Requests clients to remove the custom keyboard (user will not be able to summon this keyboard; if you want to hide the keyboard from sight but keep it accessible, use `one_time_keyboard` in `aiogram.types.reply_keyboard_markup.ReplyKeyboardMarkup`)

**selective:** `bool | None`

*Optional.* Use this parameter if you want to remove the keyboard for specific users only. Targets: 1) users that are @mentioned in the *text* of the `aiogram.types.message.Message` object; 2) if the bot's message is a reply (has `reply_to_message_id`), sender of the original message.

## ResponseParameters

```
class aiogram.types.response_parameters.ResponseParameters(*, migrate_to_chat_id: int | None =
                                                           None, retry_after: int | None = None,
                                                           **extra_data: Any)
```

Describes why a request was unsuccessful.

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

**migrate\_to\_chat\_id:** `int | None`

*Optional.* The group has been migrated to a supergroup with the specified identifier. This number may have more than 32 significant bits and some programming languages may have difficulty/silent defects in interpreting it. But it has at most 52 significant bits, so a signed 64-bit integer or double-precision float type are safe for storing this identifier.

**retry\_after:** `int | None`

*Optional.* In case of exceeding flood control, the number of seconds left to wait before the request can be repeated

## Story

**class** aiogram.types.story.Story(\*\*extra\_data: Any)

This object represents a message about a forwarded story in the chat. Currently holds no information.

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

## SwitchInlineQueryChosenChat

```
class aiogram.types.switch_inline_query_chosen_chat.SwitchInlineQueryChosenChat(*, query: str
                                                                    | None =
                                                                    None, al-
                                                                    low_user_chats:
                                                                    bool | None
                                                                    = None, al-
                                                                    low_bot_chats:
                                                                    bool | None
                                                                    = None, al-
                                                                    low_group_chats:
                                                                    bool | None
                                                                    = None, al-
                                                                    low_channel_chats:
                                                                    bool | None
                                                                    = None,
                                                                    **ex-
                                                                    tra_data:
                                                                    Any)
```

This object represents an inline button that switches the current user to inline mode in a chosen chat, with an optional default inline query.

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

**query:** str | None

*Optional.* The default inline query to be inserted in the input field. If left empty, only the bot's username will be inserted

**allow\_user\_chats:** bool | None

*Optional.* True, if private chats with users can be chosen

**allow\_bot\_chats:** bool | None

*Optional.* True, if private chats with bots can be chosen

**allow\_group\_chats:** bool | None

*Optional.* True, if group and supergroup chats can be chosen

**allow\_channel\_chats:** bool | None

*Optional.* True, if channel chats can be chosen

## User

```
class aiogram.types.user.User(*, id: int, is_bot: bool, first_name: str, last_name: str | None = None,
                               username: str | None = None, language_code: str | None = None,
                               is_premium: bool | None = None, added_to_attachment_menu: bool | None =
                               None, can_join_groups: bool | None = None, can_read_all_group_messages:
                               bool | None = None, supports_inline_queries: bool | None = None,
                               **extra_data: Any)
```

This object represents a Telegram user or bot.

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

**id: int**

Unique identifier for this user or bot. This number may have more than 32 significant bits and some programming languages may have difficulty/silent defects in interpreting it. But it has at most 52 significant bits, so a 64-bit integer or double-precision float type are safe for storing this identifier.

**is\_bot: bool**

True, if this user is a bot

**first\_name: str**

User's or bot's first name

**last\_name: str | None**

*Optional.* User's or bot's last name

**username: str | None**

*Optional.* User's or bot's username

**language\_code: str | None**

*Optional.* IETF language tag of the user's language

**is\_premium: bool | None**

*Optional.* True, if this user is a Telegram Premium user

**added\_to\_attachment\_menu: bool | None**

*Optional.* True, if this user added the bot to the attachment menu

**can\_join\_groups: bool | None**

*Optional.* True, if the bot can be invited to groups. Returned only in [aiogram.methods.get\\_me.GetMe](#).

**can\_read\_all\_group\_messages: bool | None**

*Optional.* True, if [privacy mode](#) is disabled for the bot. Returned only in [aiogram.methods.get\\_me.GetMe](#).

**supports\_inline\_queries: bool | None**

*Optional.* True, if the bot supports inline queries. Returned only in [aiogram.methods.get\\_me.GetMe](#).

**property full\_name: str**

**property url: str**

**mention\_markdown**(name: str | None = None) → str

**mention\_html**(name: str | None = None) → str



**get\_profile\_photos**(*offset: int | None = None, limit: int | None = None, \*\*kwargs: Any*) → *GetUserProfilePhotos*

Shortcut for method `aiogram.methods.get_user_profile_photos.GetUserProfilePhotos` will automatically fill method attributes:

- **user\_id**

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

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

#### Parameters

- **offset** – Sequential number of the first photo to be returned. By default, all photos are returned.
- **limit** – Limits the number of photos to be retrieved. Values between 1-100 are accepted. Defaults to 100.

#### Returns

instance of method `aiogram.methods.get_user_profile_photos.GetUserProfilePhotos`

## UserProfilePhotos

**class** `aiogram.types.user_profile_photos.UserProfilePhotos`(\**, total\_count: int, photos: List[List[PhotoSize]], \*\*extra\_data: Any*)

This object represent a user's profile pictures.

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

**total\_count: int**

Total number of profile pictures the target user has

**photos: List[List[PhotoSize]]**

Requested profile pictures (in up to 4 sizes each)

## UserShared

**class** `aiogram.types.user_shared.UserShared`(\**, request\_id: int, user\_id: int, \*\*extra\_data: Any*)

This object contains information about the user whose identifier was shared with the bot using a `aiogram.types.keyboard_button_request_user.KeyboardButtonRequestUser` button.

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

**request\_id: int**

Identifier of the request

**user\_id: int**

Identifier of the shared user. This number may have more than 32 significant bits and some programming languages may have difficulty/silent defects in interpreting it. But it has at most 52 significant bits, so a 64-bit integer or double-precision float type are safe for storing this identifier. The bot may not have access to the user and could be unable to use this identifier, unless the user is already known to the bot by some other means.

## Venue

```
class aiogram.types.venue.Venue(*, location: Location, title: str, address: str, foursquare_id: str | None =
    None, foursquare_type: str | None = None, google_place_id: str | None =
    None, google_place_type: str | None = None, **extra_data: Any)
```

This object represents a venue.

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

**location:** [Location](#)

Venue location. Can't be a live location

**title:** `str`

Name of the venue

**address:** `str`

Address of the venue

**foursquare\_id:** `str` | `None`

*Optional.* Foursquare identifier of the venue

**foursquare\_type:** `str` | `None`

*Optional.* Foursquare type of the venue. (For example, 'arts\_entertainment/default', 'arts\_entertainment/aquarium' or 'food/icecream'.)

**google\_place\_id:** `str` | `None`

*Optional.* Google Places identifier of the venue

**google\_place\_type:** `str` | `None`

*Optional.* Google Places type of the venue. (See [supported types](#).)

## Video

```
class aiogram.types.video.Video(*, file_id: str, file_unique_id: str, width: int, height: int, duration: int,
    thumbnail: PhotoSize | None = None, file_name: str | None = None,
    mime_type: str | None = None, file_size: int | None = None, **extra_data:
    Any)
```

This object represents a video file.

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

**file\_id:** `str`

Identifier for this file, which can be used to download or reuse the file

**file\_unique\_id:** `str`

Unique identifier for this file, which is supposed to be the same over time and for different bots. Can't be used to download or reuse the file.

**width:** `int`

Video width as defined by sender

**height:** `int`

Video height as defined by sender

**duration:** `int`

Duration of the video in seconds as defined by sender

**thumbnail:** [\*PhotoSize\*](#) | None

*Optional.* Video thumbnail

**file\_name:** str | None

*Optional.* Original filename as defined by sender

**mime\_type:** str | None

*Optional.* MIME type of the file as defined by sender

**file\_size:** int | None

*Optional.* File size in bytes. It can be bigger than  $2^{31}$  and some programming languages may have difficulty/silent defects in interpreting it. But it has at most 52 significant bits, so a signed 64-bit integer or double-precision float type are safe for storing this value.

### VideoChatEnded

**class** aiogram.types.video\_chat\_ended.**VideoChatEnded**(\*, duration: int, \*\*extra\_data: Any)

This object represents a service message about a video chat ended in the chat.

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

**duration:** int

Video chat duration in seconds

### VideoChatParticipantsInvited

**class** aiogram.types.video\_chat\_participants\_invited.**VideoChatParticipantsInvited**(\*, users: List[User], \*\*extra\_data: Any)

This object represents a service message about new members invited to a video chat.

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

**users:** List[User]

New members that were invited to the video chat

### VideoChatScheduled

**class** aiogram.types.video\_chat\_scheduled.**VideoChatScheduled**(\*, start\_date: datetime[datetime], \*\*extra\_data: Any)

This object represents a service message about a video chat scheduled in the chat.

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

**start\_date:** DateTime

Point in time (Unix timestamp) when the video chat is supposed to be started by a chat administrator

## VideoChatStarted

**class** aiogram.types.video\_chat\_started.**VideoChatStarted**(\*\*extra\_data: Any)

This object represents a service message about a video chat started in the chat. Currently holds no information.

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

## VideoNote

**class** aiogram.types.video\_note.**VideoNote**(\*, file\_id: str, file\_unique\_id: str, length: int, duration: int, thumbnail: PhotoSize | None = None, file\_size: int | None = None, \*\*extra\_data: Any)

This object represents a [video message](#) (available in Telegram apps as of v.4.0).

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

**file\_id: str**

Identifier for this file, which can be used to download or reuse the file

**file\_unique\_id: str**

Unique identifier for this file, which is supposed to be the same over time and for different bots. Can't be used to download or reuse the file.

**length: int**

Video width and height (diameter of the video message) as defined by sender

**duration: int**

Duration of the video in seconds as defined by sender

**thumbnail: PhotoSize | None**

*Optional.* Video thumbnail

**file\_size: int | None**

*Optional.* File size in bytes

## Voice

**class** aiogram.types.voice.**Voice**(\*, file\_id: str, file\_unique\_id: str, duration: int, mime\_type: str | None = None, file\_size: int | None = None, \*\*extra\_data: Any)

This object represents a voice note.

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

**file\_id: str**

Identifier for this file, which can be used to download or reuse the file

**file\_unique\_id: str**

Unique identifier for this file, which is supposed to be the same over time and for different bots. Can't be used to download or reuse the file.

**duration: int**

Duration of the audio in seconds as defined by sender

**mime\_type: str | None**

*Optional.* MIME type of the file as defined by sender

**file\_size:** `int` | `None`

*Optional.* File size in bytes. It can be bigger than  $2^{31}$  and some programming languages may have difficulty/silent defects in interpreting it. But it has at most 52 significant bits, so a signed 64-bit integer or double-precision float type are safe for storing this value.

## WebAppData

**class** aiogram.types.web\_app\_data.**WebAppData**(\*, data: str, button\_text: str, \*\*extra\_data: Any)

Describes data sent from a [Web App](#) to the bot.

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

**data:** `str`

The data. Be aware that a bad client can send arbitrary data in this field.

**button\_text:** `str`

Text of the *web\_app* keyboard button from which the Web App was opened. Be aware that a bad client can send arbitrary data in this field.

## WebAppInfo

**class** aiogram.types.web\_app\_info.**WebAppInfo**(\*, url: str, \*\*extra\_data: Any)

Describes a [Web App](#).

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

**url:** `str`

An HTTPS URL of a Web App to be opened with additional data as specified in [Initializing Web Apps](#)

## WriteAccessAllowed

**class** aiogram.types.write\_access\_allowed.**WriteAccessAllowed**(\*, web\_app\_name: str | None = None, \*\*extra\_data: Any)

This object represents a service message about a user allowing a bot to write messages after adding the bot to the attachment menu or launching a Web App from a link.

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

**web\_app\_name:** `str` | `None`

*Optional.* Name of the Web App which was launched from a link

## Telegram Passport

### EncryptedCredentials

**class** aiogram.types.encrypted\_credentials.**EncryptedCredentials**(\*, data: str, hash: str, secret: str, \*\*extra\_data: Any)

Describes data required for decrypting and authenticating [aiogram.types.encrypted\\_passport\\_element.EncryptedPassportElement](#). See the [Telegram Passport Documentation](#) for a complete description of the data decryption and authentication processes.

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

**data:** `str`

Base64-encoded encrypted JSON-serialized data with unique user's payload, data hashes and secrets required for `aiogram.types.encrypted_passport_element.EncryptedPassportElement` decryption and authentication

**hash:** `str`

Base64-encoded data hash for data authentication

**secret:** `str`

Base64-encoded secret, encrypted with the bot's public RSA key, required for data decryption

## EncryptedPassportElement

```
class aiogram.types.encrypted_passport_element.EncryptedPassportElement(*, type: str, hash: str,
    data: str | None =
    None, phone_number:
    str | None = None,
    email: str | None =
    None, files:
    List[PassportFile] |
    None = None,
    front_side:
    PassportFile | None =
    None, reverse_side:
    PassportFile | None =
    None, selfie:
    PassportFile | None =
    None, translation:
    List[PassportFile] |
    None = None,
    **extra_data: Any)
```

Describes documents or other Telegram Passport elements shared with the bot by the user.

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

**type:** `str`

Element type. One of 'personal\_details', 'passport', 'driver\_license', 'identity\_card', 'internal\_passport', 'address', 'utility\_bill', 'bank\_statement', 'rental\_agreement', 'passport\_registration', 'temporary\_registration', 'phone\_number', 'email'.

**hash:** `str`

Base64-encoded element hash for using in `aiogram.types.passport_element_error_unspecified.PassportElementErrorUnspecified`

**data:** `str | None`

*Optional.* Base64-encoded encrypted Telegram Passport element data provided by the user, available for 'personal\_details', 'passport', 'driver\_license', 'identity\_card', 'internal\_passport' and 'address' types. Can be decrypted and verified using the accompanying `aiogram.types.encrypted_credentials.EncryptedCredentials`.

**phone\_number:** `str | None`

*Optional.* User's verified phone number, available only for 'phone\_number' type

**email:** `str | None`

*Optional.* User's verified email address, available only for 'email' type

**files:** `List[PassportFile]` | `None`

*Optional.* Array of encrypted files with documents provided by the user, available for 'utility\_bill', 'bank\_statement', 'rental\_agreement', 'passport\_registration' and 'temporary\_registration' types. Files can be decrypted and verified using the accompanying `aiogram.types.encrypted_credentials.EncryptedCredentials`.

**front\_side:** `PassportFile` | `None`

*Optional.* Encrypted file with the front side of the document, provided by the user. Available for 'passport', 'driver\_license', 'identity\_card' and 'internal\_passport'. The file can be decrypted and verified using the accompanying `aiogram.types.encrypted_credentials.EncryptedCredentials`.

**reverse\_side:** `PassportFile` | `None`

*Optional.* Encrypted file with the reverse side of the document, provided by the user. Available for 'driver\_license' and 'identity\_card'. The file can be decrypted and verified using the accompanying `aiogram.types.encrypted_credentials.EncryptedCredentials`.

**selfie:** `PassportFile` | `None`

*Optional.* Encrypted file with the selfie of the user holding a document, provided by the user; available for 'passport', 'driver\_license', 'identity\_card' and 'internal\_passport'. The file can be decrypted and verified using the accompanying `aiogram.types.encrypted_credentials.EncryptedCredentials`.

**translation:** `List[PassportFile]` | `None`

*Optional.* Array of encrypted files with translated versions of documents provided by the user. Available if requested for 'passport', 'driver\_license', 'identity\_card', 'internal\_passport', 'utility\_bill', 'bank\_statement', 'rental\_agreement', 'passport\_registration' and 'temporary\_registration' types. Files can be decrypted and verified using the accompanying `aiogram.types.encrypted_credentials.EncryptedCredentials`.

## PassportData

```
class aiogram.types.passport_data.PassportData(*, data: List[EncryptedPassportElement], credentials:
                                             EncryptedCredentials, **extra_data: Any)
```

Describes Telegram Passport data shared with the bot by the user.

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

**data:** `List[EncryptedPassportElement]`

Array with information about documents and other Telegram Passport elements that was shared with the bot

**credentials:** `EncryptedCredentials`

Encrypted credentials required to decrypt the data

## PassportElementError

```
class aiogram.types.passport_element_error.PassportElementError(**extra_data: Any)
```

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

- `aiogram.types.passport_element_error_data_field.PassportElementErrorDataField`
- `aiogram.types.passport_element_error_front_side.PassportElementErrorFrontSide`
- `aiogram.types.passport_element_error_reverse_side.PassportElementErrorReverseSide`

- `aiogram.types.passport_element_error_selfie.PassportElementErrorSelfie`
- `aiogram.types.passport_element_error_file.PassportElementErrorFile`
- `aiogram.types.passport_element_error_files.PassportElementErrorFiles`
- `aiogram.types.passport_element_error_translation_file.PassportElementErrorTranslationFile`
- `aiogram.types.passport_element_error_translation_files.PassportElementErrorTranslationFiles`
- `aiogram.types.passport_element_error_unspecified.PassportElementErrorUnspecified`

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

## PassportElementErrorDataField

```
class aiogram.types.passport_element_error_data_field.PassportElementErrorDataField(*,
                                                                                      source:
                                                                                      ~typ-
                                                                                      ing.Literal[<PassportEle
                                                                                      'data'>]
                                                                                      =
                                                                                      Pass-
                                                                                      portEle-
                                                                                      mentEr-
                                                                                      rorType.DATA,
                                                                                      type:
                                                                                      str,
                                                                                      field_name:
                                                                                      str,
                                                                                      data_hash:
                                                                                      str,
                                                                                      mes-
                                                                                      sage:
                                                                                      str,
                                                                                      **ex-
                                                                                      tra_data:
                                                                                      ~typ-
                                                                                      ing.Any)
```

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.

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

**source:** `Literal[PassportElementErrorType.DATA]`

Error source, must be *data*

**type:** `str`

The section of the user's Telegram Passport which has the error, one of 'personal\_details', 'passport', 'driver\_license', 'identity\_card', 'internal\_passport', 'address'

**field\_name:** `str`

Name of the data field which has the error

**data\_hash:** `str`

Base64-encoded data hash



**message: str**  
Error message

### PassportElementErrorFile

```
class aiogram.types.passport_element_error_file.PassportElementErrorFile(*, source: ~typing.Literal[<PassportElementErrorType.FILE>] = PassportElementErrorType.FILE, type: str, file_hash: str, message: str, **extra_data: ~typing.Any)
```

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

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

**source: Literal[PassportElementErrorType.FILE]**

Error source, must be *file*

**type: str**

The section of the user's Telegram Passport which has the issue, one of 'utility\_bill', 'bank\_statement', 'rental\_agreement', 'passport\_registration', 'temporary\_registration'

**file\_hash: str**

Base64-encoded file hash

**message: str**

Error message

### PassportElementErrorFiles

```
class aiogram.types.passport_element_error_files.PassportElementErrorFiles(*, source: ~typing.Literal[<PassportElementErrorType.FILES>] = PassportElementErrorType.FILES, type: str, file_hashes: ~typing.List[str], message: str, **extra_data: ~typing.Any)
```

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

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

**source: Literal[PassportElementErrorType.FILES]**

Error source, must be *files*

**type:** `str`

The section of the user's Telegram Passport which has the issue, one of 'utility\_bill', 'bank\_statement', 'rental\_agreement', 'passport\_registration', 'temporary\_registration'

**file\_hashes:** `List[str]`

List of base64-encoded file hashes

**message:** `str`

Error message

## PassportElementErrorFrontSide

```
class aiogram.types.passport_element_error_front_side.PassportElementErrorFrontSide(*,
                                                                                      source:
                                                                                      ~typ-
                                                                                      ing.Literal[<PassportEle-
                                                                                      'front_side'>]
                                                                                      =
                                                                                      Pass-
                                                                                      portEle-
                                                                                      mentEr-
                                                                                      rorType.FRONT_SIDE,
                                                                                      type:
                                                                                      str,
                                                                                      file_hash:
                                                                                      str,
                                                                                      mes-
                                                                                      sage:
                                                                                      str,
                                                                                      **ex-
                                                                                      tra_data:
                                                                                      ~typ-
                                                                                      ing.Any)
```

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.

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

**source:** `Literal[PassportElementType.FRONT_SIDE]`

Error source, must be *front\_side*

**type:** `str`

The section of the user's Telegram Passport which has the issue, one of 'passport', 'driver\_license', 'identity\_card', 'internal\_passport'

**file\_hash:** `str`

Base64-encoded hash of the file with the front side of the document

**message:** `str`

Error message

## PassportElementErrorReverseSide

```
class aiogram.types.passport_element_error_reverse_side.PassportElementErrorReverseSide(*,
                                                                 source: ~typing.Literal[<PassportElementErrorType.REVERSE_SIDE>] =
                                                                 PassportElementErrorType.REVERSE_SIDE,
                                                                 type: str,
                                                                 file_hash: str,
                                                                 message: str,
                                                                 **extra_data: ~typing.Any)
```

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.

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

**source:** `Literal[PassportElementErrorType.REVERSE_SIDE]`

Error source, must be *reverse\_side*

**type:** `str`

The section of the user's Telegram Passport which has the issue, one of 'driver\_license', 'identity\_card'

**file\_hash:** `str`

Base64-encoded hash of the file with the reverse side of the document

**message:** `str`

Error message

## PassportElementErrorSelfie

```
class aiogram.types.passport_element_error_selfie.PassportElementErrorSelfie(*, source: ~typing.Literal[<PassportElementErrorType.SELFIE>] =
                                                                 PassportElementErrorType.SELFIE,
                                                                 type: str,
                                                                 file_hash: str,
                                                                 message: str,
                                                                 **extra_data: ~typing.Any)
```

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

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

**source:** `Literal[PassportElementType.SELFIE]`

Error source, must be *selfie*

**type:** `str`

The section of the user's Telegram Passport which has the issue, one of 'passport', 'driver\_license', 'identity\_card', 'internal\_passport'

**file\_hash:** `str`

Base64-encoded hash of the file with the selfie

**message:** `str`

Error message

### PassportElementErrorTranslationFile

`class aiogram.types.passport_element_error_translation_file.PassportElementErrorTranslationFile(*,`

*source:*  
*~typ-*  
*ing.Litera*  
*'trans-*  
*la-*  
*tion\_file'>*  
*=*  
*Pass-*  
*portEle-*  
*mentEr-*  
*rorType.T*  
*type:*  
*str,*  
*file\_hash:*  
*str,*  
*mes-*  
*sage:*  
*str,*  
*\*\*ex-*  
*tra\_data:*  
*~typ-*  
*ing.Any)*

Represents an issue with one of the files that constitute the translation of a document. The error is considered resolved when the file changes.

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

**source:** `Literal[PassportElementType.TRANSLATION_FILE]`

Error source, must be *translation\_file*

**type:** `str`

Type of element of the user's Telegram Passport which has the issue, one of 'passport', 'driver\_license', 'identity\_card', 'internal\_passport', 'utility\_bill', 'bank\_statement', 'rental\_agreement', 'passport\_registration', 'temporary\_registration'

**file\_hash:** **str**  
Base64-encoded file hash

**message:** **str**  
Error message

### PassportElementErrorTranslationFiles

```
class aiogram.types.passport_element_error_translation_files.PassportElementErrorTranslationFiles(*,
source: typing.Literal[PassportElementErrorType.TRANSLATION_FILES] =
    PassportElementErrorTranslationFiles(
        source: typing.Literal[PassportElementErrorType.TRANSLATION_FILES],
        file_hash: typing.Literal[PassportElementErrorType.TRANSLATION_FILES],
        message: str,
        **kwargs: Any)
    )
```

Represents an issue with the translated version of a document. The error is considered resolved when a file with the document translation change.

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

**source:** **Literal**[**PassportElementErrorType**.**TRANSLATION\_FILES**]  
Error source, must be *translation\_files*

**type:** **str**  
Type of element of the user's Telegram Passport which has the issue, one of 'passport', 'driver\_license', 'identity\_card', 'internal\_passport', 'utility\_bill', 'bank\_statement', 'rental\_agreement', 'passport\_registration', 'temporary\_registration'

**file\_hashes:** **List**[**str**]  
List of base64-encoded file hashes

**message:** **str**  
Error message

## PassportElementErrorUnspecified

```
class aiogram.types.passport_element_error_unspecified.PassportElementErrorUnspecified(*,
                                                                                       source:
                                                                                         ~typing.Literal[<Passpor
                                                                                         'un-
                                                                                         spec-
                                                                                         i-
                                                                                         fied'>]
                                                                                       =
                                                                                       Pass-
                                                                                       portEle-
                                                                                       mentEr-
                                                                                       rorType.UNSPECIFIED)
                                                                                       type:
                                                                                         str,
                                                                                         el-
                                                                                         e-
                                                                                         ment_hash:
                                                                                         str,
                                                                                         mes-
                                                                                         sage:
                                                                                         str,
                                                                                         **ex-
                                                                                         tra_data:
                                                                                         ~typing.Any)
```

Represents an issue in an unspecified place. The error is considered resolved when new data is added.

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

**source:** `Literal[PassportElementType.UNSPECIFIED]`

Error source, must be *unspecified*

**type:** `str`

Type of element of the user's Telegram Passport which has the issue

**element\_hash:** `str`

Base64-encoded element hash

**message:** `str`

Error message

## PassportFile

```
class aiogram.types.passport_file.PassportFile(*, file_id: str, file_unique_id: str, file_size: int,
                                                  file_date: int, **extra_data: Any)
```

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.

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

**file\_id: str**

Identifier for this file, which can be used to download or reuse the file

**file\_unique\_id: str**

Unique identifier for this file, which is supposed to be the same over time and for different bots. Can't be used to download or reuse the file.

**file\_size: int**

File size in bytes

**file\_date: int**

Unix time when the file was uploaded

## Getting updates

### Update

```
class aiogram.types.update.Update(*, update_id: int, message: Message | None = None, edited_message:
    Message | None = None, channel_post: Message | None = None,
    edited_channel_post: Message | None = None, inline_query:
    InlineQuery | None = None, chosen_inline_result: ChosenInlineResult |
    None = None, callback_query: CallbackQuery | None = None,
    shipping_query: ShippingQuery | None = None, pre_checkout_query:
    PreCheckoutQuery | None = None, poll: Poll | None = None,
    poll_answer: PollAnswer | None = None, my_chat_member:
    ChatMemberUpdated | None = None, chat_member:
    ChatMemberUpdated | None = None, chat_join_request:
    ChatJoinRequest | None = None, **extra_data: Any)
```

This object represents an incoming update.

At most **one** of the optional parameters can be present in any given update.Source: <https://core.telegram.org/bots/api#update>**update\_id: int**The update's unique identifier. Update identifiers start from a certain positive number and increase sequentially. This ID becomes especially handy if you're using [webhooks](#), since it allows you to ignore repeated updates or to restore the correct update sequence, should they get out of order. If there are no new updates for at least a week, then identifier of the next update will be chosen randomly instead of sequentially.**message: Message | None***Optional.* New incoming message of any kind - text, photo, sticker, etc.**edited\_message: Message | None***Optional.* New version of a message that is known to the bot and was edited**channel\_post: Message | None***Optional.* New incoming channel post of any kind - text, photo, sticker, etc.**edited\_channel\_post: Message | None***Optional.* New version of a channel post that is known to the bot and was edited**inline\_query: InlineQuery | None***Optional.* New incoming [inline](#) query

**chosen\_inline\_result:** *ChosenInlineResult* | None

*Optional.* The result of an *inline* query that was chosen by a user and sent to their chat partner. Please see our documentation on the *feedback collecting* for details on how to enable these updates for your bot.

**callback\_query:** *CallbackQuery* | None

*Optional.* New incoming callback query

**shipping\_query:** *ShippingQuery* | None

*Optional.* New incoming shipping query. Only for invoices with flexible price

**pre\_checkout\_query:** *PreCheckoutQuery* | None

*Optional.* New incoming pre-checkout query. Contains full information about checkout

**poll:** *Poll* | None

*Optional.* New poll state. Bots receive only updates about stopped polls and polls, which are sent by the bot

**poll\_answer:** *PollAnswer* | None

*Optional.* A user changed their answer in a non-anonymous poll. Bots receive new votes only in polls that were sent by the bot itself.

**my\_chat\_member:** *ChatMemberUpdated* | None

*Optional.* The bot's chat member status was updated in a chat. For private chats, this update is received only when the bot is blocked or unblocked by the user.

**chat\_member:** *ChatMemberUpdated* | None

*Optional.* A chat member's status was updated in a chat. The bot must be an administrator in the chat and must explicitly specify 'chat\_member' in the list of *allowed\_updates* to receive these updates.

**chat\_join\_request:** *ChatJoinRequest* | None

*Optional.* A request to join the chat has been sent. The bot must have the *can\_invite\_users* administrator right in the chat to receive these updates.

**property event\_type:** str

Detect update type If update type is unknown, raise *UpdateTypeLookupError*

**Returns**

**property event:** TelegramObject

**exception** aiogram.types.update.UpdateTypeLookupError

Update does not contain any known event type.

## WebhookInfo

```
class aiogram.types.webhook_info.WebhookInfo(*, url: str, has_custom_certificate: bool,
                                              pending_update_count: int, ip_address: str | None =
                                              None, last_error_date: datetime[datetime] | None =
                                              None, last_error_message: str | None = None,
                                              last_synchronization_error_date: datetime[datetime] |
                                              None = None, max_connections: int | None = None,
                                              allowed_updates: List[str] | None = None, **extra_data:
                                              Any)
```

Describes the current status of a webhook.

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



**url: str**

Webhook URL, may be empty if webhook is not set up

**has\_custom\_certificate: bool**

True, if a custom certificate was provided for webhook certificate checks

**pending\_update\_count: int**

Number of updates awaiting delivery

**ip\_address: str | None**

*Optional.* Currently used webhook IP address

**last\_error\_date: DateTime | None**

*Optional.* Unix time for the most recent error that happened when trying to deliver an update via webhook

**last\_error\_message: str | None**

*Optional.* Error message in human-readable format for the most recent error that happened when trying to deliver an update via webhook

**last\_synchronization\_error\_date: DateTime | None**

*Optional.* Unix time of the most recent error that happened when trying to synchronize available updates with Telegram datacenters

**max\_connections: int | None**

*Optional.* The maximum allowed number of simultaneous HTTPS connections to the webhook for update delivery

**allowed\_updates: List[str] | None**

*Optional.* A list of update types the bot is subscribed to. Defaults to all update types except *chat\_member*

## Stickers

### InputSticker

```
class aiogram.types.input_sticker.InputSticker(*, sticker: InputFile | str, emoji_list: List[str],
                                              mask_position: MaskPosition | None = None,
                                              keywords: List[str] | None = None, **extra_data: Any)
```

This object describes a sticker to be added to a sticker set.

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

**sticker: InputFile | str**

The added sticker. 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, upload a new one using multipart/form-data, or pass 'attach://<file\_attach\_name>' to upload a new one using multipart/form-data under <file\_attach\_name> name. Animated and video stickers can't be uploaded via HTTP URL. [More information on Sending Files »](#)

**emoji\_list: List[str]**

List of 1-20 emoji associated with the sticker

**mask\_position: MaskPosition | None**

*Optional.* Position where the mask should be placed on faces. For 'mask' stickers only.

**keywords: List[str] | None**

*Optional.* List of 0-20 search keywords for the sticker with total length of up to 64 characters. For 'regular' and 'custom\_emoji' stickers only.

## MaskPosition

```
class aiogram.types.mask_position.MaskPosition(*, point: str, x_shift: float, y_shift: float, scale: float,
                                              **extra_data: Any)
```

This object describes the position on faces where a mask should be placed by default.

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

**point: str**

The part of the face relative to which the mask should be placed. One of 'forehead', 'eyes', 'mouth', or 'chin'.

**x\_shift: float**

Shift by X-axis measured in widths of the mask scaled to the face size, from left to right. For example, choosing -1.0 will place mask just to the left of the default mask position.

**y\_shift: float**

Shift by Y-axis measured in heights of the mask scaled to the face size, from top to bottom. For example, 1.0 will place the mask just below the default mask position.

**scale: float**

Mask scaling coefficient. For example, 2.0 means double size.

## Sticker

```
class aiogram.types.sticker.Sticker(*, file_id: str, file_unique_id: str, type: str, width: int, height: int,
                                     is_animated: bool, is_video: bool, thumbnail: PhotoSize | None = None, emoji: str | None = None, set_name: str | None = None,
                                     premium_animation: File | None = None, mask_position: MaskPosition | None = None, custom_emoji_id: str | None = None,
                                     needs_repainting: bool | None = None, file_size: int | None = None,
                                     **extra_data: Any)
```

This object represents a sticker.

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

**file\_id: str**

Identifier for this file, which can be used to download or reuse the file

**file\_unique\_id: str**

Unique identifier for this file, which is supposed to be the same over time and for different bots. Can't be used to download or reuse the file.

**type: str**

Type of the sticker, currently one of 'regular', 'mask', 'custom\_emoji'. The type of the sticker is independent from its format, which is determined by the fields *is\_animated* and *is\_video*.

**width: int**

Sticker width

**height: int**

Sticker height

**is\_animated: bool**

True, if the sticker is [animated](#)

**is\_video:** `bool`

True, if the sticker is a `video sticker`

**thumbnail:** `PhotoSize | None`

*Optional.* Sticker thumbnail in the `.WEBP` or `.JPG` format

**emoji:** `str | None`

*Optional.* Emoji associated with the sticker

**set\_name:** `str | None`

*Optional.* Name of the sticker set to which the sticker belongs

**premium\_animation:** `File | None`

*Optional.* For premium regular stickers, premium animation for the sticker

**mask\_position:** `MaskPosition | None`

*Optional.* For mask stickers, the position where the mask should be placed

**custom\_emoji\_id:** `str | None`

*Optional.* For custom emoji stickers, unique identifier of the custom emoji

**needs\_repainting:** `bool | None`

*Optional.* True, if the sticker must be repainted to a text color in messages, the color of the Telegram Premium badge in emoji status, white color on chat photos, or another appropriate color in other places

**file\_size:** `int | None`

*Optional.* File size in bytes

**set\_position\_in\_set**(*position: int, \*\*kwargs: Any*) → *SetStickerPositionInSet*

Shortcut for method `aiogram.methods.set_sticker_position_in_set.SetStickerPositionInSet` will automatically fill method attributes:

- `sticker`

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

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

#### Parameters

**position** – New sticker position in the set, zero-based

#### Returns

instance of method `aiogram.methods.set_sticker_position_in_set.SetStickerPositionInSet`

**delete\_from\_set**(*\*\*kwargs: Any*) → *DeleteStickerFromSet*

Shortcut for method `aiogram.methods.delete_sticker_from_set.DeleteStickerFromSet` will automatically fill method attributes:

- `sticker`

Use this method to delete a sticker from a set created by the bot. Returns True on success.

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

#### Returns

instance of method `aiogram.methods.delete_sticker_from_set.DeleteStickerFromSet`

## StickerSet

```
class aiogram.types.sticker_set.StickerSet(*, name: str, title: str, sticker_type: str, is_animated: bool,
                                           is_video: bool, stickers: List[Sticker], thumbnail: PhotoSize
                                           | None = None, **extra_data: Any)
```

This object represents a sticker set.

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

**name: str**

Sticker set name

**title: str**

Sticker set title

**sticker\_type: str**

Type of stickers in the set, currently one of ‘regular’, ‘mask’, ‘custom\_emoji’

**is\_animated: bool**

True, if the sticker set contains [animated stickers](#)

**is\_video: bool**

True, if the sticker set contains [video stickers](#)

**stickers: List[Sticker]**

List of all set stickers

**thumbnail: PhotoSize | None**

*Optional.* Sticker set thumbnail in the .WEBP, .TGS, or .WEBM format

## Payments

### Invoice

```
class aiogram.types.invoice.Invoice(*, title: str, description: str, start_parameter: str, currency: str,
                                    total_amount: int, **extra_data: Any)
```

This object contains basic information about an invoice.

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

**title: str**

Product name

**description: str**

Product description

**start\_parameter: str**

Unique bot deep-linking parameter that can be used to generate this invoice

**currency: str**

Three-letter ISO 4217 [currency](#) code

**total\_amount: int**

Total price in the *smallest units* of the currency (integer, **not** float/double). For example, for a price of US\$ 1.45 pass `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).

## LabeledPrice

**class** aiogram.types.labeled\_price.**LabeledPrice**(\*, label: str, amount: int, \*\*extra\_data: Any)

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

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

**label: str**

Portion label

**amount: int**

Price of the product in the *smallest units* of the [currency](#) (integer, **not** float/double). For example, for a price of US\$ 1.45 pass 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).

## OrderInfo

**class** aiogram.types.order\_info.**OrderInfo**(\*, name: str | None = None, phone\_number: str | None = None, email: str | None = None, shipping\_address: [ShippingAddress](#) | None = None, \*\*extra\_data: Any)

This object represents information about an order.

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

**name: str | None**

*Optional.* User name

**phone\_number: str | None**

*Optional.* User's phone number

**email: str | None**

*Optional.* User email

**shipping\_address: [ShippingAddress](#) | None**

*Optional.* User shipping address

## PreCheckoutQuery

**class** aiogram.types.pre\_checkout\_query.**PreCheckoutQuery**(\*, id: str, from\_user: [User](#), currency: str, total\_amount: int, invoice\_payload: str, shipping\_option\_id: str | None = None, order\_info: [OrderInfo](#) | None = None, \*\*extra\_data: Any)

This object contains information about an incoming pre-checkout query.

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

**id: str**

Unique query identifier

**from\_user: [User](#)**

User who sent the query

**currency:** `str`

Three-letter ISO 4217 [currency](#) code

**total\_amount:** `int`

Total price in the *smallest units* of the currency (integer, **not** float/double). For example, for a price of US\$ 1.45 pass 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).

**invoice\_payload:** `str`

Bot specified invoice payload

**shipping\_option\_id:** `str | None`

*Optional.* Identifier of the shipping option chosen by the user

**order\_info:** [OrderInfo](#) | `None`

*Optional.* Order information provided by the user

**answer**(*ok: bool, error\_message: str | None = None, \*\*kwargs: Any*) → [AnswerPreCheckoutQuery](#)

Shortcut for method [aiogram.methods.answer\\_pre\\_checkout\\_query](#). [AnswerPreCheckoutQuery](#) will automatically fill method attributes:

- `pre_checkout_query_id`

Once the user has confirmed their payment and shipping details, the Bot API sends the final confirmation in the form of an [aiogram.types.update.Update](#) with the field `pre_checkout_query`. Use this method to respond to such pre-checkout queries. On success, `True` is returned. **Note:** The Bot API must receive an answer within 10 seconds after the pre-checkout query was sent.

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

#### Parameters

- **ok** – 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** – 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

instance of method [aiogram.methods.answer\\_pre\\_checkout\\_query](#).  
[AnswerPreCheckoutQuery](#)

## ShippingAddress

```
class aiogram.types.shipping_address.ShippingAddress(*, country_code: str, state: str, city: str,
                                                    street_line1: str, street_line2: str, post_code:
                                                    str, **extra_data: Any)
```

This object represents a shipping address.

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

**country\_code:** `str`

Two-letter ISO 3166-1 alpha-2 country code

**state: str**  
State, if applicable

**city: str**  
City

**street\_line1: str**  
First line for the address

**street\_line2: str**  
Second line for the address

**post\_code: str**  
Address post code

### ShippingOption

```
class aiogram.types.shipping_option.ShippingOption(*, id: str, title: str, prices: List[LabeledPrice],  
                                                    **extra_data: Any)
```

This object represents one shipping option.

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

**id: str**  
Shipping option identifier

**title: str**  
Option title

**prices: List[LabeledPrice]**  
List of price portions

### ShippingQuery

```
class aiogram.types.shipping_query.ShippingQuery(*, id: str, from_user: User, invoice_payload: str,  
                                                  shipping_address: ShippingAddress, **extra_data:  
                                                  Any)
```

This object contains information about an incoming shipping query.

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

**id: str**  
Unique query identifier

**from\_user: User**  
User who sent the query

**invoice\_payload: str**  
Bot specified invoice payload

**shipping\_address: ShippingAddress**  
User specified shipping address

**answer**(*ok*: bool, *shipping\_options*: List[ShippingOption] | None = None, *error\_message*: str | None = None, *\*\*kwargs*: Any) → AnswerShippingQuery

Shortcut for method `aiogram.methods.answer_shipping_query.AnswerShippingQuery` will automatically fill method attributes:

- `shipping_query_id`

If you sent an invoice requesting a shipping address and the parameter *is\_flexible* was specified, the Bot API will send an `aiogram.types.update.Update` with a *shipping\_query* field to the bot. Use this method to reply to shipping queries. On success, `True` is returned.

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

#### Parameters

- **ok** – Pass `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** – Required if *ok* is `True`. A JSON-serialized array of available shipping options.
- **error\_message** – 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

instance of method `aiogram.methods.answer_shipping_query.AnswerShippingQuery`

## SuccessfulPayment

```
class aiogram.types.successful_payment.SuccessfulPayment(*, currency: str, total_amount: int,
                                                         invoice_payload: str,
                                                         telegram_payment_charge_id: str,
                                                         provider_payment_charge_id: str,
                                                         shipping_option_id: str | None = None,
                                                         order_info: OrderInfo | None = None,
                                                         **extra_data: Any)
```

This object contains basic information about a successful payment.

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

**currency:** str

Three-letter ISO 4217 currency code

**total\_amount:** int

Total price in the *smallest units* of the currency (integer, **not** float/double). For example, for a price of US\$ 1.45 pass 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).

**invoice\_payload:** str

Bot specified invoice payload

**telegram\_payment\_charge\_id:** str

Telegram payment identifier

**provider\_payment\_charge\_id:** str

Provider payment identifier



**shipping\_option\_id:** `str` | `None`

*Optional.* Identifier of the shipping option chosen by the user

**order\_info:** `OrderInfo` | `None`

*Optional.* Order information provided by the user

## Games

### CallbackGame

**class** `aiogram.types.callback_game.CallbackGame`(\*\**extra\_data*: *Any*)

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

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

### Game

**class** `aiogram.types.game.Game`(\*, *title*: *str*, *description*: *str*, *photo*: *List*[`PhotoSize`], *text*: *str* | *None* = *None*, *text\_entities*: *List*[`MessageEntity`] | *None* = *None*, *animation*: `Animation` | *None* = *None*, \*\**extra\_data*: *Any*)

This object represents a game. Use `BotFather` to create and edit games, their short names will act as unique identifiers.

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

**title:** `str`

Title of the game

**description:** `str`

Description of the game

**photo:** `List`[`PhotoSize`]

Photo that will be displayed in the game message in chats.

**text:** `str` | `None`

*Optional.* Brief description of the game or high scores included in the game message. Can be automatically edited to include current high scores for the game when the bot calls `aiogram.methods.set_game_score.SetGameScore`, or manually edited using `aiogram.methods.edit_message_text.EditMessageText`. 0-4096 characters.

**text\_entities:** `List`[`MessageEntity`] | `None`

*Optional.* Special entities that appear in *text*, such as usernames, URLs, bot commands, etc.

**animation:** `Animation` | `None`

*Optional.* Animation that will be displayed in the game message in chats. Upload via `BotFather`

## GameHighScore

```
class aiogram.types.game_high_score.GameHighScore(*, position: int, user: User, score: int,
                                                    **extra_data: Any)
```

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 <https://core.telegram.org/bots/faq> **Bot FAQ** »

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

**position:** `int`

Position in high score table for the game

**user:** `User`

User

**score:** `int`

Score

## 2.3.4 Methods

Here is list of all available API methods:

### Available methods

#### answerCallbackQuery

Returns: `bool`

```
class aiogram.methods.answer_callback_query.AnswerCallbackQuery(*, callback_query_id: str, text:
                                                                str | None = None, show_alert:
                                                                bool | None = None, url: str |
                                                                None = None, cache_time: int |
                                                                None = None, **extra_data:
                                                                Any)
```

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. On success, `True` is returned.

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>

**callback\_query\_id:** `str`

Unique identifier for the query to be answered

**text:** `str | None`

Text of the notification. If not specified, nothing will be shown to the user, 0-200 characters

**show\_alert:** `bool | None`

If `True`, an alert will be shown by the client instead of a notification at the top of the chat screen. Defaults to `false`.

**url:** `str | None`

URL that will be opened by the user's client. If you have created a `aiogram.types.game.Game` and accepted the conditions via `@BotFather`, specify the URL that opens your game - note that this will only work if the query comes from a `https://core.telegram.org/bots/api#inlinekeyboardbutton` `callback_game` button.

**cache\_time:** `int | None`

The maximum amount of time in seconds that the result of the callback query may be cached client-side. Telegram apps will support caching starting in version 3.14. Defaults to 0.

## Usage

### As bot method

```
result: bool = await bot.answer_callback_query(...)
```

### Method as object

Imports:

- `from aiogram.methods.answer_callback_query import AnswerCallbackQuery`
- `alias: from aiogram.methods import AnswerCallbackQuery`

### With specific bot

```
result: bool = await bot(AnswerCallbackQuery(...))
```

### As reply into Webhook in handler

```
return AnswerCallbackQuery(...)
```

### As shortcut from received object

- `aiogram.types.callback_query.CallbackQuery.answer()`

## approveChatJoinRequest

Returns: `bool`

```
class aiogram.methods.approve_chat_join_request.ApproveChatJoinRequest(*, chat_id: int | str,
                                                                    user_id: int,
                                                                    **extra_data: Any)
```

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

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

**chat\_id:** `int | str`

Unique identifier for the target chat or username of the target channel (in the format @channelusername)

**user\_id:** `int`

Unique identifier of the target user

## Usage

### As bot method

```
result: bool = await bot.approve_chat_join_request(...)
```

### Method as object

Imports:

- `from aiogram.methods.approve_chat_join_request import ApproveChatJoinRequest`
- `alias: from aiogram.methods import ApproveChatJoinRequest`

### With specific bot

```
result: bool = await bot(ApproveChatJoinRequest(...))
```

### As reply into Webhook in handler

```
return ApproveChatJoinRequest(...)
```

### As shortcut from received object

- `aiogram.types.chat_join_request.ChatJoinRequest.approve()`

## banChatMember

Returns: `bool`

```
class aiogram.methods.ban_chat_member.BanChatMember(*, chat_id: int | str, user_id: int, until_date:
    datetime | timedelta | int | None = None,
    revoke_messages: bool | None = None,
    **extra_data: Any)
```

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 administrator rights. Returns `True` on success.

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

**chat\_id:** `int | str`

Unique identifier for the target group or username of the target supergroup or channel (in the format @channelusername)

**user\_id:** `int`

Unique identifier of the target user

**until\_date:** `datetime.datetime | datetime.timedelta | int | 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:** `bool | None`

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.

## Usage

### As bot method

```
result: bool = await bot.ban_chat_member(...)
```

### Method as object

Imports:

- `from aiogram.methods.ban_chat_member import BanChatMember`
- `alias: from aiogram.methods import BanChatMember`

### With specific bot

```
result: bool = await bot(BanChatMember(...))
```

### As reply into Webhook in handler

```
return BanChatMember(...)
```

### As shortcut from received object

- `aiogram.types.chat.Chat.ban()`

## banChatSenderChat

Returns: bool

**class** aiogram.methods.ban\_chat\_sender\_chat.**BanChatSenderChat**(\**chat\_id: int | str, sender\_chat\_id: int, \*\*extra\_data: Any*)

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

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

**chat\_id: int | str**

Unique identifier for the target chat or username of the target channel (in the format @channelusername)

**sender\_chat\_id: int**

Unique identifier of the target sender chat

## Usage

### As bot method

```
result: bool = await bot.ban_chat_sender_chat(...)
```

### Method as object

Imports:

- `from aiogram.methods.ban_chat_sender_chat import BanChatSenderChat`
- `alias: from aiogram.methods import BanChatSenderChat`

### With specific bot

```
result: bool = await bot(BanChatSenderChat(...))
```

### As reply into Webhook in handler

```
return BanChatSenderChat(...)
```

### As shortcut from received object

- `aiogram.types.chat.Chat.ban_sender_chat()`

### close

Returns: bool

**class** aiogram.methods.close.Close(\*\*extra\_data: Any)

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>

### Usage

#### As bot method

```
result: bool = await bot.close(...)
```

#### Method as object

Imports:

- `from aiogram.methods.close import Close`
- `alias: from aiogram.methods import Close`

#### With specific bot

```
result: bool = await bot(Close(...))
```

#### As reply into Webhook in handler

```
return Close(...)
```

## closeForumTopic

Returns: bool

```
class aiogram.methods.close_forum_topic.CloseForumTopic(*, chat_id: int | str, message_thread_id:
                                                         int, **extra_data: Any)
```

Use this method to close an open topic in a forum supergroup chat. The bot must be an administrator in the chat for this to work and must have the *can\_manage\_topics* administrator rights, unless it is the creator of the topic. Returns True on success.

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

**chat\_id: int | str**

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

**message\_thread\_id: int**

Unique identifier for the target message thread of the forum topic

## Usage

### As bot method

```
result: bool = await bot.close_forum_topic(...)
```

### Method as object

Imports:

- `from aiogram.methods.close_forum_topic import CloseForumTopic`
- `alias: from aiogram.methods import CloseForumTopic`

### With specific bot

```
result: bool = await bot(CloseForumTopic(...))
```

### As reply into Webhook in handler

```
return CloseForumTopic(...)
```



## closeGeneralForumTopic

Returns: bool

```
class aiogram.methods.close_general_forum_topic.CloseGeneralForumTopic(*, chat_id: int | str,
                                                                    **extra_data: Any)
```

Use this method to close an open ‘General’ topic in a forum supergroup chat. The bot must be an administrator in the chat for this to work and must have the *can\_manage\_topics* administrator rights. Returns True on success.

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

**chat\_id:** int | str

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

## Usage

### As bot method

```
result: bool = await bot.close_general_forum_topic(...)
```

### Method as object

Imports:

- from aiogram.methods.close\_general\_forum\_topic import CloseGeneralForumTopic
- alias: from aiogram.methods import CloseGeneralForumTopic

### With specific bot

```
result: bool = await bot(CloseGeneralForumTopic(...))
```

### As reply into Webhook in handler

```
return CloseGeneralForumTopic(...)
```

## copyMessage

Returns: MessageId

```
class aiogram.methods.copy_message.CopyMessage(*, chat_id: int | str, from_chat_id: int | str,
                                                message_id: int, message_thread_id: int | None =
                                                None, caption: str | None = None, parse_mode: str |
                                                None = sentinel.UNSET_PARSE_MODE,
                                                caption_entities: List[MessageEntity] | None = None,
                                                disable_notification: bool | None = None,
                                                protect_content: bool | None =
                                                sentinel.UNSET_PROTECT_CONTENT,
                                                reply_to_message_id: int | None = None,
                                                allow_sending_without_reply: bool | None = None,
                                                reply_markup: InlineKeyboardMarkup |
                                                ReplyKeyboardMarkup | ReplyKeyboardRemove |
                                                ForceReply | None = None, **extra_data: Any)
```

Use this method to copy messages of any kind. Service messages and invoice messages can't be copied. A quiz `aiogram.methods.poll.Poll` can be copied only if the value of the field `correct_option_id` is known to the bot. The method is analogous to the method `aiogram.methods.forward_message.ForwardMessage`, but the copied message doesn't have a link to the original message. Returns the `aiogram.types.message_id.MessageId` of the sent message on success.

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

**chat\_id: int | str**

Unique identifier for the target chat or username of the target channel (in the format @channelusername)

**from\_chat\_id: int | str**

Unique identifier for the chat where the original message was sent (or channel username in the format @channelusername)

**message\_id: int**

Message identifier in the chat specified in `from_chat_id`

**message\_thread\_id: int | None**

Unique identifier for the target message thread (topic) of the forum; for forum supergroups only

**caption: str | None**

New caption for media, 0-1024 characters after entities parsing. If not specified, the original caption is kept

**parse\_mode: str | None**

Mode for parsing entities in the new caption. See [formatting options](#) for more details.

**caption\_entities: List[MessageEntity] | None**

A JSON-serialized list of special entities that appear in the new caption, which can be specified instead of `parse_mode`

**disable\_notification: bool | None**

Sends the message [silently](#). Users will receive a notification with no sound.

**protect\_content: bool | None**

Protects the contents of the sent message from forwarding and saving

**reply\_to\_message\_id: int | None**

If the message is a reply, ID of the original message

**allow\_sending\_without\_reply: bool | None**

Pass True if the message should be sent even if the specified replied-to message is not found

**reply\_markup:** *InlineKeyboardMarkup* | *ReplyKeyboardMarkup* | *ReplyKeyboardRemove* | *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.

## Usage

### As bot method

```
result: MessageId = await bot.copy_message(...)
```

### Method as object

Imports:

- `from aiogram.methods.copy_message import CopyMessage`
- `alias: from aiogram.methods import CopyMessage`

### With specific bot

```
result: MessageId = await bot(CopyMessage(...))
```

### As reply into Webhook in handler

```
return CopyMessage(...)
```

### As shortcut from received object

- `aiogram.types.message.Message.copy_to()`

### createChatInviteLink

Returns: `ChatInviteLink`

```
class aiogram.methods.create_chat_invite_link.CreateChatInviteLink(*, chat_id: int | str, name: str | None = None,
                                                                    expire_date: datetime |
                                                                    timedelta | int | None = None,
                                                                    member_limit: int | None =
                                                                    None, creates_join_request:
                                                                    bool | None = None,
                                                                    **extra_data: Any)
```

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 administrator rights. The link can be revoked using the method

*aiogram.methods.revoke\_chat\_invite\_link.RevokeChatInviteLink*. Returns the new invite link as *aiogram.types.chat\_invite\_link.ChatInviteLink* object.

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

**chat\_id:** `int | str`

Unique identifier for the target chat or username of the target channel (in the format @channelusername)

**name:** `str | None`

Invite link name; 0-32 characters

**expire\_date:** `datetime.datetime | datetime.timedelta | int | None`

Point in time (Unix timestamp) when the link will expire

**member\_limit:** `int | None`

The maximum number of users that can be members of the chat simultaneously after joining the chat via this invite link; 1-99999

**creates\_join\_request:** `bool | None`

True, if users joining the chat via the link need to be approved by chat administrators. If True, *member\_limit* can't be specified

## Usage

### As bot method

```
result: ChatInviteLink = await bot.create_chat_invite_link(...)
```

### Method as object

Imports:

- `from aiogram.methods.create_chat_invite_link import CreateChatInviteLink`
- `alias: from aiogram.methods import CreateChatInviteLink`

### With specific bot

```
result: ChatInviteLink = await bot(CreateChatInviteLink(...))
```

### As reply into Webhook in handler

```
return CreateChatInviteLink(...)
```

### As shortcut from received object

- `aiogram.types.chat.Chat.create_invite_link()`

### createForumTopic

Returns: `ForumTopic`

```
class aiogram.methods.create_forum_topic.CreateForumTopic(*chat_id: int | str, name: str,
icon_color: int | None = None,
icon_custom_emoji_id: str | None =
None, **extra_data: Any)
```

Use this method to create a topic in a forum supergroup chat. The bot must be an administrator in the chat for this to work and must have the *can\_manage\_topics* administrator rights. Returns information about the created topic as a `aiogram.types.forum_topic.ForumTopic` object.

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

**chat\_id: int | str**

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

**name: str**

Topic name, 1-128 characters

**icon\_color: int | None**

Color of the topic icon in RGB format. Currently, must be one of 7322096 (0x6FB9F0), 16766590 (0xFFD67E), 13338331 (0xCB86DB), 9367192 (0x8EEE98), 16749490 (0xFF93B2), or 16478047 (0xFB6F5F)

**icon\_custom\_emoji\_id: str | None**

Unique identifier of the custom emoji shown as the topic icon. Use `aiogram.methods.get_forum_topic_icon_stickers.GetForumTopicIconStickers` to get all allowed custom emoji identifiers.

### Usage

#### As bot method

```
result: ForumTopic = await bot.create_forum_topic(...)
```

### Method as object

Imports:

- `from aiogram.methods.create_forum_topic import CreateForumTopic`
- `alias: from aiogram.methods import CreateForumTopic`

### With specific bot

```
result: ForumTopic = await bot(CreateForumTopic(...))
```

### As reply into Webhook in handler

```
return CreateForumTopic(...)
```

## declineChatJoinRequest

Returns: bool

```
class aiogram.methods.decline_chat_join_request.DeclineChatJoinRequest(*, chat_id: int | str,
                                                                    user_id: int,
                                                                    **extra_data: Any)
```

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

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

**chat\_id: int | str**

Unique identifier for the target chat or username of the target channel (in the format @channelusername)

**user\_id: int**

Unique identifier of the target user

## Usage

### As bot method

```
result: bool = await bot.decline_chat_join_request(...)
```

## Method as object

Imports:

- `from aiogram.methods.decline_chat_join_request import DeclineChatJoinRequest`
- `alias: from aiogram.methods import DeclineChatJoinRequest`

### With specific bot

```
result: bool = await bot(DenclineChatJoinRequest(...))
```

### As reply into Webhook in handler

```
return DenclineChatJoinRequest(...)
```

### As shortcut from received object

- `aiogram.types.chat_join_request.ChatJoinRequest.decline()`

## deleteChatPhoto

Returns: bool

**class** aiogram.methods.delete\_chat\_photo.**DeleteChatPhoto**(\**chat\_id*: int | str, \*\**extra\_data*: Any)

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 administrator rights. Returns True on success.

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

**chat\_id**: int | str

Unique identifier for the target chat or username of the target channel (in the format @channelusername)

### Usage

#### As bot method

```
result: bool = await bot.delete_chat_photo(...)
```

### Method as object

Imports:

- `from aiogram.methods.delete_chat_photo import DeleteChatPhoto`
- `alias: from aiogram.methods import DeleteChatPhoto`

### With specific bot

```
result: bool = await bot(DeleteChatPhoto(...))
```

### As reply into Webhook in handler

```
return DeleteChatPhoto(...)
```

### As shortcut from received object

- `aiogram.types.chat.Chat.delete_photo()`

### deleteChatStickerSet

Returns: bool

```
class aiogram.methods.delete_chat_sticker_set.DeleteChatStickerSet(*, chat_id: int | str,
                                                                    **extra_data: Any)
```

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 administrator rights. Use the field `can_set_sticker_set` optionally returned in `aiogram.methods.get_chat.GetChat` requests to check if the bot can use this method. Returns True on success.

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

**chat\_id: int | str**

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

### Usage

#### As bot method

```
result: bool = await bot.delete_chat_sticker_set(...)
```

### Method as object

Imports:

- `from aiogram.methods.delete_chat_sticker_set import DeleteChatStickerSet`
- `alias: from aiogram.methods import DeleteChatStickerSet`



### With specific bot

```
result: bool = await bot(DeleteChatStickerSet(...))
```

### As reply into Webhook in handler

```
return DeleteChatStickerSet(...)
```

### As shortcut from received object

- `aiogram.types.chat.Chat.delete_sticker_set()`

## deleteForumTopic

Returns: bool

```
class aiogram.methods.delete_forum_topic.DeleteForumTopic(*, chat_id: int | str, message_thread_id:
                                                         int, **extra_data: Any)
```

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

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

**chat\_id: int | str**

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

**message\_thread\_id: int**

Unique identifier for the target message thread of the forum topic

## Usage

### As bot method

```
result: bool = await bot.delete_forum_topic(...)
```

### Method as object

Imports:

- `from aiogram.methods.delete_forum_topic import DeleteForumTopic`
- `alias: from aiogram.methods import DeleteForumTopic`

### With specific bot

```
result: bool = await bot>DeleteForumTopic(...)
```

### As reply into Webhook in handler

```
return>DeleteForumTopic(...)
```

## deleteMyCommands

Returns: bool

```
class aiogram.methods.delete_my_commands.DeleteMyCommands(*, scope: BotCommandScopeDefault |  
    BotCommandScopeAllPrivateChats |  
    BotCommandScopeAllGroupChats |  
    BotCommandScopeAllChatAdministrators | BotCommandScopeChat |  
    BotCommandScopeChatAdministrators |  
    BotCommandScopeChatMember | None  
    = None, language_code: str | None =  
    None, **extra_data: Any)
```

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. Returns True on success.

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

**scope:** *BotCommandScopeDefault* | *BotCommandScopeAllPrivateChats* |  
*BotCommandScopeAllGroupChats* | *BotCommandScopeAllChatAdministrators* |  
*BotCommandScopeChat* | *BotCommandScopeChatAdministrators* | *BotCommandScopeChatMember*  
| *None*

A JSON-serialized object, describing scope of users for which the commands are relevant. Defaults to *aiogram.types.bot\_command\_scope\_default.BotCommandScopeDefault*.

**language\_code:** *str* | *None*

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

## Usage

### As bot method

```
result: bool = await bot.delete_my_commands(...)
```

## Method as object

Imports:

- `from aiogram.methods.delete_my_commands import DeleteMyCommands`
- `alias: from aiogram.methods import DeleteMyCommands`

## With specific bot

```
result: bool = await bot(DeleteMyCommands(...))
```

## As reply into Webhook in handler

```
return DeleteMyCommands(...)
```

## editChatInviteLink

Returns: `ChatInviteLink`

```
class aiogram.methods.edit_chat_invite_link.EditChatInviteLink(*, chat_id: int | str, invite_link: str, name: str | None = None, expire_date: datetime | timedelta | int | None = None, member_limit: int | None = None, creates_join_request: bool | None = None, **extra_data: Any)
```

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 administrator rights. Returns the edited invite link as a `aiogram.types.chat_invite_link.ChatInviteLink` object.

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

**chat\_id:** `int | str`

Unique identifier for the target chat or username of the target channel (in the format @channelusername)

**invite\_link:** `str`

The invite link to edit

**name:** `str | None`

Invite link name; 0-32 characters

**expire\_date:** `datetime.datetime | datetime.timedelta | int | None`

Point in time (Unix timestamp) when the link will expire

**member\_limit:** `int | None`

The maximum number of users that can be members of the chat simultaneously after joining the chat via this invite link; 1-99999

**creates\_join\_request:** `bool | None`

True, if users joining the chat via the link need to be approved by chat administrators. If True, `member_limit` can't be specified

## Usage

### As bot method

```
result: ChatInviteLink = await bot.edit_chat_invite_link(...)
```

### Method as object

Imports:

- `from aiogram.methods.edit_chat_invite_link import EditChatInviteLink`
- `alias: from aiogram.methods import EditChatInviteLink`

### With specific bot

```
result: ChatInviteLink = await bot(EditChatInviteLink(...))
```

### As reply into Webhook in handler

```
return EditChatInviteLink(...)
```

### As shortcut from received object

- `aiogram.types.chat.Chat.edit_invite_link()`

## editForumTopic

Returns: bool

```
class aiogram.methods.edit_forum_topic.EditForumTopic(*, chat_id: int | str, message_thread_id: int,
                                                         name: str | None = None,
                                                         icon_custom_emoji_id: str | None = None,
                                                         **extra_data: Any)
```

Use this method to edit name and icon of a topic in a forum supergroup chat. The bot must be an administrator in the chat for this to work and must have *can\_manage\_topics* administrator rights, unless it is the creator of the topic. Returns True on success.

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

**chat\_id: int | str**

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

**message\_thread\_id: int**

Unique identifier for the target message thread of the forum topic

**name:** `str | None`

New topic name, 0-128 characters. If not specified or empty, the current name of the topic will be kept

**icon\_custom\_emoji\_id:** `str | None`

New unique identifier of the custom emoji shown as the topic icon. Use `aiogram.methods.get_forum_topic_icon_stickers.GetForumTopicIconStickers` to get all allowed custom emoji identifiers. Pass an empty string to remove the icon. If not specified, the current icon will be kept

## Usage

### As bot method

```
result: bool = await bot.edit_forum_topic(...)
```

### Method as object

Imports:

- `from aiogram.methods.edit_forum_topic import EditForumTopic`
- `alias: from aiogram.methods import EditForumTopic`

### With specific bot

```
result: bool = await bot(EditForumTopic(...))
```

### As reply into Webhook in handler

```
return EditForumTopic(...)
```

## editGeneralForumTopic

Returns: `bool`

```
class aiogram.methods.edit_general_forum_topic.EditGeneralForumTopic(*, chat_id: int | str, name: str, **extra_data: Any)
```

Use this method to edit the name of the ‘General’ topic in a forum supergroup chat. The bot must be an administrator in the chat for this to work and must have `can_manage_topics` administrator rights. Returns True on success.

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

**chat\_id:** `int | str`

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

**name:** `str`

New topic name, 1-128 characters

## Usage

### As bot method

```
result: bool = await bot.edit_general_forum_topic(...)
```

### Method as object

Imports:

- `from aiogram.methods.edit_general_forum_topic import EditGeneralForumTopic`
- `alias: from aiogram.methods import EditGeneralForumTopic`

### With specific bot

```
result: bool = await bot(EditGeneralForumTopic(...))
```

### As reply into Webhook in handler

```
return EditGeneralForumTopic(...)
```

## exportChatInviteLink

Returns: `str`

```
class aiogram.methods.export_chat_invite_link.ExportChatInviteLink(*, chat_id: int | str,
                                                                    **extra_data: Any)
```

Use this method to generate a new primary invite link for a chat; any previously generated primary link is revoked. The bot must be an administrator in the chat for this to work and must have the appropriate administrator rights. Returns the new invite link as *String* on success.

Note: Each administrator in a chat generates their own invite links. Bots can't use invite links generated by other administrators. If you want your bot to work with invite links, it will need to generate its own link using `aiogram.methods.export_chat_invite_link.ExportChatInviteLink` or by calling the `aiogram.methods.get_chat.GetChat` method. If your bot needs to generate a new primary invite link replacing its previous one, use `aiogram.methods.export_chat_invite_link.ExportChatInviteLink` again.

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

**chat\_id:** `int | str`

Unique identifier for the target chat or username of the target channel (in the format @channelusername)

## Usage

### As bot method

```
result: str = await bot.export_chat_invite_link(...)
```

### Method as object

Imports:

- `from aiogram.methods.export_chat_invite_link import ExportChatInviteLink`
- `alias: from aiogram.methods import ExportChatInviteLink`

### With specific bot

```
result: str = await bot(ExportChatInviteLink(...))
```

### As reply into Webhook in handler

```
return ExportChatInviteLink(...)
```

### As shortcut from received object

- `aiogram.types.chat.Chat.export_invite_link()`

## forwardMessage

Returns: Message

```
class aiogram.methods.forward_message.ForwardMessage(*, chat_id: int | str, from_chat_id: int | str,
    message_id: int, message_thread_id: int | None = None, disable_notification: bool | None =
    None, protect_content: bool | None = sentinel.UNSET_PROTECT_CONTENT,
    **extra_data: Any)
```

Use this method to forward messages of any kind. Service messages can't be forwarded. On success, the sent `aiogram.types.message.Message` is returned.

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

**chat\_id: int | str**

Unique identifier for the target chat or username of the target channel (in the format @channelusername)

**from\_chat\_id: int | str**

Unique identifier for the chat where the original message was sent (or channel username in the format @channelusername)

**message\_id:** `int`

Message identifier in the chat specified in *from\_chat\_id*

**message\_thread\_id:** `int | None`

Unique identifier for the target message thread (topic) of the forum; for forum supergroups only

**disable\_notification:** `bool | None`

Sends the message [silently](#). Users will receive a notification with no sound.

**protect\_content:** `bool | None`

Protects the contents of the forwarded message from forwarding and saving

## Usage

### As bot method

```
result: Message = await bot.forward_message(...)
```

### Method as object

Imports:

- `from aiogram.methods.forward_message import ForwardMessage`
- `alias: from aiogram.methods import ForwardMessage`

### With specific bot

```
result: Message = await bot(ForwardMessage(...))
```

### As reply into Webhook in handler

```
return ForwardMessage(...)
```

### As shortcut from received object

- `aiogram.types.message.Message.forward()`



## getChat

Returns: Chat

**class** aiogram.methods.get\_chat.**GetChat**(\*, chat\_id: int | str, \*\*extra\_data: Any)

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.). Returns a *aiogram.types.chat.Chat* object on success.

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

**chat\_id: int | str**

Unique identifier for the target chat or username of the target supergroup or channel (in the format @channelusername)

## Usage

### As bot method

```
result: Chat = await bot.get_chat(...)
```

## Method as object

Imports:

- from aiogram.methods.get\_chat import GetChat
- alias: from aiogram.methods import GetChat

### With specific bot

```
result: Chat = await bot(GetChat(...))
```

## getChatAdministrators

Returns: List[Union[ChatMemberOwner, ChatMemberAdministrator, ChatMemberMember, ChatMemberRestricted, ChatMemberLeft, ChatMemberBanned]]

**class** aiogram.methods.get\_chat\_administrators.**GetChatAdministrators**(\*, chat\_id: int | str, \*\*extra\_data: Any)

Use this method to get a list of administrators in a chat, which aren't bots. Returns an Array of *aiogram.types.chat\_member.ChatMember* objects.

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

**chat\_id: int | str**

Unique identifier for the target chat or username of the target supergroup or channel (in the format @channelusername)

## Usage

### As bot method

```
result: List[Union[ChatMemberOwner, ChatMemberAdministrator, ChatMemberMember,
↳ ChatMemberRestricted, ChatMemberLeft, ChatMemberBanned]] = await bot.get_chat_
↳ administrators(...)
```

### Method as object

Imports:

- `from aiogram.methods.get_chat_administrators import GetChatAdministrators`
- `alias: from aiogram.methods import GetChatAdministrators`

### With specific bot

```
result: List[Union[ChatMemberOwner, ChatMemberAdministrator, ChatMemberMember,
↳ ChatMemberRestricted, ChatMemberLeft, ChatMemberBanned]] = await
↳ bot(GetChatAdministrators(...))
```

### As shortcut from received object

- `aiogram.types.chat.Chat.get_administrators()`

## getChatMember

Returns: `Union[ChatMemberOwner, ChatMemberAdministrator, ChatMemberMember, ChatMemberRestricted, ChatMemberLeft, ChatMemberBanned]`

```
class aiogram.methods.get_chat_member.GetChatMember(*, chat_id: int | str, user_id: int, **extra_data:
Any)
```

Use this method to get information about a member of a chat. The method is only guaranteed to work for other users if the bot is an administrator in the chat. Returns a `aiogram.types.chat_member.ChatMember` object on success.

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

**chat\_id: int | str**

Unique identifier for the target chat or username of the target supergroup or channel (in the format @channelusername)

**user\_id: int**

Unique identifier of the target user

## Usage

### As bot method

```
result: Union[ChatMemberOwner, ChatMemberAdministrator, ChatMemberMember,
↳ ChatMemberRestricted, ChatMemberLeft, ChatMemberBanned] = await bot.get_chat_member(...
↳ )
```

### Method as object

Imports:

- `from aiogram.methods.get_chat_member import GetChatMember`
- `alias: from aiogram.methods import GetChatMember`

### With specific bot

```
result: Union[ChatMemberOwner, ChatMemberAdministrator, ChatMemberMember,
↳ ChatMemberRestricted, ChatMemberLeft, ChatMemberBanned] = await bot(GetChatMember(...))
```

### As shortcut from received object

- `aiogram.types.chat.Chat.get_member()`

## getChatMemberCount

Returns: `int`

```
class aiogram.methods.get_chat_member_count.GetChatMemberCount(*, chat_id: int | str, **extra_data:
Any)
```

Use this method to get the number of members in a chat. Returns *Int* on success.

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

**chat\_id: int | str**

Unique identifier for the target chat or username of the target supergroup or channel (in the format @channelusername)

## Usage

### As bot method

```
result: int = await bot.get_chat_member_count(...)
```

## Method as object

Imports:

- `from aiogram.methods.get_chat_member_count import GetChatMemberCount`
- `alias: from aiogram.methods import GetChatMemberCount`

## With specific bot

```
result: int = await bot(GetChatMemberCount(...))
```

## As shortcut from received object

- `aiogram.types.chat.Chat.get_member_count()`

## getChatMenuButton

Returns: `Union[MenuButtonDefault, MenuButtonWebApp, MenuButtonCommands]`

```
class aiogram.methods.get_chat_menu_button.GetChatMenuButton(*, chat_id: int | None = None,
                                                             **extra_data: Any)
```

Use this method to get the current value of the bot's menu button in a private chat, or the default menu button. Returns `aiogram.types.menu_button.MenuButton` on success.

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

**chat\_id: int | None**

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

## Usage

### As bot method

```
result: Union[MenuButtonDefault, MenuButtonWebApp, MenuButtonCommands] = await bot.get_
    ↪ chat_menu_button(...)
```

## Method as object

Imports:

- `from aiogram.methods.get_chat_menu_button import GetChatMenuButton`
- `alias: from aiogram.methods import GetChatMenuButton`

### With specific bot

```
result: Union[MenuButtonDefault, MenuButtonWebApp, MenuButtonCommands] = await_
↳ bot(GetChatMenuButton(...))
```

### getFile

Returns: File

**class** aiogram.methods.get\_file.**GetFile**(\*, file\_id: str, \*\*extra\_data: Any)

Use this method to get basic information about a file and prepare it for downloading. For the moment, bots can download files of up to 20MB in size. On success, a *aiogram.types.file.File* object is returned. The file can then be downloaded via the link [https://api.telegram.org/file/bot<token>/<file\\_path>](https://api.telegram.org/file/bot<token>/<file_path>), where *<file\_path>* is taken from the response. 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 *aiogram.methods.get\_file.GetFile* again. **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>

**file\_id:** str

File identifier to get information about

### Usage

#### As bot method

```
result: File = await bot.get_file(...)
```

#### Method as object

Imports:

- `from aiogram.methods.get_file import GetFile`
- `alias: from aiogram.methods import GetFile`

### With specific bot

```
result: File = await bot(GetFile(...))
```

## getForumTopicIconStickers

Returns: List[Sticker]

**class** aiogram.methods.get\_forum\_topic\_icon\_stickers.**GetForumTopicIconStickers**(\*\**extra\_data*: Any)

Use this method to get custom emoji stickers, which can be used as a forum topic icon by any user. Requires no parameters. Returns an Array of *aiogram.types.sticker.Sticker* objects.

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

## Usage

### As bot method

```
result: List[Sticker] = await bot.get_forum_topic_icon_stickers(...)
```

### Method as object

Imports:

- from aiogram.methods.get\_forum\_topic\_icon\_stickers import GetForumTopicIconStickers
- alias: from aiogram.methods import GetForumTopicIconStickers

### With specific bot

```
result: List[Sticker] = await bot(GetForumTopicIconStickers(...))
```

## getMe

Returns: User

**class** aiogram.methods.get\_me.**GetMe**(\*\**extra\_data*: Any)

A simple method for testing your bot's authentication token. Requires no parameters. Returns basic information about the bot in form of a *aiogram.types.user.User* object.

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

## Usage

### As bot method

```
result: User = await bot.get_me(...)
```

## Method as object

Imports:

- `from aiogram.methods.get_me import GetMe`
- `alias: from aiogram.methods import GetMe`

## With specific bot

```
result: User = await bot(GetMe(...))
```

## getMyCommands

Returns: `List[BotCommand]`

```
class aiogram.methods.get_my_commands.GetMyCommands(*, scope: BotCommandScopeDefault |
    BotCommandScopeAllPrivateChats |
    BotCommandScopeAllGroupChats |
    BotCommandScopeAllChatAdministrators |
    BotCommandScopeChat |
    BotCommandScopeChatAdministrators |
    BotCommandScopeChatMember | None = None,
    language_code: str | None = None,
    **extra_data: Any)
```

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

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

**scope:** *BotCommandScopeDefault* | *BotCommandScopeAllPrivateChats* |  
*BotCommandScopeAllGroupChats* | *BotCommandScopeAllChatAdministrators* |  
*BotCommandScopeChat* | *BotCommandScopeChatAdministrators* | *BotCommandScopeChatMember*  
 | *None*

A JSON-serialized object, describing scope of users. Defaults to *aiogram.types.bot\_command\_scope\_default.BotCommandScopeDefault*.

**language\_code:** *str* | *None*

A two-letter ISO 639-1 language code or an empty string

## Usage

### As bot method

```
result: List[BotCommand] = await bot.get_my_commands(...)
```

## Method as object

Imports:

- `from aiogram.methods.get_my_commands import GetMyCommands`
- `alias: from aiogram.methods import GetMyCommands`

## With specific bot

```
result: List[BotCommand] = await bot(GetMyCommands(...))
```

## getMyDefaultAdministratorRights

Returns: `ChatAdministratorRights`

```
class aiogram.methods.get_my_default_administrator_rights.GetMyDefaultAdministratorRights(*,
                                                                                          for_channels:
                                                                                          bool
                                                                                          |
                                                                                          None
                                                                                          =
                                                                                          None,
                                                                                          **extra_data:
                                                                                          Any)
```

Use this method to get the current default administrator rights of the bot. Returns *aiogram.types.chat\_administrator\_rights.ChatAdministratorRights* on success.

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

**for\_channels:** `bool` | `None`

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

## Usage

### As bot method

```
result: ChatAdministratorRights = await bot.get_my_default_administrator_rights(...)
```



## Method as object

Imports:

- `from aiogram.methods.get_my_default_administrator_rights import GetMyDefaultAdministratorRights`
- `alias: from aiogram.methods import GetMyDefaultAdministratorRights`

## With specific bot

```
result: ChatAdministratorRights = await bot(GetMyDefaultAdministratorRights(...))
```

## getMyDescription

Returns: BotDescription

```
class aiogram.methods.get_my_description.GetMyDescription(*, language_code: str | None = None,
                                                         **extra_data: Any)
```

Use this method to get the current bot description for the given user language. Returns *aiogram.types.bot\_description.BotDescription* on success.

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

**language\_code:** `str | None`

A two-letter ISO 639-1 language code or an empty string

## Usage

### As bot method

```
result: BotDescription = await bot.get_my_description(...)
```

## Method as object

Imports:

- `from aiogram.methods.get_my_description import GetMyDescription`
- `alias: from aiogram.methods import GetMyDescription`

### With specific bot

```
result: BotDescription = await bot(GetMyDescription(...))
```

### getMyName

Returns: BotName

**class** aiogram.methods.get\_my\_name.**GetMyName**(\*, language\_code: str | None = None, \*\*extra\_data: Any)  
Use this method to get the current bot name for the given user language. Returns *aiogram.types.bot\_name.BotName* on success.

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

**language\_code:** str | None

A two-letter ISO 639-1 language code or an empty string

### Usage

#### As bot method

```
result: BotName = await bot.get_my_name(...)
```

### Method as object

Imports:

- from aiogram.methods.get\_my\_name import GetMyName
- alias: from aiogram.methods import GetMyName

### With specific bot

```
result: BotName = await bot(GetMyName(...))
```

### getMyShortDescription

Returns: BotShortDescription

**class** aiogram.methods.get\_my\_short\_description.**GetMyShortDescription**(\*, language\_code: str | None = None, \*\*extra\_data: Any)

Use this method to get the current bot short description for the given user language. Returns *aiogram.types.bot\_short\_description.BotShortDescription* on success.

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

**language\_code:** str | None

A two-letter ISO 639-1 language code or an empty string

## Usage

### As bot method

```
result: BotShortDescription = await bot.get_my_short_description(...)
```

### Method as object

Imports:

- `from aiogram.methods.get_my_short_description import GetMyShortDescription`
- `alias: from aiogram.methods import GetMyShortDescription`

### With specific bot

```
result: BotShortDescription = await bot(GetMyShortDescription(...))
```

## getUserProfilePhotos

Returns: `UserProfilePhotos`

```
class aiogram.methods.get_user_profile_photos.GetUserProfilePhotos(*user_id: int, offset: int |  
                                                                    None = None, limit: int |  
                                                                    None = None, **extra_data:  
                                                                    Any)
```

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

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

**user\_id: int**

Unique identifier of the target user

**offset: int | None**

Sequential number of the first photo to be returned. By default, all photos are returned.

**limit: int | None**

Limits the number of photos to be retrieved. Values between 1-100 are accepted. Defaults to 100.

## Usage

### As bot method

```
result: UserProfilePhotos = await bot.get_user_profile_photos(...)
```

## Method as object

Imports:

- `from aiogram.methods.get_user_profile_photos import GetUserProfilePhotos`
- `alias: from aiogram.methods import GetUserProfilePhotos`

## With specific bot

```
result: UserProfilePhotos = await bot(GetUserProfilePhotos(...))
```

## As shortcut from received object

- `aiogram.types.user.User.get_profile_photos()`

## hideGeneralForumTopic

Returns: bool

```
class aiogram.methods.hide_general_forum_topic.HideGeneralForumTopic(*, chat_id: int | str,  
                                                                    **extra_data: Any)
```

Use this method to hide the ‘General’ topic in a forum supergroup chat. The bot must be an administrator in the chat for this to work and must have the *can\_manage\_topics* administrator rights. The topic will be automatically closed if it was open. Returns True on success.

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

**chat\_id: int | str**

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

## Usage

### As bot method

```
result: bool = await bot.hide_general_forum_topic(...)
```

## Method as object

Imports:

- `from aiogram.methods.hide_general_forum_topic import HideGeneralForumTopic`
- `alias: from aiogram.methods import HideGeneralForumTopic`

### With specific bot

```
result: bool = await bot(HideGeneralForumTopic(...))
```

### As reply into Webhook in handler

```
return HideGeneralForumTopic(...)
```

## leaveChat

Returns: bool

**class** aiogram.methods.leave\_chat.**LeaveChat**(\*, chat\_id: int | str, \*\*extra\_data: Any)

Use this method for your bot to leave a group, supergroup or channel. Returns True on success.

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

**chat\_id: int | str**

Unique identifier for the target chat or username of the target supergroup or channel (in the format @channelusername)

## Usage

### As bot method

```
result: bool = await bot.leave_chat(...)
```

### Method as object

Imports:

- from aiogram.methods.leave\_chat import LeaveChat
- alias: from aiogram.methods import LeaveChat

### With specific bot

```
result: bool = await bot(LeaveChat(...))
```

### As reply into Webhook in handler

```
return LeaveChat(...)
```

### As shortcut from received object

- `aiogram.types.chat.Chat.leave()`

## logOut

Returns: bool

**class** aiogram.methods.log\_out.LogOut(\*\*extra\_data: Any)

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 can immediately log in on a local server, but will not be able to log in back to the cloud Bot API server for 10 minutes. Returns True on success. Requires no parameters.

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

## Usage

### As bot method

```
result: bool = await bot.log_out(...)
```

### Method as object

Imports:

- `from aiogram.methods.log_out import LogOut`
- `alias: from aiogram.methods import LogOut`

### With specific bot

```
result: bool = await bot(LogOut(...))
```

### As reply into Webhook in handler

```
return Logout(...)
```

### pinChatMessage

Returns: bool

```
class aiogram.methods.pin_chat_message.PinChatMessage(*, chat_id: int | str, message_id: int,
                                                         disable_notification: bool | None = None,
                                                         **extra_data: Any)
```

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’ administrator right in a supergroup or ‘can\_edit\_messages’ administrator right in a channel. Returns True on success.

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

**chat\_id: int | str**

Unique identifier for the target chat or username of the target channel (in the format @channelusername)

**message\_id: int**

Identifier of a message to pin

**disable\_notification: bool | None**

Pass True if it is not necessary to send a notification to all chat members about the new pinned message. Notifications are always disabled in channels and private chats.

### Usage

#### As bot method

```
result: bool = await bot.pin_chat_message(...)
```

#### Method as object

Imports:

- `from aiogram.methods.pin_chat_message import PinChatMessage`
- `alias: from aiogram.methods import PinChatMessage`

#### With specific bot

```
result: bool = await bot(PinChatMessage(...))
```

### As reply into Webhook in handler

```
return PinChatMessage(...)
```

### As shortcut from received object

- `aiogram.types.message.Message.pin()`
- `aiogram.types.chat.Chat.pin_message()`

### promoteChatMember

Returns: bool

```
class aiogram.methods.promote_chat_member.PromoteChatMember(*, chat_id: int | str, user_id: int,
                                                             is_anonymous: bool | None = None,
                                                             can_manage_chat: bool | None =
                                                             None, can_post_messages: bool |
                                                             None = None, can_edit_messages:
                                                             bool | None = None,
                                                             can_delete_messages: bool | None =
                                                             None, can_manage_video_chats: bool
                                                             | None = None,
                                                             can_restrict_members: bool | None =
                                                             None, can_promote_members: bool |
                                                             None = None, can_change_info: bool
                                                             | None = None, can_invite_users:
                                                             bool | None = None,
                                                             can_pin_messages: bool | None =
                                                             None, can_manage_topics: bool |
                                                             None = None, **extra_data: Any)
```

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 administrator rights. Pass `False` for all boolean parameters to demote a user. Returns `True` on success.

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

**chat\_id:** int | str

Unique identifier for the target chat or username of the target channel (in the format @channelusername)

**user\_id:** int

Unique identifier of the target user

**is\_anonymous:** bool | None

Pass `True` if the administrator's presence in the chat is hidden

**can\_manage\_chat:** bool | None

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\_post\_messages:** bool | None

Pass `True` if the administrator can create channel posts, channels only



**can\_edit\_messages:** bool | None

Pass True if the administrator can edit messages of other users and can pin messages, channels only

**can\_delete\_messages:** bool | None

Pass True if the administrator can delete messages of other users

**can\_manage\_video\_chats:** bool | None

Pass True if the administrator can manage video chats

**can\_restrict\_members:** bool | None

Pass True if the administrator can restrict, ban or unban chat members

**can\_promote\_members:** bool | None

Pass True if the administrator can add new administrators with a subset of their own privileges or demote administrators that they have promoted, directly or indirectly (promoted by administrators that were appointed by him)

**can\_change\_info:** bool | None

Pass True if the administrator can change chat title, photo and other settings

**can\_invite\_users:** bool | None

Pass True if the administrator can invite new users to the chat

**can\_pin\_messages:** bool | None

Pass True if the administrator can pin messages, supergroups only

**can\_manage\_topics:** bool | None

Pass True if the user is allowed to create, rename, close, and reopen forum topics, supergroups only

## Usage

### As bot method

```
result: bool = await bot.promote_chat_member(...)
```

### Method as object

Imports:

- from aiogram.methods.promote\_chat\_member import PromoteChatMember
- alias: from aiogram.methods import PromoteChatMember

### With specific bot

```
result: bool = await bot(PromoteChatMember(...))
```

### As reply into Webhook in handler

```
return PromoteChatMember(...)
```

### As shortcut from received object

- `aiogram.types.chat.Chat.promote()`

### reopenForumTopic

Returns: bool

```
class aiogram.methods.reopen_forum_topic.ReopenForumTopic(*, chat_id: int | str, message_thread_id:
                                                         int, **extra_data: Any)
```

Use this method to reopen a closed topic in a forum supergroup chat. The bot must be an administrator in the chat for this to work and must have the *can\_manage\_topics* administrator rights, unless it is the creator of the topic. Returns True on success.

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

**chat\_id: int | str**

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

**message\_thread\_id: int**

Unique identifier for the target message thread of the forum topic

### Usage

#### As bot method

```
result: bool = await bot.reopen_forum_topic(...)
```

### Method as object

Imports:

- `from aiogram.methods.reopen_forum_topic import ReopenForumTopic`
- `alias: from aiogram.methods import ReopenForumTopic`

### With specific bot

```
result: bool = await bot(ReopenForumTopic(...))
```

### As reply into Webhook in handler

```
return ReopenForumTopic(...)
```

## reopenGeneralForumTopic

Returns: bool

```
class aiogram.methods.reopen_general_forum_topic.ReopenGeneralForumTopic(*, chat_id: int | str,
                                                                            **extra_data: Any)
```

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

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

**chat\_id: int | str**

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

### Usage

#### As bot method

```
result: bool = await bot.reopen_general_forum_topic(...)
```

### Method as object

Imports:

- `from aiogram.methods.reopen_general_forum_topic import ReopenGeneralForumTopic`
- `alias: from aiogram.methods import ReopenGeneralForumTopic`

### With specific bot

```
result: bool = await bot(ReopenGeneralForumTopic(...))
```

### As reply into Webhook in handler

```
return ReopenGeneralForumTopic(...)
```

### restrictChatMember

Returns: bool

```
class aiogram.methods.restrict_chat_member.RestrictChatMember(*, chat_id: int | str, user_id: int,
                                                                permissions: ChatPermissions,
                                                                use_independent_chat_permissions:
                                                                bool | None = None, until_date:
                                                                datetime | timedelta | int | None =
                                                                None, **extra_data: Any)
```

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 administrator rights. Pass `True` for all permissions to lift restrictions from a user. Returns `True` on success.

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

**chat\_id: int | str**

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

**user\_id: int**

Unique identifier of the target user

**permissions: ChatPermissions**

A JSON-serialized object for new user permissions

**use\_independent\_chat\_permissions: bool | None**

Pass `True` if chat permissions are set independently. Otherwise, the `can_send_other_messages` and `can_add_web_page_previews` permissions will imply the `can_send_messages`, `can_send_audios`, `can_send_documents`, `can_send_photos`, `can_send_videos`, `can_send_video_notes`, and `can_send_voice_notes` permissions; the `can_send_polls` permission will imply the `can_send_messages` permission.

**until\_date: datetime.datetime | datetime.timedelta | int | None**

Date when restrictions will be lifted for the user, unix time. If user is restricted for more than 366 days or less than 30 seconds from the current time, they are considered to be restricted forever

### Usage

#### As bot method

```
result: bool = await bot.restrict_chat_member(...)
```

## Method as object

Imports:

- `from aiogram.methods.restrict_chat_member import RestrictChatMember`
- `alias: from aiogram.methods import RestrictChatMember`

## With specific bot

```
result: bool = await bot(RestrictChatMember(...))
```

## As reply into Webhook in handler

```
return RestrictChatMember(...)
```

## As shortcut from received object

- `aiogram.types.chat.Chat.restrict()`

## revokeChatInviteLink

Returns: `ChatInviteLink`

```
class aiogram.methods.revoke_chat_invite_link.RevokeChatInviteLink(*, chat_id: int | str,
                                                                    invite_link: str,
                                                                    **extra_data: Any)
```

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 administrator rights. Returns the revoked invite link as `aiogram.types.chat_invite_link.ChatInviteLink` object.

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

**chat\_id: int | str**

Unique identifier of the target chat or username of the target channel (in the format @channelusername)

**invite\_link: str**

The invite link to revoke

## Usage

### As bot method

```
result: ChatInviteLink = await bot.revoke_chat_invite_link(...)
```

## Method as object

Imports:

- `from aiogram.methods.revoke_chat_invite_link import RevokeChatInviteLink`
- `alias: from aiogram.methods import RevokeChatInviteLink`

## With specific bot

```
result: ChatInviteLink = await bot(RevokeChatInviteLink(...))
```

## As reply into Webhook in handler

```
return RevokeChatInviteLink(...)
```

## As shortcut from received object

- `aiogram.types.chat.Chat.revoke_invite_link()`

## sendAnimation

Returns: Message

```
class aiogram.methods.send_animation.SendAnimation(*, chat_id: int | str, animation: InputFile | str,
    message_thread_id: int | None = None, duration:
    int | None = None, width: int | None = None,
    height: int | None = None, thumbnail: InputFile |
    str | None = None, caption: str | None = None,
    parse_mode: str | None =
    sentinel.UNSET_PARSE_MODE,
    caption_entities: List[MessageEntity] | None =
    None, has_spoiler: bool | None = None,
    disable_notification: bool | None = None,
    protect_content: bool | None =
    sentinel.UNSET_PROTECT_CONTENT,
    reply_to_message_id: int | None = None,
    allow_sending_without_reply: bool | None =
    None, reply_markup: InlineKeyboardMarkup |
    ReplyKeyboardMarkup | ReplyKeyboardRemove |
    ForceReply | None = None, **extra_data: Any)
```

Use this method to send animation files (GIF or H.264/MPEG-4 AVC video without sound). On success, the sent `aiogram.types.message.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>

**chat\_id: int | str**

Unique identifier for the target chat or username of the target channel (in the format @channelusername)

**animation:** [InputFile](#) | str

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. [More information on Sending Files](#) »

**message\_thread\_id:** int | None

Unique identifier for the target message thread (topic) of the forum; for forum supergroups only

**duration:** int | None

Duration of sent animation in seconds

**width:** int | None

Animation width

**height:** int | None

Animation height

**thumbnail:** [InputFile](#) | str | None

Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can't be reused and can be only uploaded as a new file, so you can pass 'attach://<file\_attach\_name>' if the thumbnail was uploaded using multipart/form-data under <file\_attach\_name>. [More information on Sending Files](#) »

**caption:** str | None

Animation caption (may also be used when resending animation by *file\_id*), 0-1024 characters after entities parsing

**parse\_mode:** str | None

Mode for parsing entities in the animation caption. See [formatting options](#) for more details.

**caption\_entities:** List[[MessageEntity](#)] | None

A JSON-serialized list of special entities that appear in the caption, which can be specified instead of *parse\_mode*

**has\_spoiler:** bool | None

Pass True if the animation needs to be covered with a spoiler animation

**disable\_notification:** bool | None

Sends the message [silently](#). Users will receive a notification with no sound.

**protect\_content:** bool | None

Protects the contents of the sent message from forwarding and saving

**reply\_to\_message\_id:** int | None

If the message is a reply, ID of the original message

**allow\_sending\_without\_reply:** bool | None

Pass True if the message should be sent even if the specified replied-to message is not found

**reply\_markup:** [InlineKeyboardMarkup](#) | [ReplyKeyboardMarkup](#) | [ReplyKeyboardRemove](#) | [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.

## Usage

### As bot method

```
result: Message = await bot.send_animation(...)
```

### Method as object

Imports:

- `from aiogram.methods.send_animation import SendAnimation`
- `alias: from aiogram.methods import SendAnimation`

### With specific bot

```
result: Message = await bot(SendAnimation(...))
```

### As reply into Webhook in handler

```
return SendAnimation(...)
```

### As shortcut from received object

- `aiogram.types.message.Message.answer_animation()`
- `aiogram.types.message.Message.reply_animation()`
- `aiogram.types.chat_member_updated.ChatMemberUpdated.answer_animation()`
- `aiogram.types.chat_join_request.ChatJoinRequest.answer_animation()`
- `aiogram.types.chat_join_request.ChatJoinRequest.answer_animation_pm()`

## sendAudio

Returns: Message



```
class aiogram.methods.send_audio.SendAudio(*, chat_id: int | str, audio: InputFile | str,
                                           message_thread_id: int | None = None, caption: str | None =
                                           None, parse_mode: str | None =
                                           sentinel.UNSET_PARSE_MODE, caption_entities:
                                           List[MessageEntity] | None = None, duration: int | None =
                                           None, performer: str | None = None, title: str | None =
                                           None, thumbnail: InputFile | str | None = None,
                                           disable_notification: bool | None = None, protect_content:
                                           bool | None = sentinel.UNSET_PROTECT_CONTENT,
                                           reply_to_message_id: int | None = None,
                                           allow_sending_without_reply: bool | None = None,
                                           reply_markup: InlineKeyboardMarkup |
                                           ReplyKeyboardMarkup | ReplyKeyboardRemove |
                                           ForceReply | None = None, **extra_data: Any)
```

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 or .M4A format. On success, the sent `aiogram.types.message.Message` is returned. Bots can currently send audio files of up to 50 MB in size, this limit may be changed in the future. For sending voice messages, use the `aiogram.methods.send_voice.SendVoice` method instead.

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

**chat\_id:** `int | str`

Unique identifier for the target chat or username of the target channel (in the format @channelusername)

**audio:** `InputFile | str`

Audio file to send. Pass a file\_id as String to send an audio file that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get an audio file from the Internet, or upload a new one using multipart/form-data. *More information on Sending Files »*

**message\_thread\_id:** `int | None`

Unique identifier for the target message thread (topic) of the forum; for forum supergroups only

**caption:** `str | None`

Audio caption, 0-1024 characters after entities parsing

**parse\_mode:** `str | None`

Mode for parsing entities in the audio caption. See [formatting options](#) for more details.

**caption\_entities:** `List[MessageEntity] | None`

A JSON-serialized list of special entities that appear in the caption, which can be specified instead of `parse_mode`

**duration:** `int | None`

Duration of the audio in seconds

**performer:** `str | None`

Performer

**title:** `str | None`

Track name

**thumbnail:** `InputFile | str | None`

Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can't be reused and can be only uploaded as a new file, so you can pass 'attach://<file\_attach\_name>' if the thumbnail was uploaded using multipart/form-data under <file\_attach\_name>. *More information on Sending Files »*

**disable\_notification:** `bool` | `None`

Sends the message `silently`. Users will receive a notification with no sound.

**protect\_content:** `bool` | `None`

Protects the contents of the sent message from forwarding and saving

**reply\_to\_message\_id:** `int` | `None`

If the message is a reply, ID of the original message

**allow\_sending\_without\_reply:** `bool` | `None`

Pass `True` if the message should be sent even if the specified replied-to message is not found

**reply\_markup:** `InlineKeyboardMarkup` | `ReplyKeyboardMarkup` | `ReplyKeyboardRemove` | `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.

## Usage

### As bot method

```
result: Message = await bot.send_audio(...)
```

### Method as object

Imports:

- `from aiogram.methods.send_audio import SendAudio`
- `alias: from aiogram.methods import SendAudio`

### With specific bot

```
result: Message = await bot(SendAudio(...))
```

### As reply into Webhook in handler

```
return SendAudio(...)
```

### As shortcut from received object

- `aiogram.types.message.Message.answer_audio()`
- `aiogram.types.message.Message.reply_audio()`
- `aiogram.types.chat_member_updated.ChatMemberUpdated.answer_audio()`
- `aiogram.types.chat_join_request.ChatJoinRequest.answer_audio()`
- `aiogram.types.chat_join_request.ChatJoinRequest.answer_audio_pm()`

## sendChatAction

Returns: bool

```
class aiogram.methods.send_chat_action.SendChatAction(*, chat_id: int | str, action: str,
                                                       message_thread_id: int | None = None,
                                                       **extra_data: Any)
```

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 `aiogram.methods.send_chat_action.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>

**chat\_id:** int | str

Unique identifier for the target chat or username of the target channel (in the format @channelusername)

**action:** str

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, *choose\_sticker* for stickers, *find\_location* for location data, *record\_video\_note* or *upload\_video\_note* for video notes.

**message\_thread\_id:** int | None

Unique identifier for the target message thread; supergroups only

## Usage

### As bot method

```
result: bool = await bot.send_chat_action(...)
```

### Method as object

Imports:

- `from aiogram.methods.send_chat_action import SendChatAction`
- `alias: from aiogram.methods import SendChatAction`

### With specific bot

```
result: bool = await bot(SendChatAction(...))
```

### As reply into Webhook in handler

```
return SendChatAction(...)
```

### As shortcut from received object

- `aiogram.types.chat.Chat.do()`

### sendContact

Returns: Message

```
class aiogram.methods.send_contact.SendContact(*, chat_id: int | str, phone_number: str, first_name: str, message_thread_id: int | None = None, last_name: str | None = None, vcard: str | None = None, disable_notification: bool | None = None, protect_content: bool | None = sentinel.UNSET_PROTECT_CONTENT, reply_to_message_id: int | None = None, allow_sending_without_reply: bool | None = None, reply_markup: InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply | None = None, **extra_data: Any)
```

Use this method to send phone contacts. On success, the sent `aiogram.types.message.Message` is returned.

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

**chat\_id: int | str**

Unique identifier for the target chat or username of the target channel (in the format @channelusername)

**phone\_number: str**

Contact's phone number

**first\_name: str**

Contact's first name

**message\_thread\_id: int | None**

Unique identifier for the target message thread (topic) of the forum; for forum supergroups only

**last\_name: str | None**

Contact's last name

**vcard: str | None**

Additional data about the contact in the form of a vCard, 0-2048 bytes

**disable\_notification: bool | None**

Sends the message `silently`. Users will receive a notification with no sound.

**protect\_content:** `bool` | `None`

Protects the contents of the sent message from forwarding and saving

**reply\_to\_message\_id:** `int` | `None`

If the message is a reply, ID of the original message

**allow\_sending\_without\_reply:** `bool` | `None`

Pass True if the message should be sent even if the specified replied-to message is not found

**reply\_markup:** `InlineKeyboardMarkup` | `ReplyKeyboardMarkup` | `ReplyKeyboardRemove` | `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.

## Usage

### As bot method

```
result: Message = await bot.send_contact(...)
```

### Method as object

Imports:

- `from aiogram.methods.send_contact import SendContact`
- `alias: from aiogram.methods import SendContact`

### With specific bot

```
result: Message = await bot(SendContact(...))
```

### As reply into Webhook in handler

```
return SendContact(...)
```

### As shortcut from received object

- `aiogram.types.message.Message.answer_contact()`
- `aiogram.types.message.Message.reply_contact()`
- `aiogram.types.chat_member_updated.ChatMemberUpdated.answer_contact()`
- `aiogram.types.chat_join_request.ChatJoinRequest.answer_contact()`
- `aiogram.types.chat_join_request.ChatJoinRequest.answer_contact_pm()`

## sendDice

Returns: Message

```
class aiogram.methods.send_dice.SendDice(*, chat_id: int | str, message_thread_id: int | None = None,
                                          emoji: str | None = None, disable_notification: bool | None =
                                          None, protect_content: bool | None =
                                          sentinel.UNSET_PROTECT_CONTENT,
                                          reply_to_message_id: int | None = None,
                                          allow_sending_without_reply: bool | None = None,
                                          reply_markup: InlineKeyboardMarkup |
                                          ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply
                                          | None = None, **extra_data: Any)
```

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

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

**chat\_id: int | str**

Unique identifier for the target chat or username of the target channel (in the format @channelusername)

**message\_thread\_id: int | None**

Unique identifier for the target message thread (topic) of the forum; for forum supergroups only

**emoji: str | None**

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: bool | None**

Sends the message *silently*. Users will receive a notification with no sound.

**protect\_content: bool | None**

Protects the contents of the sent message from forwarding

**reply\_to\_message\_id: int | None**

If the message is a reply, ID of the original message

**allow\_sending\_without\_reply: bool | None**

Pass True if the message should be sent even if the specified replied-to message is not found

**reply\_markup: InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | 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.

## Usage

### As bot method

```
result: Message = await bot.send_dice(...)
```

## Method as object

Imports:

- `from aiogram.methods.send_dice import SendDice`
- alias: `from aiogram.methods import SendDice`

## With specific bot

```
result: Message = await bot(SendDice(...))
```

## As reply into Webhook in handler

```
return SendDice(...)
```

## As shortcut from received object

- `aiogram.types.message.Message.answer_dice()`
- `aiogram.types.message.Message.reply_dice()`
- `aiogram.types.chat_member_updated.ChatMemberUpdated.answer_dice()`
- `aiogram.types.chat_join_request.ChatJoinRequest.answer_dice()`
- `aiogram.types.chat_join_request.ChatJoinRequest.answer_dice_pm()`

## sendDocument

Returns: Message

```
class aiogram.methods.send_document.SendDocument(*, chat_id: int | str, document: InputFile | str,
    message_thread_id: int | None = None, thumbnail:
    InputFile | str | None = None, caption: str | None =
    None, parse_mode: str | None =
    sentinel.UNSET_PARSE_MODE, caption_entities:
    List[MessageEntity] | None = None,
    disable_content_type_detection: bool | None =
    None, disable_notification: bool | None = None,
    protect_content: bool | None =
    sentinel.UNSET_PROTECT_CONTENT,
    reply_to_message_id: int | None = None,
    allow_sending_without_reply: bool | None = None,
    reply_markup: InlineKeyboardMarkup |
    ReplyKeyboardMarkup | ReplyKeyboardRemove |
    ForceReply | None = None, **extra_data: Any)
```

Use this method to send general files. On success, the sent `aiogram.types.message.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>

**chat\_id:** `int` | `str`

Unique identifier for the target chat or username of the target channel (in the format @channelusername)

**document:** `InputFile` | `str`

File to send. Pass a `file_id` as `String` to send a file that exists on the Telegram servers (recommended), 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 information on Sending Files »](#)

**message\_thread\_id:** `int` | `None`

Unique identifier for the target message thread (topic) of the forum; for forum supergroups only

**thumbnail:** `InputFile` | `str` | `None`

Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can't be reused and can be only uploaded as a new file, so you can pass 'attach://<file\_attach\_name>' if the thumbnail was uploaded using multipart/form-data under <file\_attach\_name>. [More information on Sending Files »](#)

**caption:** `str` | `None`

Document caption (may also be used when resending documents by `file_id`), 0-1024 characters after entities parsing

**parse\_mode:** `str` | `None`

Mode for parsing entities in the document caption. See [formatting options](#) for more details.

**caption\_entities:** `List[MessageEntity]` | `None`

A JSON-serialized list of special entities that appear in the caption, which can be specified instead of `parse_mode`

**disable\_content\_type\_detection:** `bool` | `None`

Disables automatic server-side content type detection for files uploaded using multipart/form-data

**disable\_notification:** `bool` | `None`

Sends the message [silently](#). Users will receive a notification with no sound.

**protect\_content:** `bool` | `None`

Protects the contents of the sent message from forwarding and saving

**reply\_to\_message\_id:** `int` | `None`

If the message is a reply, ID of the original message

**allow\_sending\_without\_reply:** `bool` | `None`

Pass `True` if the message should be sent even if the specified replied-to message is not found

**reply\_markup:** `InlineKeyboardMarkup` | `ReplyKeyboardMarkup` | `ReplyKeyboardRemove` | `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.



## Usage

### As bot method

```
result: Message = await bot.send_document(...)
```

### Method as object

Imports:

- `from aiogram.methods.send_document import SendDocument`
- `alias: from aiogram.methods import SendDocument`

### With specific bot

```
result: Message = await bot(SendDocument(...))
```

### As reply into Webhook in handler

```
return SendDocument(...)
```

### As shortcut from received object

- `aiogram.types.message.Message.answer_document()`
- `aiogram.types.message.Message.reply_document()`
- `aiogram.types.chat_member_updated.ChatMemberUpdated.answer_document()`
- `aiogram.types.chat_join_request.ChatJoinRequest.answer_document()`
- `aiogram.types.chat_join_request.ChatJoinRequest.answer_document_pm()`

## sendLocation

Returns: Message

```
class aiogram.methods.send_location.SendLocation(*, chat_id: int | str, latitude: float, longitude: float,
message_thread_id: int | None = None,
horizontal_accuracy: float | None = None,
live_period: int | None = None, heading: int | None = None, proximity_alert_radius: int | None = None,
disable_notification: bool | None = None,
protect_content: bool | None =
sentinel.UNSET_PROTECT_CONTENT,
reply_to_message_id: int | None = None,
allow_sending_without_reply: bool | None = None,
reply_markup: InlineKeyboardMarkup |
ReplyKeyboardMarkup | ReplyKeyboardRemove |
ForceReply | None = None, **extra_data: Any)
```

Use this method to send point on the map. On success, the sent `aiogram.types.message.Message` is returned.

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

**chat\_id:** `int` | `str`

Unique identifier for the target chat or username of the target channel (in the format @channelusername)

**latitude:** `float`

Latitude of the location

**longitude:** `float`

Longitude of the location

**message\_thread\_id:** `int` | `None`

Unique identifier for the target message thread (topic) of the forum; for forum supergroups only

**horizontal\_accuracy:** `float` | `None`

The radius of uncertainty for the location, measured in meters; 0-1500

**live\_period:** `int` | `None`

Period in seconds for which the location will be updated (see [Live Locations](#), should be between 60 and 86400).

**heading:** `int` | `None`

For live locations, a direction in which the user is moving, in degrees. Must be between 1 and 360 if specified.

**proximity\_alert\_radius:** `int` | `None`

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:** `bool` | `None`

Sends the message [silently](#). Users will receive a notification with no sound.

**protect\_content:** `bool` | `None`

Protects the contents of the sent message from forwarding and saving

**reply\_to\_message\_id:** `int` | `None`

If the message is a reply, ID of the original message

**allow\_sending\_without\_reply:** `bool` | `None`

Pass True if the message should be sent even if the specified replied-to message is not found

**reply\_markup:** [InlineKeyboardMarkup](#) | [ReplyKeyboardMarkup](#) | [ReplyKeyboardRemove](#) | [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.

## Usage

### As bot method

```
result: Message = await bot.send_location(...)
```

### Method as object

Imports:

- `from aiogram.methods.send_location import SendLocation`
- `alias: from aiogram.methods import SendLocation`

### With specific bot

```
result: Message = await bot(SendLocation(...))
```

### As reply into Webhook in handler

```
return SendLocation(...)
```

### As shortcut from received object

- `aiogram.types.message.Message.answer_location()`
- `aiogram.types.message.Message.reply_location()`
- `aiogram.types.chat_member_updated.ChatMemberUpdated.answer_location()`
- `aiogram.types.chat_join_request.ChatJoinRequest.answer_location()`
- `aiogram.types.chat_join_request.ChatJoinRequest.answer_location_pm()`

## sendMediaGroup

Returns: `List[Message]`

```
class aiogram.methods.send_media_group.SendMediaGroup(*, chat_id: int | str, media:
    List[InputMediaAudio | InputMediaDocument
    | InputMediaPhoto | InputMediaVideo],
    message_thread_id: int | None = None,
    disable_notification: bool | None = None,
    protect_content: bool | None =
    sentinel.UNSET_PROTECT_CONTENT,
    reply_to_message_id: int | None = None,
    allow_sending_without_reply: bool | None =
    None, **extra_data: Any)
```

Use this method to send a group of photos, videos, documents or audios as an album. Documents and audio files can be only grouped 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>

**chat\_id:** `int | str`

Unique identifier for the target chat or username of the target channel (in the format @channelusername)

**media:** `List[InputMediaAudio | InputMediaDocument | InputMediaPhoto | InputMediaVideo]`

A JSON-serialized array describing messages to be sent, must include 2-10 items

**message\_thread\_id:** `int | None`

Unique identifier for the target message thread (topic) of the forum; for forum supergroups only

**disable\_notification:** `bool | None`

Sends messages `silently`. Users will receive a notification with no sound.

**protect\_content:** `bool | None`

Protects the contents of the sent messages from forwarding and saving

**reply\_to\_message\_id:** `int | None`

If the messages are a reply, ID of the original message

**allow\_sending\_without\_reply:** `bool | None`

Pass True if the message should be sent even if the specified replied-to message is not found

## Usage

### As bot method

```
result: List[Message] = await bot.send_media_group(...)
```

### Method as object

Imports:

- `from aiogram.methods.send_media_group import SendMediaGroup`
- `alias: from aiogram.methods import SendMediaGroup`

### With specific bot

```
result: List[Message] = await bot(SendMediaGroup(...))
```

### As reply into Webhook in handler

```
return SendMediaGroup(...)
```

### As shortcut from received object

- `aiogram.types.message.Message.answer_media_group()`
- `aiogram.types.message.Message.reply_media_group()`
- `aiogram.types.chat_member_updated.ChatMemberUpdated.answer_media_group()`
- `aiogram.types.chat_join_request.ChatJoinRequest.answer_media_group()`
- `aiogram.types.chat_join_request.ChatJoinRequest.answer_media_group_pm()`

### sendMessage

Returns: Message

```
class aiogram.methods.send_message.SendMessage(*, chat_id: int | str, text: str, message_thread_id: int |
None = None, parse_mode: str | None =
sentinel.UNSET_PARSE_MODE, entities:
List[MessageEntity] | None = None,
disable_web_page_preview: bool | None =
sentinel.UNSET_DISABLE_WEB_PAGE_PREVIEW,
disable_notification: bool | None = None,
protect_content: bool | None =
sentinel.UNSET_PROTECT_CONTENT,
reply_to_message_id: int | None = None,
allow_sending_without_reply: bool | None = None,
reply_markup: InlineKeyboardMarkup |
ReplyKeyboardMarkup | ReplyKeyboardRemove |
ForceReply | None = None, **extra_data: Any)
```

Use this method to send text messages. On success, the sent `aiogram.types.message.Message` is returned.

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

**chat\_id: int | str**

Unique identifier for the target chat or username of the target channel (in the format @channelusername)

**text: str**

Text of the message to be sent, 1-4096 characters after entities parsing

**message\_thread\_id: int | None**

Unique identifier for the target message thread (topic) of the forum; for forum supergroups only

**parse\_mode: str | None**

Mode for parsing entities in the message text. See [formatting options](#) for more details.

**entities: List[MessageEntity] | None**

A JSON-serialized list of special entities that appear in message text, which can be specified instead of `parse_mode`

**disable\_web\_page\_preview:** `bool` | `None`

Disables link previews for links in this message

**disable\_notification:** `bool` | `None`

Sends the message `silently`. Users will receive a notification with no sound.

**protect\_content:** `bool` | `None`

Protects the contents of the sent message from forwarding and saving

**reply\_to\_message\_id:** `int` | `None`

If the message is a reply, ID of the original message

**allow\_sending\_without\_reply:** `bool` | `None`

Pass True if the message should be sent even if the specified replied-to message is not found

**reply\_markup:** `InlineKeyboardMarkup` | `ReplyKeyboardMarkup` | `ReplyKeyboardRemove` | `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.

## Usage

### As bot method

```
result: Message = await bot.send_message(...)
```

### Method as object

Imports:

- `from aiogram.methods.send_message import SendMessage`
- `alias: from aiogram.methods import SendMessage`

### With specific bot

```
result: Message = await bot(SendMessage(...))
```

### As reply into Webhook in handler

```
return SendMessage(...)
```

### As shortcut from received object

- `aiogram.types.message.Message.answer()`
- `aiogram.types.message.Message.reply()`
- `aiogram.types.chat_member_updated.ChatMemberUpdated.answer()`
- `aiogram.types.chat_join_request.ChatJoinRequest.answer()`
- `aiogram.types.chat_join_request.ChatJoinRequest.answer_pm()`

### sendPhoto

Returns: `Message`

```
class aiogram.methods.send_photo.SendPhoto(*, chat_id: int | str, photo: InputFile | str,
                                           message_thread_id: int | None = None, caption: str | None =
                                           None, parse_mode: str | None =
                                           sentinel.UNSET_PARSE_MODE, caption_entities:
                                           List[MessageEntity] | None = None, has_spoiler: bool |
                                           None = None, disable_notification: bool | None = None,
                                           protect_content: bool | None =
                                           sentinel.UNSET_PROTECT_CONTENT,
                                           reply_to_message_id: int | None = None,
                                           allow_sending_without_reply: bool | None = None,
                                           reply_markup: InlineKeyboardMarkup |
                                           ReplyKeyboardMarkup | ReplyKeyboardRemove |
                                           ForceReply | None = None, **extra_data: Any)
```

Use this method to send photos. On success, the sent `aiogram.types.message.Message` is returned.

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

**chat\_id:** `int | str`

Unique identifier for the target chat or username of the target channel (in the format @channelusername)

**photo:** `InputFile | str`

Photo to send. Pass a `file_id` as `String` to send a photo that exists on the Telegram servers (recommended), pass an HTTP URL as a `String` for Telegram to get a photo from the Internet, or upload a new photo using multipart/form-data. The photo must be at most 10 MB in size. The photo's width and height must not exceed 10000 in total. Width and height ratio must be at most 20. [More information on Sending Files »](#)

**message\_thread\_id:** `int | None`

Unique identifier for the target message thread (topic) of the forum; for forum supergroups only

**caption:** `str | None`

Photo caption (may also be used when resending photos by `file_id`), 0-1024 characters after entities parsing

**parse\_mode:** `str | None`

Mode for parsing entities in the photo caption. See [formatting options](#) for more details.

**caption\_entities:** `List[MessageEntity] | None`

A JSON-serialized list of special entities that appear in the caption, which can be specified instead of `parse_mode`

**has\_spoiler:** `bool | None`

Pass `True` if the photo needs to be covered with a spoiler animation

**disable\_notification:** `bool` | `None`

Sends the message `silently`. Users will receive a notification with no sound.

**protect\_content:** `bool` | `None`

Protects the contents of the sent message from forwarding and saving

**reply\_to\_message\_id:** `int` | `None`

If the message is a reply, ID of the original message

**allow\_sending\_without\_reply:** `bool` | `None`

Pass True if the message should be sent even if the specified replied-to message is not found

**reply\_markup:** `InlineKeyboardMarkup` | `ReplyKeyboardMarkup` | `ReplyKeyboardRemove` | `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.

## Usage

### As bot method

```
result: Message = await bot.send_photo(...)
```

### Method as object

Imports:

- `from aiogram.methods.send_photo import SendPhoto`
- `alias: from aiogram.methods import SendPhoto`

### With specific bot

```
result: Message = await bot(SendPhoto(...))
```

### As reply into Webhook in handler

```
return SendPhoto(...)
```

### As shortcut from received object

- `aiogram.types.message.Message.answer_photo()`
- `aiogram.types.message.Message.reply_photo()`
- `aiogram.types.chat_member_updated.ChatMemberUpdated.answer_photo()`
- `aiogram.types.chat_join_request.ChatJoinRequest.answer_photo()`
- `aiogram.types.chat_join_request.ChatJoinRequest.answer_photo_pm()`



## sendPoll

Returns: Message

```
class aiogram.methods.send_poll.SendPoll(*, chat_id: int | str, question: str, options: List[str],
                                         message_thread_id: int | None = None, is_anonymous: bool |
                                         None = None, type: str | None = None,
                                         allows_multiple_answers: bool | None = None,
                                         correct_option_id: int | None = None, explanation: str | None
                                         = None, explanation_parse_mode: str | None =
                                         sentinel.UNSET_PARSE_MODE, explanation_entities:
                                         List[MessageEntity] | None = None, open_period: int | None =
                                         None, close_date: datetime | timedelta | int | None = None,
                                         is_closed: bool | None = None, disable_notification: bool |
                                         None = None, protect_content: bool | None =
                                         sentinel.UNSET_PROTECT_CONTENT,
                                         reply_to_message_id: int | None = None,
                                         allow_sending_without_reply: bool | None = None,
                                         reply_markup: InlineKeyboardMarkup |
                                         ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply
                                         | None = None, **extra_data: Any)
```

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

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

**chat\_id:** int | str

Unique identifier for the target chat or username of the target channel (in the format @channelusername)

**question:** str

Poll question, 1-300 characters

**options:** List[str]

A JSON-serialized list of answer options, 2-10 strings 1-100 characters each

**message\_thread\_id:** int | None

Unique identifier for the target message thread (topic) of the forum; for forum supergroups only

**is\_anonymous:** bool | None

True, if the poll needs to be anonymous, defaults to True

**type:** str | None

Poll type, 'quiz' or 'regular', defaults to 'regular'

**allows\_multiple\_answers:** bool | None

True, if the poll allows multiple answers, ignored for polls in quiz mode, defaults to False

**correct\_option\_id:** int | None

0-based identifier of the correct answer option, required for polls in quiz mode

**explanation:** str | None

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:** str | None

Mode for parsing entities in the explanation. See [formatting options](#) for more details.

**explanation\_entities:** `List[MessageEntity] | None`

A JSON-serialized list of special entities that appear in the poll explanation, which can be specified instead of *parse\_mode*

**open\_period:** `int | None`

Amount of time in seconds the poll will be active after creation, 5-600. Can't be used together with *close\_date*.

**close\_date:** `datetime.datetime | datetime.timedelta | int | 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:** `bool | None`

Pass True if the poll needs to be immediately closed. This can be useful for poll preview.

**disable\_notification:** `bool | None`

Sends the message [silently](#). Users will receive a notification with no sound.

**protect\_content:** `bool | None`

Protects the contents of the sent message from forwarding and saving

**reply\_to\_message\_id:** `int | None`

If the message is a reply, ID of the original message

**allow\_sending\_without\_reply:** `bool | None`

Pass True if the message should be sent even if the specified replied-to message is not found

**reply\_markup:** `InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | 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.

## Usage

### As bot method

```
result: Message = await bot.send_poll(...)
```

### Method as object

Imports:

- `from aiogram.methods.send_poll import SendPoll`
- `alias: from aiogram.methods import SendPoll`

### With specific bot

```
result: Message = await bot(SendPoll(...))
```

### As reply into Webhook in handler

```
return SendPoll(...)
```

### As shortcut from received object

- `aiogram.types.message.Message.answer_poll()`
- `aiogram.types.message.Message.reply_poll()`
- `aiogram.types.chat_member_updated.ChatMemberUpdated.answer_poll()`
- `aiogram.types.chat_join_request.ChatJoinRequest.answer_poll()`
- `aiogram.types.chat_join_request.ChatJoinRequest.answer_poll_pm()`

### sendVenue

Returns: Message

```
class aiogram.methods.send_venue.SendVenue(*chat_id: int | str, latitude: float, longitude: float, title: str,
address: str, message_thread_id: int | None = None,
foursquare_id: str | None = None, foursquare_type: str |
None = None, google_place_id: str | None = None,
google_place_type: str | None = None, disable_notification:
bool | None = None, protect_content: bool | None =
sentinel.UNSET_PROTECT_CONTENT,
reply_to_message_id: int | None = None,
allow_sending_without_reply: bool | None = None,
reply_markup: InlineKeyboardMarkup |
ReplyKeyboardMarkup | ReplyKeyboardRemove |
ForceReply | None = None, **extra_data: Any)
```

Use this method to send information about a venue. On success, the sent `aiogram.types.message.Message` is returned.

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

**chat\_id: int | str**

Unique identifier for the target chat or username of the target channel (in the format @channelusername)

**latitude: float**

Latitude of the venue

**longitude: float**

Longitude of the venue

**title: str**

Name of the venue

**address:** `str`

Address of the venue

**message\_thread\_id:** `int | None`

Unique identifier for the target message thread (topic) of the forum; for forum supergroups only

**foursquare\_id:** `str | None`

Foursquare identifier of the venue

**foursquare\_type:** `str | None`

Foursquare type of the venue, if known. (For example, 'arts\_entertainment/default', 'arts\_entertainment/aquarium' or 'food/icecream'.)

**google\_place\_id:** `str | None`

Google Places identifier of the venue

**google\_place\_type:** `str | None`

Google Places type of the venue. (See [supported types](#).)

**disable\_notification:** `bool | None`

Sends the message [silently](#). Users will receive a notification with no sound.

**protect\_content:** `bool | None`

Protects the contents of the sent message from forwarding and saving

**reply\_to\_message\_id:** `int | None`

If the message is a reply, ID of the original message

**allow\_sending\_without\_reply:** `bool | None`

Pass True if the message should be sent even if the specified replied-to message is not found

**reply\_markup:** [InlineKeyboardMarkup](#) | [ReplyKeyboardMarkup](#) | [ReplyKeyboardRemove](#) | [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.

## Usage

### As bot method

```
result: Message = await bot.send_venue(...)
```

### Method as object

Imports:

- `from aiogram.methods.send_venue import SendVenue`
- `alias: from aiogram.methods import SendVenue`

### With specific bot

```
result: Message = await bot(SendVenue(...))
```

### As reply into Webhook in handler

```
return SendVenue(...)
```

### As shortcut from received object

- `aiogram.types.message.Message.answer_venue()`
- `aiogram.types.message.Message.reply_venue()`
- `aiogram.types.chat_member_updated.ChatMemberUpdated.answer_venue()`
- `aiogram.types.chat_join_request.ChatJoinRequest.answer_venue()`
- `aiogram.types.chat_join_request.ChatJoinRequest.answer_venue_pm()`

### sendVideo

Returns: Message

```
class aiogram.methods.send_video.SendVideo(*, chat_id: int | str, video: InputFile | str,
                                           message_thread_id: int | None = None, duration: int | None = None, width: int | None = None, height: int | None = None, thumbnail: InputFile | str | None = None, caption: str | None = None, parse_mode: str | None = sentinel.UNSET_PARSE_MODE, caption_entities: List[MessageEntity] | None = None, has_spoiler: bool | None = None, supports_streaming: bool | None = None, disable_notification: bool | None = None, protect_content: bool | None = sentinel.UNSET_PROTECT_CONTENT, reply_to_message_id: int | None = None, allow_sending_without_reply: bool | None = None, reply_markup: InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply | None = None, **extra_data: Any)
```

Use this method to send video files, Telegram clients support MPEG4 videos (other formats may be sent as `aiogram.types.document.Document`). On success, the sent `aiogram.types.message.Message` is returned. Bots can currently send video files of up to 50 MB in size, this limit may be changed in the future.

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

**chat\_id:** `int | str`

Unique identifier for the target chat or username of the target channel (in the format @channelusername)

**video:** `InputFile | str`

Video to send. Pass a file\_id as String to send a video that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get a video from the Internet, or upload a new video using multipart/form-data. [More information on Sending Files »](#)

**message\_thread\_id:** `int` | `None`

Unique identifier for the target message thread (topic) of the forum; for forum supergroups only

**duration:** `int` | `None`

Duration of sent video in seconds

**width:** `int` | `None`

Video width

**height:** `int` | `None`

Video height

**thumbnail:** [`InputFile`](#) | `str` | `None`

Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can't be reused and can be only uploaded as a new file, so you can pass 'attach://<file\_attach\_name>' if the thumbnail was uploaded using multipart/form-data under <file\_attach\_name>. [More information on Sending Files »](#)

**caption:** `str` | `None`

Video caption (may also be used when resending videos by *file\_id*), 0-1024 characters after entities parsing

**parse\_mode:** `str` | `None`

Mode for parsing entities in the video caption. See [formatting options](#) for more details.

**caption\_entities:** `List`[[`MessageEntity`](#)] | `None`

A JSON-serialized list of special entities that appear in the caption, which can be specified instead of *parse\_mode*

**has\_spoiler:** `bool` | `None`

Pass True if the video needs to be covered with a spoiler animation

**supports\_streaming:** `bool` | `None`

Pass True if the uploaded video is suitable for streaming

**disable\_notification:** `bool` | `None`

Sends the message [silently](#). Users will receive a notification with no sound.

**protect\_content:** `bool` | `None`

Protects the contents of the sent message from forwarding and saving

**reply\_to\_message\_id:** `int` | `None`

If the message is a reply, ID of the original message

**allow\_sending\_without\_reply:** `bool` | `None`

Pass True if the message should be sent even if the specified replied-to message is not found

**reply\_markup:** [`InlineKeyboardMarkup`](#) | [`ReplyKeyboardMarkup`](#) | [`ReplyKeyboardRemove`](#) | [`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.

## Usage

### As bot method

```
result: Message = await bot.send_video(...)
```

### Method as object

Imports:

- `from aiogram.methods.send_video import SendVideo`
- `alias: from aiogram.methods import SendVideo`

### With specific bot

```
result: Message = await bot(SendVideo(...))
```

### As reply into Webhook in handler

```
return SendVideo(...)
```

### As shortcut from received object

- `aiogram.types.message.Message.answer_video()`
- `aiogram.types.message.Message.reply_video()`
- `aiogram.types.chat_member_updated.ChatMemberUpdated.answer_video()`
- `aiogram.types.chat_join_request.ChatJoinRequest.answer_video()`
- `aiogram.types.chat_join_request.ChatJoinRequest.answer_video_pm()`

### sendVideoNote

Returns: Message

```
class aiogram.methods.send_video_note.SendVideoNote(*, chat_id: int | str, video_note: InputFile | str,
    message_thread_id: int | None = None,
    duration: int | None = None, length: int | None =
    None, thumbnail: InputFile | str | None = None,
    disable_notification: bool | None = None,
    protect_content: bool | None =
    sentinel.UNSET_PROTECT_CONTENT,
    reply_to_message_id: int | None = None,
    allow_sending_without_reply: bool | None =
    None, reply_markup: InlineKeyboardMarkup |
ReplyKeyboardMarkup | ReplyKeyboardRemove
    | ForceReply | None = None, **extra_data: Any)
```

As of [v.4.0](#), Telegram clients support rounded square MPEG4 videos of up to 1 minute long. Use this method to send video messages. On success, the sent `aiogram.types.message.Message` is returned.

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

**chat\_id:** `int | str`

Unique identifier for the target chat or username of the target channel (in the format @channelusername)

**video\_note:** `InputFile | str`

Video note to send. Pass a file\_id as String to send a video note that exists on the Telegram servers (recommended) or upload a new video using multipart/form-data. *More information on Sending Files »*. Sending video notes by a URL is currently unsupported

**message\_thread\_id:** `int | None`

Unique identifier for the target message thread (topic) of the forum; for forum supergroups only

**duration:** `int | None`

Duration of sent video in seconds

**length:** `int | None`

Video width and height, i.e. diameter of the video message

**thumbnail:** `InputFile | str | None`

Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can't be reused and can be only uploaded as a new file, so you can pass 'attach://<file\_attach\_name>' if the thumbnail was uploaded using multipart/form-data under <file\_attach\_name>. *More information on Sending Files »*

**disable\_notification:** `bool | None`

Sends the message [silently](#). Users will receive a notification with no sound.

**protect\_content:** `bool | None`

Protects the contents of the sent message from forwarding and saving

**reply\_to\_message\_id:** `int | None`

If the message is a reply, ID of the original message

**allow\_sending\_without\_reply:** `bool | None`

Pass True if the message should be sent even if the specified replied-to message is not found

**reply\_markup:** `InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | 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.

## Usage

### As bot method

```
result: Message = await bot.send_video_note(...)
```



## Method as object

Imports:

- `from aiogram.methods.send_video_note import SendVideoNote`
- `alias: from aiogram.methods import SendVideoNote`

## With specific bot

```
result: Message = await bot(SendVideoNote(...))
```

## As reply into Webhook in handler

```
return SendVideoNote(...)
```

## As shortcut from received object

- `aiogram.types.message.Message.answer_video_note()`
- `aiogram.types.message.Message.reply_video_note()`
- `aiogram.types.chat_member_updated.ChatMemberUpdated.answer_video_note()`
- `aiogram.types.chat_join_request.ChatJoinRequest.answer_video_note()`
- `aiogram.types.chat_join_request.ChatJoinRequest.answer_video_note_pm()`

## sendVoice

Returns: Message

```
class aiogram.methods.send_voice.SendVoice(*chat_id: int | str, voice: InputFile | str,
message_thread_id: int | None = None, caption: str | None =
None, parse_mode: str | None =
sentinel.UNSET_PARSE_MODE, caption_entities:
List[MessageEntity] | None = None, duration: int | None =
None, disable_notification: bool | None = None,
protect_content: bool | None =
sentinel.UNSET_PROTECT_CONTENT,
reply_to_message_id: int | None = None,
allow_sending_without_reply: bool | None = None,
reply_markup: InlineKeyboardMarkup |
ReplyKeyboardMarkup | ReplyKeyboardRemove |
ForceReply | None = None, **extra_data: Any)
```

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 `aiogram.types.audio.Audio` or `aiogram.types.document.Document`). On success, the sent `aiogram.types.message.Message` is returned. Bots can currently send voice messages of up to 50 MB in size, this limit may be changed in the future.

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

**chat\_id:** `int` | `str`

Unique identifier for the target chat or username of the target channel (in the format @channelusername)

**voice:** `InputFile` | `str`

Audio file to send. Pass a `file_id` as `String` to send a file that exists on the Telegram servers (recommended), 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 information on Sending Files »](#)

**message\_thread\_id:** `int` | `None`

Unique identifier for the target message thread (topic) of the forum; for forum supergroups only

**caption:** `str` | `None`

Voice message caption, 0-1024 characters after entities parsing

**parse\_mode:** `str` | `None`

Mode for parsing entities in the voice message caption. See [formatting options](#) for more details.

**caption\_entities:** `List[MessageEntity]` | `None`

A JSON-serialized list of special entities that appear in the caption, which can be specified instead of `parse_mode`

**duration:** `int` | `None`

Duration of the voice message in seconds

**disable\_notification:** `bool` | `None`

Sends the message [silently](#). Users will receive a notification with no sound.

**protect\_content:** `bool` | `None`

Protects the contents of the sent message from forwarding and saving

**reply\_to\_message\_id:** `int` | `None`

If the message is a reply, ID of the original message

**allow\_sending\_without\_reply:** `bool` | `None`

Pass `True` if the message should be sent even if the specified replied-to message is not found

**reply\_markup:** `InlineKeyboardMarkup` | `ReplyKeyboardMarkup` | `ReplyKeyboardRemove` | `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.

## Usage

### As bot method

```
result: Message = await bot.send_voice(...)
```

## Method as object

Imports:

- `from aiogram.methods.send_voice import SendVoice`
- `alias: from aiogram.methods import SendVoice`

## With specific bot

```
result: Message = await bot(SendVoice(...))
```

## As reply into Webhook in handler

```
return SendVoice(...)
```

## As shortcut from received object

- `aiogram.types.message.Message.answer_voice()`
- `aiogram.types.message.Message.reply_voice()`
- `aiogram.types.chat_member_updated.ChatMemberUpdated.answer_voice()`
- `aiogram.types.chat_join_request.ChatJoinRequest.answer_voice()`
- `aiogram.types.chat_join_request.ChatJoinRequest.answer_voice_pm()`

## setChatAdministratorCustomTitle

Returns: bool

```
class aiogram.methods.set_chat_administrator_custom_title.SetChatAdministratorCustomTitle(*,
                                                                 chat_id:
                                                                 int
                                                                 |
                                                                 str,
                                                                 user_id:
                                                                 int,
                                                                 cus-
                                                                 tom_title:
                                                                 str,
                                                                 **ex-
                                                                 tra_data:
                                                                 Any)
```

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>

**chat\_id:** `int | str`

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

**user\_id:** `int`

Unique identifier of the target user

**custom\_title:** `str`

New custom title for the administrator; 0-16 characters, emoji are not allowed

## Usage

### As bot method

```
result: bool = await bot.set_chat_administrator_custom_title(...)
```

### Method as object

Imports:

- `from aiogram.methods.set_chat_administrator_custom_title import SetChatAdministratorCustomTitle`
- `alias: from aiogram.methods import SetChatAdministratorCustomTitle`

### With specific bot

```
result: bool = await bot(SetChatAdministratorCustomTitle(...))
```

### As reply into Webhook in handler

```
return SetChatAdministratorCustomTitle(...)
```

### As shortcut from received object

- `aiogram.types.chat.Chat.set_administrator_custom_title()`

## setChatDescription

Returns: `bool`

**class** `aiogram.methods.set_chat_description.SetChatDescription`(\*`chat_id: int | str`, `description: str | None = None`, `**extra_data: Any`)

Use this method to change the description of a group, a supergroup or a channel. The bot must be an administrator in the chat for this to work and must have the appropriate administrator rights. Returns True on success.

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

**chat\_id:** `int | str`

Unique identifier for the target chat or username of the target channel (in the format @channelusername)

**description:** `str | None`

New chat description, 0-255 characters

## Usage

### As bot method

```
result: bool = await bot.set_chat_description(...)
```

### Method as object

Imports:

- `from aiogram.methods.set_chat_description import SetChatDescription`
- `alias: from aiogram.methods import SetChatDescription`

### With specific bot

```
result: bool = await bot(SetChatDescription(...))
```

### As reply into Webhook in handler

```
return SetChatDescription(...)
```

### As shortcut from received object

- `aiogram.types.chat.Chat.set_description()`

## setChatMenuButton

Returns: `bool`

```
class aiogram.methods.set_chat_menu_button.SetChatMenuButton(*, chat_id: int | None = None,
    menu_button:
        MenuButtonCommands |
        MenuButtonWebApp |
        MenuButtonDefault | None = None,
    **extra_data: Any)
```

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

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

**chat\_id:** `int` | `None`

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

**menu\_button:** `MenuButtonCommands` | `MenuButtonWebApp` | `MenuButtonDefault` | `None`

A JSON-serialized object for the bot's new menu button. Defaults to `aiogram.types.menu_button_default.MenuButtonDefault`

## Usage

### As bot method

```
result: bool = await bot.set_chat_menu_button(...)
```

### Method as object

Imports:

- `from aiogram.methods.set_chat_menu_button import SetChatMenuButton`
- `alias: from aiogram.methods import SetChatMenuButton`

### With specific bot

```
result: bool = await bot(SetChatMenuButton(...))
```

### As reply into Webhook in handler

```
return SetChatMenuButton(...)
```

## setChatPermissions

Returns: `bool`

```
class aiogram.methods.set_chat_permissions.SetChatPermissions(*, chat_id: int
                                                                | str, permissions: ChatPermissions,
                                                                use_independent_chat_permissions:
                                                                bool | None = None, **extra_data:
                                                                Any)
```

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` administrator rights. Returns True on success.

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

**chat\_id:** `int` | `str`

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

**permissions:** *ChatPermissions*

A JSON-serialized object for new default chat permissions

**use\_independent\_chat\_permissions:** `bool` | `None`

Pass `True` if chat permissions are set independently. Otherwise, the *can\_send\_other\_messages* and *can\_add\_web\_page\_previews* permissions will imply the *can\_send\_messages*, *can\_send\_audios*, *can\_send\_documents*, *can\_send\_photos*, *can\_send\_videos*, *can\_send\_video\_notes*, and *can\_send\_voice\_notes* permissions; the *can\_send\_polls* permission will imply the *can\_send\_messages* permission.

## Usage

### As bot method

```
result: bool = await bot.set_chat_permissions(...)
```

### Method as object

Imports:

- `from aiogram.methods.set_chat_permissions import SetChatPermissions`
- alias: `from aiogram.methods import SetChatPermissions`

### With specific bot

```
result: bool = await bot(SetChatPermissions(...))
```

### As reply into Webhook in handler

```
return SetChatPermissions(...)
```

### As shortcut from received object

- *aiogram.types.chat.Chat.set\_permissions()*

## setChatPhoto

Returns: `bool`

**class** `aiogram.methods.set_chat_photo.SetChatPhoto`(\**chat\_id*: `int` | `str`, *photo*: `InputFile`, *\*\*extra\_data*: *Any*)

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 administrator rights. Returns `True` on success.

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

**chat\_id:** `int | str`

Unique identifier for the target chat or username of the target channel (in the format @channelusername)

**photo:** `InputFile`

New chat photo, uploaded using multipart/form-data

## Usage

### As bot method

```
result: bool = await bot.set_chat_photo(...)
```

### Method as object

Imports:

- `from aiogram.methods.set_chat_photo import SetChatPhoto`
- `alias: from aiogram.methods import SetChatPhoto`

### With specific bot

```
result: bool = await bot(SetChatPhoto(...))
```

### As shortcut from received object

- `aiogram.types.chat.Chat.set_photo()`

## setChatStickerSet

Returns: `bool`

```
class aiogram.methods.set_chat_sticker_set.SetChatStickerSet(*, chat_id: int | str,
                                                             sticker_set_name: str, **extra_data:
                                                             Any)
```

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 administrator rights. Use the field `can_set_sticker_set` optionally returned in `aiogram.methods.get_chat.GetChat` requests to check if the bot can use this method. Returns `True` on success.

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

**chat\_id:** `int | str`

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

**sticker\_set\_name:** `str`

Name of the sticker set to be set as the group sticker set



## Usage

### As bot method

```
result: bool = await bot.set_chat_sticker_set(...)
```

### Method as object

Imports:

- `from aiogram.methods.set_chat_sticker_set import SetChatStickerSet`
- `alias: from aiogram.methods import SetChatStickerSet`

### With specific bot

```
result: bool = await bot(SetChatStickerSet(...))
```

### As reply into Webhook in handler

```
return SetChatStickerSet(...)
```

### As shortcut from received object

- `aiogram.types.chat.Chat.set_sticker_set()`

## setChatTitle

Returns: bool

**class** `aiogram.methods.set_chat_title.SetChatTitle(*, chat_id: int | str, title: str, **extra_data: Any)`

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 administrator rights. Returns True on success.

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

**chat\_id:** `int | str`

Unique identifier for the target chat or username of the target channel (in the format @channelusername)

**title:** `str`

New chat title, 1-128 characters

## Usage

### As bot method

```
result: bool = await bot.set_chat_title(...)
```

### Method as object

Imports:

- `from aiogram.methods.set_chat_title import SetChatTitle`
- `alias: from aiogram.methods import SetChatTitle`

### With specific bot

```
result: bool = await bot(SetChatTitle(...))
```

### As reply into Webhook in handler

```
return SetChatTitle(...)
```

### As shortcut from received object

- `aiogram.types.chat.Chat.set_title()`

## setMyCommands

Returns: bool

```
class aiogram.methods.set_my_commands.SetMyCommands(*, commands: List[BotCommand], scope:
    BotCommandScopeDefault |
    BotCommandScopeAllPrivateChats |
    BotCommandScopeAllGroupChats |
    BotCommandScopeAllChatAdministrators |
    BotCommandScopeChat |
    BotCommandScopeChatAdministrators |
    BotCommandScopeChatMember | None = None,
    language_code: str | None = None,
    **extra_data: Any)
```

Use this method to change the list of the bot's commands. See [this manual](#) for more details about bot commands. Returns True on success.

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

**commands:** List[[BotCommand](#)]

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:** *BotCommandScopeDefault* | *BotCommandScopeAllPrivateChats* | *BotCommandScopeAllGroupChats* | *BotCommandScopeAllChatAdministrators* | *BotCommandScopeChat* | *BotCommandScopeChatAdministrators* | *BotCommandScopeChatMember* | **None**

A JSON-serialized object, describing scope of users for which the commands are relevant. Defaults to *aiogram.types.bot\_command\_scope\_default.BotCommandScopeDefault*.

**language\_code:** **str** | **None**

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

## Usage

### As bot method

```
result: bool = await bot.set_my_commands(...)
```

### Method as object

Imports:

- from aiogram.methods.set\_my\_commands import SetMyCommands
- alias: from aiogram.methods import SetMyCommands

### With specific bot

```
result: bool = await bot(SetMyCommands(...))
```

### As reply into Webhook in handler

```
return SetMyCommands(...)
```

### setMyDefaultAdministratorRights

Returns: bool

```
class aiogram.methods.set_my_default_administrator_rights.SetMyDefaultAdministratorRights(*,
                                                                                          rights:
                                                                                          ChatAd-
                                                                                          min-
                                                                                          is-
                                                                                          tra-
                                                                                          tor-
                                                                                          Rights
                                                                                          |
                                                                                          None
                                                                                          =
                                                                                          None,
                                                                                          for_channels:
                                                                                          bool
                                                                                          |
                                                                                          None
                                                                                          =
                                                                                          None,
                                                                                          **ex-
                                                                                          tra_data:
                                                                                          Any)
```

Use this method to change the default administrator rights requested by the bot when it's added as an administrator to groups or channels. These rights will be suggested to users, but they are free to modify the list before adding the bot. Returns True on success.

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

**rights:** `ChatAdministratorRights` | `None`

A JSON-serialized object describing new default administrator rights. If not specified, the default administrator rights will be cleared.

**for\_channels:** `bool` | `None`

Pass True to change the default administrator rights of the bot in channels. Otherwise, the default administrator rights of the bot for groups and supergroups will be changed.

## Usage

### As bot method

```
result: bool = await bot.set_my_default_administrator_rights(...)
```

### Method as object

Imports:

- `from aiogram.methods.set_my_default_administrator_rights import SetMyDefaultAdministratorRights`
- `alias: from aiogram.methods import SetMyDefaultAdministratorRights`

### With specific bot

```
result: bool = await bot(SetMyDefaultAdministratorRights(...))
```

### As reply into Webhook in handler

```
return SetMyDefaultAdministratorRights(...)
```

## setMyDescription

Returns: bool

```
class aiogram.methods.set_my_description.SetMyDescription(*, description: str | None = None,
                                                         language_code: str | None = None,
                                                         **extra_data: Any)
```

Use this method to change the bot's description, which is shown in the chat with the bot if the chat is empty. Returns True on success.

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

**description:** str | None

New bot description; 0-512 characters. Pass an empty string to remove the dedicated description for the given language.

**language\_code:** str | None

A two-letter ISO 639-1 language code. If empty, the description will be applied to all users for whose language there is no dedicated description.

## Usage

### As bot method

```
result: bool = await bot.set_my_description(...)
```

## Method as object

Imports:

- `from aiogram.methods.set_my_description import SetMyDescription`
- `alias: from aiogram.methods import SetMyDescription`

### With specific bot

```
result: bool = await bot(SetMyDescription(...))
```

### As reply into Webhook in handler

```
return SetMyDescription(...)
```

## setMyName

Returns: bool

```
class aiogram.methods.set_my_name.SetMyName(*, name: str | None = None, language_code: str | None = None, **extra_data: Any)
```

Use this method to change the bot's name. Returns True on success.

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

**name:** str | None

New bot name; 0-64 characters. Pass an empty string to remove the dedicated name for the given language.

**language\_code:** str | None

A two-letter ISO 639-1 language code. If empty, the name will be shown to all users for whose language there is no dedicated name.

## Usage

### As bot method

```
result: bool = await bot.set_my_name(...)
```

## Method as object

Imports:

- `from aiogram.methods.set_my_name import SetMyName`
- `alias: from aiogram.methods import SetMyName`

### With specific bot

```
result: bool = await bot(SetMyName(...))
```

### As reply into Webhook in handler

```
return SetMyName(...)
```

### setMyShortDescription

Returns: bool

```
class aiogram.methods.set_my_short_description.SetMyShortDescription(*, short_description: str |
                                                                    None = None,
                                                                    language_code: str | None
                                                                    = None, **extra_data:
                                                                    Any)
```

Use this method to change the bot's short description, which is shown on the bot's profile page and is sent together with the link when users share the bot. Returns True on success.

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

**short\_description: str | None**

New short description for the bot; 0-120 characters. Pass an empty string to remove the dedicated short description for the given language.

**language\_code: str | None**

A two-letter ISO 639-1 language code. If empty, the short description will be applied to all users for whose language there is no dedicated short description.

### Usage

#### As bot method

```
result: bool = await bot.set_my_short_description(...)
```

#### Method as object

Imports:

- `from aiogram.methods.set_my_short_description import SetMyShortDescription`
- `alias: from aiogram.methods import SetMyShortDescription`

#### With specific bot

```
result: bool = await bot(SetMyShortDescription(...))
```

### As reply into Webhook in handler

```
return SetMyShortDescription(...)
```

### unbanChatMember

Returns: bool

```
class aiogram.methods.unban_chat_member.UnbanChatMember(*, chat_id: int | str, user_id: int,
                                                         only_if_banned: bool | None = None,
                                                         **extra_data: Any)
```

Use this method to unban a previously banned 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>

**chat\_id: int | str**

Unique identifier for the target group or username of the target supergroup or channel (in the format @channelusername)

**user\_id: int**

Unique identifier of the target user

**only\_if\_banned: bool | None**

Do nothing if the user is not banned

### Usage

#### As bot method

```
result: bool = await bot.unban_chat_member(...)
```

### Method as object

Imports:

- `from aiogram.methods.unban_chat_member import UnbanChatMember`
- `alias: from aiogram.methods import UnbanChatMember`



### With specific bot

```
result: bool = await bot(UnbanChatMember(...))
```

### As reply into Webhook in handler

```
return UnbanChatMember(...)
```

### As shortcut from received object

- `aiogram.types.chat.Chat.unban()`

### unbanChatSenderChat

Returns: bool

```
class aiogram.methods.unban_chat_sender_chat.UnbanChatSenderChat(*, chat_id: int | str,
                                                                    sender_chat_id: int,
                                                                    **extra_data: Any)
```

Use this method to unban a previously banned channel chat in a supergroup or channel. The bot must be an administrator for this to work and must have the appropriate administrator rights. Returns True on success.

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

**chat\_id: int | str**

Unique identifier for the target chat or username of the target channel (in the format @channelusername)

**sender\_chat\_id: int**

Unique identifier of the target sender chat

### Usage

#### As bot method

```
result: bool = await bot.unban_chat_sender_chat(...)
```

### Method as object

Imports:

- `from aiogram.methods.unban_chat_sender_chat import UnbanChatSenderChat`
- `alias: from aiogram.methods import UnbanChatSenderChat`

### With specific bot

```
result: bool = await bot(UnbanChatSenderChat(...))
```

### As reply into Webhook in handler

```
return UnbanChatSenderChat(...)
```

### As shortcut from received object

- `aiogram.types.chat.Chat.unban_sender_chat()`

## unhideGeneralForumTopic

Returns: bool

```
class aiogram.methods.unhide_general_forum_topic.UnhideGeneralForumTopic(*, chat_id: int | str,
                                                                           **extra_data: Any)
```

Use this method to unhide the ‘General’ topic in a forum supergroup chat. The bot must be an administrator in the chat for this to work and must have the `can_manage_topics` administrator rights. Returns True on success.

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

**chat\_id: int | str**

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

## Usage

### As bot method

```
result: bool = await bot.unhide_general_forum_topic(...)
```

### Method as object

Imports:

- `from aiogram.methods.unhide_general_forum_topic import UnhideGeneralForumTopic`
- `alias: from aiogram.methods import UnhideGeneralForumTopic`

### With specific bot

```
result: bool = await bot(UnhideGeneralForumTopic(...))
```

### As reply into Webhook in handler

```
return UnhideGeneralForumTopic(...)
```

## unpinAllChatMessages

Returns: bool

```
class aiogram.methods.unpin_all_chat_messages.UnpinAllChatMessages(*, chat_id: int | str,
                                                                    **extra_data: Any)
```

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’ administrator right in a supergroup or ‘can\_edit\_messages’ administrator right in a channel. Returns True on success.

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

**chat\_id: int | str**

Unique identifier for the target chat or username of the target channel (in the format @channelusername)

### Usage

#### As bot method

```
result: bool = await bot.unpin_all_chat_messages(...)
```

### Method as object

Imports:

- `from aiogram.methods.unpin_all_chat_messages import UnpinAllChatMessages`
- `alias: from aiogram.methods import UnpinAllChatMessages`

### With specific bot

```
result: bool = await bot(UnpinAllChatMessages(...))
```

### As reply into Webhook in handler

```
return UnpinAllChatMessages(...)
```

### As shortcut from received object

- `aiogram.types.chat.Chat.unpin_all_messages()`

### unpinAllForumTopicMessages

Returns: bool

```
class aiogram.methods.unpin_all_forum_topic_messages.UnpinAllForumTopicMessages(*, chat_id:
                                                                    int | str,
                                                                    mes-
                                                                    sage_thread_id:
                                                                    int, **ex-
                                                                    tra_data:
                                                                    Any)
```

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

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

**chat\_id: int | str**

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

**message\_thread\_id: int**

Unique identifier for the target message thread of the forum topic

### Usage

#### As bot method

```
result: bool = await bot.unpin_all_forum_topic_messages(...)
```

### Method as object

Imports:

- `from aiogram.methods.unpin_all_forum_topic_messages import UnpinAllForumTopicMessages`
- `alias: from aiogram.methods import UnpinAllForumTopicMessages`

### With specific bot

```
result: bool = await bot(UnpinAllForumTopicMessages(...))
```

### As reply into Webhook in handler

```
return UnpinAllForumTopicMessages(...)
```

## unpinAllGeneralForumTopicMessages

Returns: bool

```
class aiogram.methods.unpin_all_general_forum_topic_messages.UnpinAllGeneralForumTopicMessages(*,
                                                                                               chat_id:
                                                                                               int
                                                                                               |
                                                                                               str,
                                                                                               **ex-
                                                                                               tra_data:
                                                                                               Any)
```

Use this method to clear the list of pinned messages in a General forum topic. The bot must be an administrator in the chat for this to work and must have the *can\_pin\_messages* administrator right in the supergroup. Returns True on success.

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

**chat\_id:** int | str

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

## Usage

### As bot method

```
result: bool = await bot.unpin_all_general_forum_topic_messages(...)
```

### Method as object

Imports:

- from aiogram.methods.unpin\_all\_general\_forum\_topic\_messages import UnpinAllGeneralForumTopicMessages
- alias: from aiogram.methods import UnpinAllGeneralForumTopicMessages

### With specific bot

```
result: bool = await bot(UnpinAllGeneralForumTopicMessages(...))
```

### As reply into Webhook in handler

```
return UnpinAllGeneralForumTopicMessages(...)
```

### As shortcut from received object

- `aiogram.types.chat.Chat.unpin_all_general_forum_topic_messages()`

## unpinChatMessage

Returns: bool

```
class aiogram.methods.unpin_chat_message.UnpinChatMessage(*, chat_id: int | str, message_id: int |  
                                                           None = None, **extra_data: Any)
```

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’ administrator right in a supergroup or ‘can\_edit\_messages’ administrator right in a channel. Returns True on success.

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

**chat\_id: int | str**

Unique identifier for the target chat or username of the target channel (in the format @channelusername)

**message\_id: int | None**

Identifier of a message to unpin. If not specified, the most recent pinned message (by sending date) will be unpinned.

## Usage

### As bot method

```
result: bool = await bot.unpin_chat_message(...)
```

### Method as object

Imports:

- `from aiogram.methods.unpin_chat_message import UnpinChatMessage`
- `alias: from aiogram.methods import UnpinChatMessage`

### With specific bot

```
result: bool = await bot(UnpinChatMessage(...))
```

### As reply into Webhook in handler

```
return UnpinChatMessage(...)
```

### As shortcut from received object

- `aiogram.types.message.Message.unpin()`
- `aiogram.types.chat.Chat.unpin_message()`

## Payments

### answerPreCheckoutQuery

Returns: bool

```
class aiogram.methods.answer_pre_checkout_query.AnswerPreCheckoutQuery(*,
                                                                    pre_checkout_query_id:
                                                                    str, ok: bool,
                                                                    error_message: str |
                                                                    None = None,
                                                                    **extra_data: Any)
```

Once the user has confirmed their payment and shipping details, the Bot API sends the final confirmation in the form of an `aiogram.types.update.Update` with the field `pre_checkout_query`. Use this method to respond to such pre-checkout queries. On success, `True` is returned. **Note:** The Bot API must receive an answer within 10 seconds after the pre-checkout query was sent.

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

**pre\_checkout\_query\_id:** str

Unique identifier for the query to be answered

**ok:** bool

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:** str | None

Required if `ok` is `False`. Error message in human readable form that explains the reason for failure to proceed with the checkout (e.g. “Sorry, somebody just bought the last of our amazing black T-shirts while you were busy filling out your payment details. Please choose a different color or garment!”). Telegram will display this message to the user.

## Usage

### As bot method

```
result: bool = await bot.answer_pre_checkout_query(...)
```

### Method as object

Imports:

- `from aiogram.methods.answer_pre_checkout_query import AnswerPreCheckoutQuery`
- `alias: from aiogram.methods import AnswerPreCheckoutQuery`

### With specific bot

```
result: bool = await bot(AnswerPreCheckoutQuery(...))
```

### As reply into Webhook in handler

```
return AnswerPreCheckoutQuery(...)
```

### As shortcut from received object

- `aiogram.types.pre_checkout_query.PreCheckoutQuery.answer()`

## answerShippingQuery

Returns: bool

```
class aiogram.methods.answer_shipping_query.AnswerShippingQuery(*, shipping_query_id: str, ok: bool, shipping_options: List[ShippingOption] | None = None, error_message: str | None = None, **extra_data: Any)
```

If you sent an invoice requesting a shipping address and the parameter *is\_flexible* was specified, the Bot API will send an `aiogram.types.update.Update` with a *shipping\_query* field to the bot. Use this method to reply to shipping queries. On success, True is returned.

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

**shipping\_query\_id:** str

Unique identifier for the query to be answered

**ok:** bool

Pass 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:** `List[ShippingOption] | None`

Required if *ok* is True. A JSON-serialized array of available shipping options.

**error\_message:** `str | None`

Required if *ok* is False. Error message in human readable form that explains why it is impossible to complete the order (e.g. “Sorry, delivery to your desired address is unavailable”). Telegram will display this message to the user.

## Usage

### As bot method

```
result: bool = await bot.answer_shipping_query(...)
```

### Method as object

Imports:

- `from aiogram.methods.answer_shipping_query import AnswerShippingQuery`
- alias: `from aiogram.methods import AnswerShippingQuery`

### With specific bot

```
result: bool = await bot(AnswerShippingQuery(...))
```

### As reply into Webhook in handler

```
return AnswerShippingQuery(...)
```

### As shortcut from received object

- `aiogram.types.shipping_query.ShippingQuery.answer()`

### createInvoiceLink

Returns: `str`

```
class aiogram.methods.create_invoice_link.CreateInvoiceLink(*, title: str, description: str, payload: str, provider_token: str, currency: str, prices: List[LabeledPrice], max_tip_amount: int | None = None, suggested_tip_amounts: List[int] | None = None, provider_data: str | None = None, photo_url: str | None = None, photo_size: int | None = None, photo_width: int | None = None, photo_height: int | None = None, need_name: bool | None = None, need_phone_number: bool | None = None, need_email: bool | None = None, need_shipping_address: bool | None = None, send_phone_number_to_provider: bool | None = None, send_email_to_provider: bool | None = None, is_flexible: bool | None = None, **extra_data: Any)
```

Use this method to create a link for an invoice. Returns the created invoice link as *String* on success.

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

**title: str**

Product name, 1-32 characters

**description: str**

Product description, 1-255 characters

**payload: str**

Bot-defined invoice payload, 1-128 bytes. This will not be displayed to the user, use for your internal processes.

**provider\_token: str**

Payment provider token, obtained via [BotFather](#)

**currency: str**

Three-letter ISO 4217 currency code, see [more on currencies](#)

**prices: List[LabeledPrice]**

Price breakdown, a JSON-serialized list of components (e.g. product price, tax, discount, delivery cost, delivery tax, bonus, etc.)

**max\_tip\_amount: int | None**

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: List[int] | None**

A JSON-serialized array of suggested amounts of tips in the *smallest units* of the currency (integer, **not** float/double). At most 4 suggested tip amounts can be specified. The suggested tip amounts must be positive, passed in a strictly increased order and must not exceed *max\_tip\_amount*.

**provider\_data:** `str | None`

JSON-serialized data about the invoice, which will be shared with the payment provider. A detailed description of required fields should be provided by the payment provider.

**photo\_url:** `str | None`

URL of the product photo for the invoice. Can be a photo of the goods or a marketing image for a service.

**photo\_size:** `int | None`

Photo size in bytes

**photo\_width:** `int | None`

Photo width

**photo\_height:** `int | None`

Photo height

**need\_name:** `bool | None`

Pass True if you require the user's full name to complete the order

**need\_phone\_number:** `bool | None`

Pass True if you require the user's phone number to complete the order

**need\_email:** `bool | None`

Pass True if you require the user's email address to complete the order

**need\_shipping\_address:** `bool | None`

Pass True if you require the user's shipping address to complete the order

**send\_phone\_number\_to\_provider:** `bool | None`

Pass True if the user's phone number should be sent to the provider

**send\_email\_to\_provider:** `bool | None`

Pass True if the user's email address should be sent to the provider

**is\_flexible:** `bool | None`

Pass True if the final price depends on the shipping method

## Usage

### As bot method

```
result: str = await bot.create_invoice_link(...)
```

### Method as object

Imports:

- `from aiogram.methods.create_invoice_link import CreateInvoiceLink`
- `alias: from aiogram.methods import CreateInvoiceLink`

### With specific bot

```
result: str = await bot(CreateInvoiceLink(...))
```

### As reply into Webhook in handler

```
return CreateInvoiceLink(...)
```

## sendInvoice

Returns: Message

```
class aiogram.methods.send_invoice.SendInvoice(*, chat_id: int | str, title: str, description: str, payload: str, provider_token: str, currency: str, prices: List[LabeledPrice], message_thread_id: int | None = None, max_tip_amount: int | None = None, suggested_tip_amounts: List[int] | None = None, start_parameter: str | None = None, provider_data: str | None = None, photo_url: str | None = None, photo_size: int | None = None, photo_width: int | None = None, photo_height: int | None = None, need_name: bool | None = None, need_phone_number: bool | None = None, need_email: bool | None = None, need_shipping_address: bool | None = None, send_phone_number_to_provider: bool | None = None, send_email_to_provider: bool | None = None, is_flexible: bool | None = None, disable_notification: bool | None = None, protect_content: bool | None = sentinel.UNSET_PROTECT_CONTENT, reply_to_message_id: int | None = None, allow_sending_without_reply: bool | None = None, reply_markup: InlineKeyboardMarkup | None = None, **extra_data: Any)
```

Use this method to send invoices. On success, the sent `aiogram.types.message.Message` is returned.

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

**chat\_id:** int | str

Unique identifier for the target chat or username of the target channel (in the format @channelusername)

**title:** str

Product name, 1-32 characters

**description:** str

Product description, 1-255 characters

**payload:** str

Bot-defined invoice payload, 1-128 bytes. This will not be displayed to the user, use for your internal processes.

**provider\_token:** str

Payment provider token, obtained via @BotFather

**currency:** `str`Three-letter ISO 4217 currency code, see [more on currencies](#)**prices:** `List[LabeledPrice]`

Price breakdown, a JSON-serialized list of components (e.g. product price, tax, discount, delivery cost, delivery tax, bonus, etc.)

**message\_thread\_id:** `int | None`

Unique identifier for the target message thread (topic) of the forum; for forum supergroups only

**max\_tip\_amount:** `int | None`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:** `List[int] | None`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:** `str | None`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:** `str | None`

JSON-serialized data about the invoice, which will be shared with the payment provider. A detailed description of required fields should be provided by the payment provider.

**photo\_url:** `str | None`

URL of the product photo for the invoice. Can be a photo of the goods or a marketing image for a service. People like it better when they see what they are paying for.

**photo\_size:** `int | None`

Photo size in bytes

**photo\_width:** `int | None`

Photo width

**photo\_height:** `int | None`

Photo height

**need\_name:** `bool | None`

Pass True if you require the user's full name to complete the order

**need\_phone\_number:** `bool | None`

Pass True if you require the user's phone number to complete the order

**need\_email:** `bool | None`

Pass True if you require the user's email address to complete the order

**need\_shipping\_address:** `bool | None`

Pass True if you require the user's shipping address to complete the order

**send\_phone\_number\_to\_provider:** `bool | None`

Pass True if the user's phone number should be sent to provider

**send\_email\_to\_provider:** `bool | None`

Pass True if the user's email address should be sent to provider

**is\_flexible:** `bool | None`

Pass True if the final price depends on the shipping method

**disable\_notification:** `bool | None`

Sends the message `silently`. Users will receive a notification with no sound.

**protect\_content:** `bool | None`

Protects the contents of the sent message from forwarding and saving

**reply\_to\_message\_id:** `int | None`

If the message is a reply, ID of the original message

**allow\_sending\_without\_reply:** `bool | None`

Pass True if the message should be sent even if the specified replied-to message is not found

**reply\_markup:** `InlineKeyboardMarkup | None`

A JSON-serialized object for an `inline keyboard`. If empty, one 'Pay total price' button will be shown. If not empty, the first button must be a Pay button.

## Usage

### As bot method

```
result: Message = await bot.send_invoice(...)
```

### Method as object

Imports:

- `from aiogram.methods.send_invoice import SendInvoice`
- `alias: from aiogram.methods import SendInvoice`

### With specific bot

```
result: Message = await bot(SendInvoice(...))
```

### As reply into Webhook in handler

```
return SendInvoice(...)
```

### As shortcut from received object

- `aiogram.types.message.Message.answer_invoice()`
- `aiogram.types.message.Message.reply_invoice()`
- `aiogram.types.chat_member_updated.ChatMemberUpdated.answer_invoice()`
- `aiogram.types.chat_join_request.ChatJoinRequest.answer_invoice()`
- `aiogram.types.chat_join_request.ChatJoinRequest.answer_invoice_pm()`

## Stickers

### addStickerToSet

Returns: bool

```
class aiogram.methods.add_sticker_to_set.AddStickerToSet(*, user_id: int, name: str, sticker:
                                                         InputSticker, **extra_data: Any)
```

Use this method to add a new sticker to a set created by the bot. The format of the added sticker must match the format of the other stickers in the set. Emoji sticker sets can have up to 200 stickers. Animated and video sticker sets can have up to 50 stickers. Static sticker sets can have up to 120 stickers. Returns True on success.

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

**user\_id:** `int`

User identifier of sticker set owner

**name:** `str`

Sticker set name

**sticker:** `InputSticker`

A JSON-serialized object with information about the added sticker. If exactly the same sticker had already been added to the set, then the set isn't changed.

## Usage

### As bot method

```
result: bool = await bot.add_sticker_to_set(...)
```

## Method as object

Imports:

- `from aiogram.methods.add_sticker_to_set import AddStickerToSet`
- `alias: from aiogram.methods import AddStickerToSet`

## With specific bot

```
result: bool = await bot(AddStickerToSet(...))
```

## As reply into Webhook in handler

```
return AddStickerToSet(...)
```

## createNewStickerSet

Returns: bool

```
class aiogram.methods.create_new_sticker_set.CreateNewStickerSet(*, user_id: int, name: str, title: str, stickers: List[InputSticker], sticker_format: str, sticker_type: str | None = None, needs_repainting: bool | None = None, **extra_data: Any)
```

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. Returns True on success.

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

**user\_id: int**

User identifier of created sticker set owner

**name: str**

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: str**

Sticker set title, 1-64 characters

**stickers: List[InputSticker]**

A JSON-serialized list of 1-50 initial stickers to be added to the sticker set

**sticker\_format: str**

Format of stickers in the set, must be one of 'static', 'animated', 'video'

**sticker\_type: str | None**

Type of stickers in the set, pass 'regular', 'mask', or 'custom\_emoji'. By default, a regular sticker set is created.



**needs\_repainting:** `bool` | `None`

Pass `True` if stickers in the sticker set must be repainted to the color of text when used in messages, the accent color if used as emoji status, white on chat photos, or another appropriate color based on context; for custom emoji sticker sets only

## Usage

### As bot method

```
result: bool = await bot.create_new_sticker_set(...)
```

### Method as object

Imports:

- `from aiogram.methods.create_new_sticker_set import CreateNewStickerSet`
- `alias: from aiogram.methods import CreateNewStickerSet`

### With specific bot

```
result: bool = await bot(CreateNewStickerSet(...))
```

### As reply into Webhook in handler

```
return CreateNewStickerSet(...)
```

## deleteStickerFromSet

Returns: `bool`

```
class aiogram.methods.delete_sticker_from_set.DeleteStickerFromSet(*, sticker: str, **extra_data: Any)
```

Use this method to delete a sticker from a set created by the bot. Returns `True` on success.

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

**sticker:** `str`

File identifier of the sticker

## Usage

### As bot method

```
result: bool = await bot.delete_sticker_from_set(...)
```

### Method as object

Imports:

- `from aiogram.methods.delete_sticker_from_set import DeleteStickerFromSet`
- `alias: from aiogram.methods import DeleteStickerFromSet`

### With specific bot

```
result: bool = await bot(DeleteStickerFromSet(...))
```

### As reply into Webhook in handler

```
return DeleteStickerFromSet(...)
```

### As shortcut from received object

- `aiogram.types.sticker.Sticker.delete_from_set()`

## deleteStickerSet

Returns: bool

**class** `aiogram.methods.delete_sticker_set.DeleteStickerSet(*, name: str, **extra_data: Any)`

Use this method to delete a sticker set that was created by the bot. Returns True on success.

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

**name:** `str`

Sticker set name

## Usage

### As bot method

```
result: bool = await bot.delete_sticker_set(...)
```

## Method as object

Imports:

- `from aiogram.methods.delete_sticker_set import DeleteStickerSet`
- `alias: from aiogram.methods import DeleteStickerSet`

## With specific bot

```
result: bool = await bot(DeleteStickerSet(...))
```

## As reply into Webhook in handler

```
return DeleteStickerSet(...)
```

## getCustomEmojiStickers

Returns: `List[Sticker]`

```
class aiogram.methods.get_custom_emoji_stickers.GetCustomEmojiStickers(*, custom_emoji_ids:
                                                                    List[str], **extra_data:
                                                                    Any)
```

Use this method to get information about custom emoji stickers by their identifiers. Returns an Array of *aiogram.types.sticker.Sticker* objects.

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

**custom\_emoji\_ids:** `List[str]`

List of custom emoji identifiers. At most 200 custom emoji identifiers can be specified.

## Usage

### As bot method

```
result: List[Sticker] = await bot.get_custom_emoji_stickers(...)
```

## Method as object

Imports:

- `from aiogram.methods.get_custom_emoji_stickers import GetCustomEmojiStickers`
- `alias: from aiogram.methods import GetCustomEmojiStickers`

### With specific bot

```
result: List[Sticker] = await bot(GetCustomEmojiStickers(...))
```

### getStickerSet

Returns: StickerSet

**class** aiogram.methods.get\_sticker\_set.**GetStickerSet**(\*, name: str, \*\*extra\_data: Any)

Use this method to get a sticker set. On success, a `aiogram.types.sticker_set.StickerSet` object is returned.

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

**name:** str

Name of the sticker set

### Usage

#### As bot method

```
result: StickerSet = await bot.get_sticker_set(...)
```

### Method as object

Imports:

- `from aiogram.methods.get_sticker_set import GetStickerSet`
- `alias: from aiogram.methods import GetStickerSet`

### With specific bot

```
result: StickerSet = await bot(GetStickerSet(...))
```

### sendSticker

Returns: Message

**class** aiogram.methods.send\_sticker.**SendSticker**(\*, chat\_id: int | str, sticker: InputFile | str, message\_thread\_id: int | None = None, emoji: str | None = None, disable\_notification: bool | None = None, protect\_content: bool | None = sentinel.UNSET\_PROTECT\_CONTENT, reply\_to\_message\_id: int | None = None, allow\_sending\_without\_reply: bool | None = None, reply\_markup: InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply | None = None, \*\*extra\_data: Any)

Use this method to send static `.WEBP`, `animated.TGS`, or `video.WEBM` stickers. On success, the sent `aiogram.types.message.Message` is returned.

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

**chat\_id:** `int` | `str`

Unique identifier for the target chat or username of the target channel (in the format `@channelusername`)

**sticker:** `InputFile` | `str`

Sticker to send. Pass a `file_id` as `String` to send a file that exists on the Telegram servers (recommended), pass an HTTP URL as a `String` for Telegram to get a `.WEBP` sticker from the Internet, or upload a new `.WEBP` or `.TGS` sticker using multipart/form-data. *More information on Sending Files »*. Video stickers can only be sent by a `file_id`. Animated stickers can't be sent via an HTTP URL.

**message\_thread\_id:** `int` | `None`

Unique identifier for the target message thread (topic) of the forum; for forum supergroups only

**emoji:** `str` | `None`

Emoji associated with the sticker; only for just uploaded stickers

**disable\_notification:** `bool` | `None`

Sends the message `silently`. Users will receive a notification with no sound.

**protect\_content:** `bool` | `None`

Protects the contents of the sent message from forwarding and saving

**reply\_to\_message\_id:** `int` | `None`

If the message is a reply, ID of the original message

**allow\_sending\_without\_reply:** `bool` | `None`

Pass `True` if the message should be sent even if the specified replied-to message is not found

**reply\_markup:** `InlineKeyboardMarkup` | `ReplyKeyboardMarkup` | `ReplyKeyboardRemove` | `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.

## Usage

### As bot method

```
result: Message = await bot.send_sticker(...)
```

### Method as object

Imports:

- `from aiogram.methods.send_sticker import SendSticker`
- `alias: from aiogram.methods import SendSticker`

### With specific bot

```
result: Message = await bot(SendSticker(...))
```

### As reply into Webhook in handler

```
return SendSticker(...)
```

### As shortcut from received object

- `aiogram.types.message.Message.answer_sticker()`
- `aiogram.types.message.Message.reply_sticker()`
- `aiogram.types.chat_member_updated.ChatMemberUpdated.answer_sticker()`
- `aiogram.types.chat_join_request.ChatJoinRequest.answer_sticker()`
- `aiogram.types.chat_join_request.ChatJoinRequest.answer_sticker_pm()`

### setCustomEmojiStickerSetThumbnail

Returns: bool

```
class aiogram.methods.set_custom_emoji_sticker_set_thumbnail.SetCustomEmojiStickerSetThumbnail(*,
                                                                                             name:
                                                                                             str,
                                                                                             cus-
                                                                                             tom_emoji_
                                                                                             str
                                                                                             |
                                                                                             None
                                                                                             =
                                                                                             None,
                                                                                             **ex-
                                                                                             tra_data:
                                                                                             Any)
```

Use this method to set the thumbnail of a custom emoji sticker set. Returns `True` on success.

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

**name:** `str`

Sticker set name

**custom\_emoji\_id:** `str | None`

Custom emoji identifier of a sticker from the sticker set; pass an empty string to drop the thumbnail and use the first sticker as the thumbnail.

## Usage

### As bot method

```
result: bool = await bot.set_custom_emoji_sticker_set_thumbnail(...)
```

### Method as object

Imports:

- `from aiogram.methods.set_custom_emoji_sticker_set_thumbnail import SetCustomEmojiStickerSetThumbnail`
- `alias: from aiogram.methods import SetCustomEmojiStickerSetThumbnail`

### With specific bot

```
result: bool = await bot(SetCustomEmojiStickerSetThumbnail(...))
```

### As reply into Webhook in handler

```
return SetCustomEmojiStickerSetThumbnail(...)
```

## setStickerEmojiList

Returns: bool

```
class aiogram.methods.set_sticker_emoji_list.SetStickerEmojiList(*, sticker: str, emoji_list:
                                                                    List[str], **extra_data: Any)
```

Use this method to change the list of emoji assigned to a regular or custom emoji sticker. The sticker must belong to a sticker set created by the bot. Returns True on success.

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

**sticker: str**

File identifier of the sticker

**emoji\_list: List[str]**

A JSON-serialized list of 1-20 emoji associated with the sticker

## Usage

### As bot method

```
result: bool = await bot.set_sticker_emoji_list(...)
```

### Method as object

Imports:

- `from aiogram.methods.set_sticker_emoji_list import SetStickerEmojiList`
- `alias: from aiogram.methods import SetStickerEmojiList`

### With specific bot

```
result: bool = await bot(SetStickerEmojiList(...))
```

### As reply into Webhook in handler

```
return SetStickerEmojiList(...)
```

## setStickerKeywords

Returns: bool

```
class aiogram.methods.set_sticker_keywords.SetStickerKeywords(*, sticker: str, keywords: List[str] |  
                                                                None = None, **extra_data: Any)
```

Use this method to change search keywords assigned to a regular or custom emoji sticker. The sticker must belong to a sticker set created by the bot. Returns True on success.

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

**sticker: str**

File identifier of the sticker

**keywords: List[str] | None**

A JSON-serialized list of 0-20 search keywords for the sticker with total length of up to 64 characters

## Usage

### As bot method

```
result: bool = await bot.set_sticker_keywords(...)
```



## Method as object

Imports:

- `from aiogram.methods.set_sticker_keywords import SetStickerKeywords`
- `alias: from aiogram.methods import SetStickerKeywords`

## With specific bot

```
result: bool = await bot(SetStickerKeywords(...))
```

## As reply into Webhook in handler

```
return SetStickerKeywords(...)
```

## setStickerMaskPosition

Returns: bool

```
class aiogram.methods.set_sticker_mask_position.SetStickerMaskPosition(*, sticker: str,
    mask_position:
        MaskPosition | None =
        None, **extra_data:
        Any)
```

Use this method to change the `mask position` of a mask sticker. The sticker must belong to a sticker set that was created by the bot. Returns True on success.

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

**sticker:** str

File identifier of the sticker

**mask\_position:** `MaskPosition` | None

A JSON-serialized object with the position where the mask should be placed on faces. Omit the parameter to remove the mask position.

## Usage

### As bot method

```
result: bool = await bot.set_sticker_mask_position(...)
```

## Method as object

Imports:

- `from aiogram.methods.set_sticker_mask_position import SetStickerMaskPosition`
- `alias: from aiogram.methods import SetStickerMaskPosition`

## With specific bot

```
result: bool = await bot(SetStickerMaskPosition(...))
```

## As reply into Webhook in handler

```
return SetStickerMaskPosition(...)
```

## setStickerPositionInSet

Returns: bool

```
class aiogram.methods.set_sticker_position_in_set.SetStickerPositionInSet(*, sticker: str,
                                                                           position: int,
                                                                           **extra_data: Any)
```

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

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

**sticker: str**

File identifier of the sticker

**position: int**

New sticker position in the set, zero-based

## Usage

### As bot method

```
result: bool = await bot.set_sticker_position_in_set(...)
```

## Method as object

Imports:

- `from aiogram.methods.set_sticker_position_in_set import SetStickerPositionInSet`
- `alias: from aiogram.methods import SetStickerPositionInSet`

### With specific bot

```
result: bool = await bot(SetStickerPositionInSet(...))
```

### As reply into Webhook in handler

```
return SetStickerPositionInSet(...)
```

### As shortcut from received object

- `aiogram.types.sticker.Sticker.set_position_in_set()`

### setStickerSetThumbnail

Returns: bool

```
class aiogram.methods.set_sticker_set_thumbnail.SetStickerSetThumbnail(*, name: str, user_id:
    int, thumbnail:
        InputFile | str | None =
        None, **extra_data:
            Any)
```

Use this method to set the thumbnail of a regular or mask sticker set. The format of the thumbnail file must match the format of the stickers in the set. Returns True on success.

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

**name:** str

Sticker set name

**user\_id:** int

User identifier of the sticker set owner

**thumbnail:** `InputFile` | str | None

A **.WEBP** or **.PNG** image with the thumbnail, must be up to 128 kilobytes in size and have a width and height of exactly 100px, or a **.TGS** animation with a thumbnail up to 32 kilobytes in size (see <https://core.telegram.org/stickers#animated-sticker-requirements> <<https://core.telegram.org/stickers#animated-sticker-requirements>>`\_`<https://core.telegram.org/stickers#animated-sticker-requirements> for animated sticker technical requirements), or a **WEBM** video with the thumbnail up to 32 kilobytes in size; see <https://core.telegram.org/stickers#video-sticker-requirements> <<https://core.telegram.org/stickers#video-sticker-requirements>>`\_`<https://core.telegram.org/stickers#video-sticker-requirements> for video sticker technical requirements. Pass a *file\_id* as a String to send a file that already exists on the Telegram servers, pass an HTTP URL as a String for Telegram to get a file from the Internet, or upload a new one using multipart/form-data. *More information on Sending Files* ». Animated and video sticker set thumbnails can't be uploaded via HTTP URL. If omitted, then the thumbnail is dropped and the first sticker is used as the thumbnail.

## Usage

### As bot method

```
result: bool = await bot.set_sticker_set_thumbnail(...)
```

### Method as object

Imports:

- `from aiogram.methods.set_sticker_set_thumbnail import SetStickerSetThumbnail`
- `alias: from aiogram.methods import SetStickerSetThumbnail`

### With specific bot

```
result: bool = await bot(SetStickerSetThumbnail(...))
```

### As reply into Webhook in handler

```
return SetStickerSetThumbnail(...)
```

## setStickerSetTitle

Returns: bool

```
class aiogram.methods.set_sticker_set_title.SetStickerSetTitle(*, name: str, title: str,  
                                                             **extra_data: Any)
```

Use this method to set the title of a created sticker set. Returns True on success.

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

**name: str**

Sticker set name

**title: str**

Sticker set title, 1-64 characters

## Usage

### As bot method

```
result: bool = await bot.set_sticker_set_title(...)
```

## Method as object

Imports:

- `from aiogram.methods.set_sticker_set_title import SetStickerSetTitle`
- `alias: from aiogram.methods import SetStickerSetTitle`

## With specific bot

```
result: bool = await bot(SetStickerSetTitle(...))
```

## As reply into Webhook in handler

```
return SetStickerSetTitle(...)
```

## uploadStickerFile

Returns: File

```
class aiogram.methods.upload_sticker_file.UploadStickerFile(*, user_id: int, sticker: InputFile,
                                                             sticker_format: str, **extra_data:
                                                             Any)
```

Use this method to upload a file with a sticker for later use in the `aiogram.methods.create_new_sticker_set.CreateNewStickerSet` and `aiogram.methods.add_sticker_to_set.AddStickerToSet` methods (the file can be used multiple times). Returns the uploaded `aiogram.types.file.File` on success.

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

**user\_id: int**

User identifier of sticker file owner

**sticker: InputFile**

A file with the sticker in .WEBP, .PNG, .TGS, or .WEBM format. See <https://core.telegram.org/stickers> <<https://core.telegram.org/stickers>>`\_`<https://core.telegram.org/stickers> for technical requirements. *More information on Sending Files »*

**sticker\_format: str**

Format of the sticker, must be one of 'static', 'animated', 'video'

## Usage

### As bot method

```
result: File = await bot.upload_sticker_file(...)
```

## Method as object

Imports:

- `from aiogram.methods.upload_sticker_file import UploadStickerFile`
- `alias: from aiogram.methods import UploadStickerFile`

## With specific bot

```
result: File = await bot(UploadStickerFile(...))
```

## Games

### getGameHighScores

Returns: `List[GameHighScore]`

```
class aiogram.methods.get_game_high_scores.GetGameHighScores(*, user_id: int, chat_id: int | None =  
    None, message_id: int | None =  
    None, inline_message_id: str | None  
    = None, **extra_data: Any)
```

Use this method to get data for high score tables. Will return the score of the specified user and several of their neighbors in a game. Returns an Array of `aiogram.types.game_high_score.GameHighScore` objects.

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

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

**user\_id: int**

Target user id

**chat\_id: int | None**

Required if `inline_message_id` is not specified. Unique identifier for the target chat

**message\_id: int | None**

Required if `inline_message_id` is not specified. Identifier of the sent message

**inline\_message\_id: str | None**

Required if `chat_id` and `message_id` are not specified. Identifier of the inline message

## Usage

### As bot method

```
result: List[GameHighScore] = await bot.get_game_high_scores(...)
```

## Method as object

Imports:

- `from aiogram.methods.get_game_high_scores import GetGameHighScores`
- `alias: from aiogram.methods import GetGameHighScores`

## With specific bot

```
result: List[GameHighScore] = await bot(GetGameHighScores(...))
```

## sendGame

Returns: Message

```
class aiogram.methods.send_game.SendGame(*, chat_id: int, game_short_name: str, message_thread_id: int
    | None = None, disable_notification: bool | None = None,
    protect_content: bool | None =
    sentinel.UNSET_PROTECT_CONTENT,
    reply_to_message_id: int | None = None,
    allow_sending_without_reply: bool | None = None,
    reply_markup: InlineKeyboardMarkup | None = None,
    **extra_data: Any)
```

Use this method to send a game. On success, the sent `aiogram.types.message.Message` is returned.

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

**chat\_id: int**

Unique identifier for the target chat

**game\_short\_name: str**

Short name of the game, serves as the unique identifier for the game. Set up your games via [@BotFather](#).

**message\_thread\_id: int | None**

Unique identifier for the target message thread (topic) of the forum; for forum supergroups only

**disable\_notification: bool | None**

Sends the message [silently](#). Users will receive a notification with no sound.

**protect\_content: bool | None**

Protects the contents of the sent message from forwarding and saving

**reply\_to\_message\_id: int | None**

If the message is a reply, ID of the original message

**allow\_sending\_without\_reply: bool | None**

Pass True if the message should be sent even if the specified replied-to message is not found

**reply\_markup: InlineKeyboardMarkup | None**

A JSON-serialized object for an [inline keyboard](#). If empty, one 'Play game\_title' button will be shown. If not empty, the first button must launch the game.

## Usage

### As bot method

```
result: Message = await bot.send_game(...)
```

### Method as object

Imports:

- `from aiogram.methods.send_game import SendGame`
- `alias: from aiogram.methods import SendGame`

### With specific bot

```
result: Message = await bot(SendGame(...))
```

### As reply into Webhook in handler

```
return SendGame(...)
```

### As shortcut from received object

- `aiogram.types.message.Message.answer_game()`
- `aiogram.types.message.Message.reply_game()`
- `aiogram.types.chat_member_updated.ChatMemberUpdated.answer_game()`
- `aiogram.types.chat_join_request.ChatJoinRequest.answer_game()`
- `aiogram.types.chat_join_request.ChatJoinRequest.answer_game_pm()`

## setGameScore

Returns: `Union[Message, bool]`

```
class aiogram.methods.set_game_score.SetGameScore(*, user_id: int, score: int, force: bool | None = None, disable_edit_message: bool | None = None, chat_id: int | None = None, message_id: int | None = None, inline_message_id: str | None = None, **extra_data: Any)
```

Use this method to set the score of the specified user in a game message. On success, if the message is not an inline message, the `aiogram.types.message.Message` is returned, otherwise `True` is returned. Returns an error, if the new score is not greater than the user's current score in the chat and `force` is `False`.

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



**user\_id: int**

User identifier

**score: int**

New score, must be non-negative

**force: bool | None**

Pass True if the high score is allowed to decrease. This can be useful when fixing mistakes or banning cheaters

**disable\_edit\_message: bool | None**

Pass True if the game message should not be automatically edited to include the current scoreboard

**chat\_id: int | None**Required if *inline\_message\_id* is not specified. Unique identifier for the target chat**message\_id: int | None**Required if *inline\_message\_id* is not specified. Identifier of the sent message**inline\_message\_id: str | None**Required if *chat\_id* and *message\_id* are not specified. Identifier of the inline message

## Usage

### As bot method

```
result: Union[Message, bool] = await bot.set_game_score(...)
```

### Method as object

Imports:

- `from aiogram.methods.set_game_score import SetGameScore`
- alias: `from aiogram.methods import SetGameScore`

### With specific bot

```
result: Union[Message, bool] = await bot(SetGameScore(...))
```

### As reply into Webhook in handler

```
return SetGameScore(...)
```

## Getting updates

### deleteWebhook

Returns: bool

```
class aiogram.methods.delete_webhook.DeleteWebhook(*, drop_pending_updates: bool | None = None,
                                                    **extra_data: Any)
```

Use this method to remove webhook integration if you decide to switch back to `aiogram.methods.get_updates.GetUpdates`. Returns True on success.

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

**drop\_pending\_updates:** bool | None

Pass True to drop all pending updates

## Usage

### As bot method

```
result: bool = await bot.delete_webhook(...)
```

### Method as object

Imports:

- `from aiogram.methods.delete_webhook import DeleteWebhook`
- `alias: from aiogram.methods import DeleteWebhook`

### With specific bot

```
result: bool = await bot(DeleteWebhook(...))
```

### As reply into Webhook in handler

```
return DeleteWebhook(...)
```

## getUpdates

Returns: List[Update]

```
class aiogram.methods.get_updates.GetUpdates(*, offset: int | None = None, limit: int | None = None,
                                              timeout: int | None = None, allowed_updates: List[str] |
                                              None = None, **extra_data: Any)
```

Use this method to receive incoming updates using long polling ([wiki](#)). Returns an Array of `aiogram.types.update.Update` objects.

### 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>

**offset: int | None**

Identifier of the first update to be returned. Must be greater by one than the highest among the identifiers of previously received updates. By default, updates starting with the earliest unconfirmed update are returned. An update is considered confirmed as soon as `aiogram.methods.get_updates.GetUpdates` is called with an *offset* higher than its *update\_id*. The negative offset can be specified to retrieve updates starting from *-offset* update from the end of the updates queue. All previous updates will be forgotten.

**limit: int | None**

Limits the number of updates to be retrieved. Values between 1-100 are accepted. Defaults to 100.

**timeout: int | None**

Timeout in seconds for long polling. Defaults to 0, i.e. usual short polling. Should be positive, short polling should be used for testing purposes only.

**allowed\_updates: List[str] | None**

A JSON-serialized 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 `aiogram.types.update.Update` for a complete list of available update types. Specify an empty list to receive all update types except *chat\_member* (default). If not specified, the previous setting will be used.

## Usage

### As bot method

```
result: List[Update] = await bot.get_updates(...)
```

### Method as object

Imports:

- `from aiogram.methods.get_updates import GetUpdates`
- `alias: from aiogram.methods import GetUpdates`

### With specific bot

```
result: List[Update] = await bot(GetUpdates(...))
```

## getWebhookInfo

Returns: `WebhookInfo`

**class** `aiogram.methods.get_webhook_info.GetWebhookInfo`(\*\**extra\_data*: Any)

Use this method to get current webhook status. Requires no parameters. On success, returns a `aiogram.types.webhook_info.WebhookInfo` object. If the bot is using `aiogram.methods.get_updates.GetUpdates`, will return an object with the `url` field empty.

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

## Usage

### As bot method

```
result: WebhookInfo = await bot.get_webhook_info(...)
```

### Method as object

Imports:

- `from aiogram.methods.get_webhook_info import GetWebhookInfo`
- `alias: from aiogram.methods import GetWebhookInfo`

### With specific bot

```
result: WebhookInfo = await bot(GetWebhookInfo(...))
```

## setWebhook

Returns: `bool`

**class** `aiogram.methods.set_webhook.SetWebhook`(\*, *url*: str, *certificate*: `InputFile` | `None` = `None`, *ip\_address*: str | `None` = `None`, *max\_connections*: int | `None` = `None`, *allowed\_updates*: List[str] | `None` = `None`, *drop\_pending\_updates*: bool | `None` = `None`, *secret\_token*: str | `None` = `None`, \*\**extra\_data*: Any)

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 `aiogram.types.update.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 was set by you, you can specify secret data in the parameter *secret\_token*. If specified, the request will contain a header 'X-Telegram-Bot-Api-Secret-Token' with the secret token as content.

### Notes

1. You will not be able to receive updates using `aiogram.methods.get_updates.GetUpdates` for as long as an outgoing webhook is set up.

2. To use a self-signed certificate, you need to upload your [public key certificate](#) using *certificate* parameter. Please upload as `InputFile`, sending a `String` will not work.

3. Ports currently supported *for webhooks*: **443, 80, 88, 8443**. If you're having any trouble setting up webhooks, please check out this [amazing guide to webhooks](#).

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

**url:** `str`

HTTPS URL to send updates to. Use an empty string to remove webhook integration

**certificate:** `InputFile` | `None`

Upload your public key certificate so that the root certificate in use can be checked. See our [self-signed guide](#) for details.

**ip\_address:** `str` | `None`

The fixed IP address which will be used to send webhook requests instead of the IP address resolved through DNS

**max\_connections:** `int` | `None`

The 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:** `List[str]` | `None`

A JSON-serialized 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 [aiogram.types.update.Update](#) for a complete list of available update types. Specify an empty list to receive all update types except *chat\_member* (default). If not specified, the previous setting will be used.

**drop\_pending\_updates:** `bool` | `None`

Pass `True` to drop all pending updates

**secret\_token:** `str` | `None`

A secret token to be sent in a header `'X-Telegram-Bot-API-Secret-Token'` in every webhook request, 1-256 characters. Only characters `A-Z`, `a-z`, `0-9`, `_` and `-` are allowed. The header is useful to ensure that the request comes from a webhook set by you.

## Usage

### As bot method

```
result: bool = await bot.set_webhook(...)
```

### Method as object

Imports:

- `from aiogram.methods.set_webhook import SetWebhook`
- `alias: from aiogram.methods import SetWebhook`

### With specific bot

```
result: bool = await bot(SetWebhook(...))
```

### As reply into Webhook in handler

```
return SetWebhook(...)
```

## Telegram Passport

### setPassportDataErrors

Returns: bool

```
class aiogram.methods.set_passport_data_errors.SetPassportDataErrors(*, user_id: int, errors:
    List[PassportElementErrorDataField
    | PassportElementErrorFrontSide |
    PassportElementErrorReverseSide |
    PassportElementErrorSelfie |
    PassportElementErrorFile
    | PassportElementErrorFiles |
    PassportElementErrorTranslationFile |
    PassportElementErrorTranslationFiles |
    PassportElementErrorUnspecified], **extra_data:
    Any)
```

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>

**user\_id: int**

User identifier

**errors: List[*PassportElementErrorDataField* | *PassportElementErrorFrontSide* | *PassportElementErrorReverseSide* | *PassportElementErrorSelfie* | *PassportElementErrorFile* | *PassportElementErrorFiles* | *PassportElementErrorTranslationFile* | *PassportElementErrorTranslationFiles* | *PassportElementErrorUnspecified*]**

A JSON-serialized array describing the errors

## Usage

### As bot method

```
result: bool = await bot.set_passport_data_errors(...)
```

### Method as object

Imports:

- `from aiogram.methods.set_passport_data_errors import SetPassportDataErrors`
- `alias: from aiogram.methods import SetPassportDataErrors`

### With specific bot

```
result: bool = await bot(SetPassportDataErrors(...))
```

### As reply into Webhook in handler

```
return SetPassportDataErrors(...)
```

## Updating messages

### deleteMessage

Returns: bool

```
class aiogram.methods.delete_message.DeleteMessage(*, chat_id: int | str, message_id: int,
                                                    **extra_data: Any)
```

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.
- Service messages about a supergroup, channel, or forum topic creation can't be deleted.
- A dice message in a private chat can only be deleted if it was sent more than 24 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.

Returns True on success.

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

**chat\_id:** `int | str`

Unique identifier for the target chat or username of the target channel (in the format @channelusername)

**message\_id:** `int`

Identifier of the message to delete

## Usage

### As bot method

```
result: bool = await bot.delete_message(...)
```

### Method as object

Imports:

- `from aiogram.methods.delete_message import DeleteMessage`
- `alias: from aiogram.methods import DeleteMessage`

### With specific bot

```
result: bool = await bot(DeleteMessage(...))
```

### As reply into Webhook in handler

```
return DeleteMessage(...)
```

### As shortcut from received object

- `aiogram.types.message.Message.delete()`
- `aiogram.types.chat.Chat.delete_message()`

## editMessageCaption

Returns: `Union[Message, bool]`

```
class aiogram.methods.edit_message_caption.EditMessageCaption(*, chat_id: int | str | None = None,
    message_id: int | None = None,
    inline_message_id: str | None =
    None, caption: str | None = None,
    parse_mode: str | None =
    sentinel.UNSET_PARSE_MODE,
    caption_entities:
    List[MessageEntity] | None =
    None, reply_markup:
    InlineKeyboardMarkup | None =
    None, **extra_data: Any)
```



Use this method to edit captions of messages. On success, if the edited message is not an inline message, the edited `aiogram.types.message.Message` is returned, otherwise `True` is returned.

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

**chat\_id:** `int | str | None`

Required if `inline_message_id` is not specified. Unique identifier for the target chat or username of the target channel (in the format `@channelusername`)

**message\_id:** `int | None`

Required if `inline_message_id` is not specified. Identifier of the message to edit

**inline\_message\_id:** `str | None`

Required if `chat_id` and `message_id` are not specified. Identifier of the inline message

**caption:** `str | None`

New caption of the message, 0-1024 characters after entities parsing

**parse\_mode:** `str | None`

Mode for parsing entities in the message caption. See [formatting options](#) for more details.

**caption\_entities:** `List[MessageEntity] | None`

A JSON-serialized list of special entities that appear in the caption, which can be specified instead of `parse_mode`

**reply\_markup:** `InlineKeyboardMarkup | None`

A JSON-serialized object for an [inline keyboard](#).

## Usage

### As bot method

```
result: Union[Message, bool] = await bot.edit_message_caption(...)
```

### Method as object

Imports:

- `from aiogram.methods.edit_message_caption import EditMessageCaption`
- `alias: from aiogram.methods import EditMessageCaption`

### With specific bot

```
result: Union[Message, bool] = await bot(EditMessageCaption(...))
```

### As reply into Webhook in handler

```
return EditMessageCaption(...)
```

### As shortcut from received object

- `aiogram.types.message.Message.edit_caption()`

### `editMessageLiveLocation`

Returns: Union[Message, bool]

```
class aiogram.methods.edit_message_live_location.EditMessageLiveLocation(*, latitude: float,
                                                                           longitude: float,
                                                                           chat_id: int | str |
                                                                           None = None,
                                                                           message_id: int |
                                                                           None = None,
                                                                           inline_message_id:
                                                                           str | None = None,
                                                                           horizontal_accuracy:
                                                                           float | None = None,
                                                                           heading: int | None
                                                                           = None, proximity_
                                                                           alert_radius: int
                                                                           | None = None,
                                                                           reply_markup: In-
                                                                           lineKeyboardMarkup
                                                                           | None = None,
                                                                           **extra_data: Any)
```

Use this method to edit live location messages. A location can be edited until its *live\_period* expires or editing is explicitly disabled by a call to `aiogram.methods.stop_message_live_location.StopMessageLiveLocation`. On success, if the edited message is not an inline message, the edited `aiogram.types.message.Message` is returned, otherwise `True` is returned.

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

**latitude: float**

Latitude of new location

**longitude: float**

Longitude of new location

**chat\_id: int | str | None**

Required if *inline\_message\_id* is not specified. Unique identifier for the target chat or username of the target channel (in the format @channelusername)

**message\_id: int | None**

Required if *inline\_message\_id* is not specified. Identifier of the message to edit

**inline\_message\_id: str | None**

Required if *chat\_id* and *message\_id* are not specified. Identifier of the inline message

**horizontal\_accuracy:** `float` | `None`

The radius of uncertainty for the location, measured in meters; 0-1500

**heading:** `int` | `None`

Direction in which the user is moving, in degrees. Must be between 1 and 360 if specified.

**proximity\_alert\_radius:** `int` | `None`

The maximum distance for proximity alerts about approaching another chat member, in meters. Must be between 1 and 100000 if specified.

**reply\_markup:** `InlineKeyboardMarkup` | `None`

A JSON-serialized object for a new `inline keyboard`.

## Usage

### As bot method

```
result: Union[Message, bool] = await bot.edit_message_live_location(...)
```

### Method as object

Imports:

- `from aiogram.methods.edit_message_live_location import EditMessageLiveLocation`
- `alias: from aiogram.methods import EditMessageLiveLocation`

### With specific bot

```
result: Union[Message, bool] = await bot(EditMessageLiveLocation(...))
```

### As reply into Webhook in handler

```
return EditMessageLiveLocation(...)
```

### As shortcut from received object

- `aiogram.types.message.Message.edit_live_location()`

## editMessageMedia

Returns: Union[Message, bool]

```
class aiogram.methods.edit_message_media.EditMessageMedia(*, media: InputMediaAnimation |  
    InputMediaDocument |  
    InputMediaAudio | InputMediaPhoto |  
    InputMediaVideo, chat_id: int | str |  
    None = None, message_id: int | None =  
    None, inline_message_id: str | None =  
    None, reply_markup: InlineKeyboardMarkup | None = None,  
    **extra_data: Any)
```

Use this method to edit animation, audio, document, photo, or video messages. If a message is part of a message album, then it can be edited only to an audio for audio albums, only to a document for document albums and to a photo or a video otherwise. When an inline message is edited, a new file can't be uploaded; use a previously uploaded file via its `file_id` or specify a URL. On success, if the edited message is not an inline message, the edited `aiogram.types.message.Message` is returned, otherwise `True` is returned.

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

**media:** `InputMediaAnimation` | `InputMediaDocument` | `InputMediaAudio` | `InputMediaPhoto` | `InputMediaVideo`

A JSON-serialized object for a new media content of the message

**chat\_id:** `int` | `str` | `None`

Required if `inline_message_id` is not specified. Unique identifier for the target chat or username of the target channel (in the format `@channelusername`)

**message\_id:** `int` | `None`

Required if `inline_message_id` is not specified. Identifier of the message to edit

**inline\_message\_id:** `str` | `None`

Required if `chat_id` and `message_id` are not specified. Identifier of the inline message

**reply\_markup:** `InlineKeyboardMarkup` | `None`

A JSON-serialized object for a new inline keyboard.

## Usage

### As bot method

```
result: Union[Message, bool] = await bot.edit_message_media(...)
```

## Method as object

Imports:

- `from aiogram.methods.edit_message_media import EditMessageMedia`
- `alias: from aiogram.methods import EditMessageMedia`

## With specific bot

```
result: Union[Message, bool] = await bot(EditMessageMedia(...))
```

## As reply into Webhook in handler

```
return EditMessageMedia(...)
```

## As shortcut from received object

- `aiogram.types.message.Message.edit_media()`

## editMessageReplyMarkup

Returns: Union[Message, bool]

```
class aiogram.methods.edit_message_reply_markup.EditMessageReplyMarkup(*, chat_id: int | str |
    None = None,
    message_id: int | None
    = None,
    inline_message_id: str |
    None = None,
    reply_markup:
    InlineKeyboardMarkup
    | None = None,
    **extra_data: Any)
```

Use this method to edit only the reply markup of messages. On success, if the edited message is not an inline message, the edited `aiogram.types.message.Message` is returned, otherwise True is returned.

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

**chat\_id:** int | str | None

Required if `inline_message_id` is not specified. Unique identifier for the target chat or username of the target channel (in the format @channelusername)

**message\_id:** int | None

Required if `inline_message_id` is not specified. Identifier of the message to edit

**inline\_message\_id:** str | None

Required if `chat_id` and `message_id` are not specified. Identifier of the inline message

**reply\_markup:** `InlineKeyboardMarkup` | None

A JSON-serialized object for an inline keyboard.

## Usage

### As bot method

```
result: Union[Message, bool] = await bot.edit_message_reply_markup(...)
```

### Method as object

Imports:

- `from aiogram.methods.edit_message_reply_markup import EditMessageReplyMarkup`
- `alias: from aiogram.methods import EditMessageReplyMarkup`

### With specific bot

```
result: Union[Message, bool] = await bot(EditMessageReplyMarkup(...))
```

### As reply into Webhook in handler

```
return EditMessageReplyMarkup(...)
```

### As shortcut from received object

- `aiogram.types.message.Message.edit_reply_markup()`
- `aiogram.types.message.Message.delete_reply_markup()`

## editMessageText

Returns: Union[Message, bool]

```
class aiogram.methods.edit_message_text.EditMessageText(*, text: str, chat_id: int | str | None = None,
                                                         message_id: int | None = None,
                                                         inline_message_id: str | None = None,
                                                         parse_mode: str | None =
                                                         sentinel.UNSET_PARSE_MODE, entities:
                                                         List[MessageEntity] | None = None,
                                                         disable_web_page_preview: bool | None =
                                                         sentinel.UNSET_DISABLE_WEB_PAGE_PREVIEW,
                                                         reply_markup: InlineKeyboardMarkup |
                                                         None = None, **extra_data: Any)
```

Use this method to edit text and `game` messages. On success, if the edited message is not an inline message, the edited `aiogram.types.message.Message` is returned, otherwise `True` is returned.

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

**text:** `str`

New text of the message, 1-4096 characters after entities parsing

**chat\_id:** `int | str | None`

Required if *inline\_message\_id* is not specified. Unique identifier for the target chat or username of the target channel (in the format @channelusername)

**message\_id:** `int | None`

Required if *inline\_message\_id* is not specified. Identifier of the message to edit

**inline\_message\_id:** `str | None`

Required if *chat\_id* and *message\_id* are not specified. Identifier of the inline message

**parse\_mode:** `str | None`

Mode for parsing entities in the message text. See [formatting options](#) for more details.

**entities:** `List[MessageEntity] | None`

A JSON-serialized list of special entities that appear in message text, which can be specified instead of *parse\_mode*

**disable\_web\_page\_preview:** `bool | None`

Disables link previews for links in this message

**reply\_markup:** `InlineKeyboardMarkup | None`

A JSON-serialized object for an [inline keyboard](#).

## Usage

### As bot method

```
result: Union[Message, bool] = await bot.edit_message_text(...)
```

### Method as object

Imports:

- `from aiogram.methods.edit_message_text import EditMessageText`
- `alias: from aiogram.methods import EditMessageText`

### With specific bot

```
result: Union[Message, bool] = await bot(EditMessageText(...))
```

### As reply into Webhook in handler

```
return EditMessageText(...)
```

### As shortcut from received object

- `aiogram.types.message.Message.edit_text()`

### stopMessageLiveLocation

Returns: Union[Message, bool]

```
class aiogram.methods.stop_message_live_location.StopMessageLiveLocation(*, chat_id: int | str |  
    None = None,  
    message_id: int |  
    None = None,  
    inline_message_id:  
    str | None = None,  
    reply_markup: In-  
    lineKeyboardMarkup  
    | None = None,  
    **extra_data: Any)
```

Use this method to stop updating a live location message before *live\_period* expires. On success, if the message is not an inline message, the edited `aiogram.types.message.Message` is returned, otherwise True is returned.

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

**chat\_id:** int | str | None

Required if *inline\_message\_id* is not specified. Unique identifier for the target chat or username of the target channel (in the format @channelusername)

**message\_id:** int | None

Required if *inline\_message\_id* is not specified. Identifier of the message with live location to stop

**inline\_message\_id:** str | None

Required if *chat\_id* and *message\_id* are not specified. Identifier of the inline message

**reply\_markup:** `InlineKeyboardMarkup` | None

A JSON-serialized object for a new inline keyboard.

### Usage

#### As bot method

```
result: Union[Message, bool] = await bot.stop_message_live_location(...)
```



## Method as object

Imports:

- `from aiogram.methods.stop_message_live_location import StopMessageLiveLocation`
- `alias: from aiogram.methods import StopMessageLiveLocation`

## With specific bot

```
result: Union[Message, bool] = await bot(StopMessageLiveLocation(...))
```

## As reply into Webhook in handler

```
return StopMessageLiveLocation(...)
```

## As shortcut from received object

- `aiogram.types.message.Message.stop_live_location()`

## stopPoll

Returns: Poll

```
class aiogram.methods.stop_poll.StopPoll(*, chat_id: int | str, message_id: int, reply_markup:
                                         InlineKeyboardMarkup | None = None, **extra_data: Any)
```

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

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

**chat\_id: int | str**

Unique identifier for the target chat or username of the target channel (in the format @channelusername)

**message\_id: int**

Identifier of the original message with the poll

**reply\_markup: InlineKeyboardMarkup | None**

A JSON-serialized object for a new message inline keyboard.

## Usage

### As bot method

```
result: Poll = await bot.stop_poll(...)
```

## Method as object

Imports:

- `from aiogram.methods.stop_poll import StopPoll`
- `alias: from aiogram.methods import StopPoll`

## With specific bot

```
result: Poll = await bot(StopPoll(...))
```

## As reply into Webhook in handler

```
return StopPoll(...)
```

## Inline mode

### answerInlineQuery

Returns: bool

```
class aiogram.methods.answer_inline_query.AnswerInlineQuery(*, inline_query_id: str, results:
    List[InlineQueryResultCachedAudio |
    InlineQueryResultCachedDocument |
    InlineQueryResultCachedGif |
    InlineQueryResultCachedMpeg4Gif |
    InlineQueryResultCachedPhoto |
    InlineQueryResultCachedSticker |
    InlineQueryResultCachedVideo |
    InlineQueryResultCachedVoice |
    InlineQueryResultArticle |
    InlineQueryResultAudio |
    InlineQueryResultContact |
    InlineQueryResultGame |
    InlineQueryResultDocument |
    InlineQueryResultGif |
    InlineQueryResultLocation |
    InlineQueryResultMpeg4Gif |
    InlineQueryResultPhoto |
    InlineQueryResultVenue |
    InlineQueryResultVideo |
    InlineQueryResultVoice], cache_time:
    int | None = None, is_personal: bool |
    None = None, next_offset: str | None
    = None, button:
    InlineQueryResultsButton | None =
    None, switch_pm_parameter: str |
    None = None, switch_pm_text: str |
    None = None, **extra_data: Any)
```

Use this method to send answers to an inline query. On success, True is returned.

No more than **50** results per query are allowed.

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

**inline\_query\_id:** `str`

Unique identifier for the answered query

**results:** `List[InlineQueryResultCachedAudio | InlineQueryResultCachedDocument | InlineQueryResultCachedGif | InlineQueryResultCachedMpeg4Gif | InlineQueryResultCachedPhoto | InlineQueryResultCachedSticker | InlineQueryResultCachedVideo | InlineQueryResultCachedVoice | InlineQueryResultArticle | InlineQueryResultAudio | InlineQueryResultContact | InlineQueryResultGame | InlineQueryResultDocument | InlineQueryResultGif | InlineQueryResultLocation | InlineQueryResultMpeg4Gif | InlineQueryResultPhoto | InlineQueryResultVenue | InlineQueryResultVideo | InlineQueryResultVoice]`

A JSON-serialized array of results for the inline query

**cache\_time:** `int | None`

The maximum amount of time in seconds that the result of the inline query may be cached on the server. Defaults to 300.

**is\_personal:** `bool | None`

Pass True if results may be cached on the server side only for the user that sent the query. By default, results may be returned to any user who sends the same query.

**next\_offset:** `str | None`

Pass the offset that a client should send in the next query with the same text to receive more results. Pass an empty string if there are no more results or if you don't support pagination. Offset length can't exceed 64 bytes.

**button:** `InlineQueryResultsButton | None`

A JSON-serialized object describing a button to be shown above inline query results

**switch\_pm\_parameter:** `str | None`

Deep-linking parameter for the /start message sent to the bot when user presses the switch button. 1-64 characters, only A-Z, a-z, 0-9, \_ and - are allowed.

Deprecated since version API:6.7: <https://core.telegram.org/bots/api-changelog#april-21-2023>

**switch\_pm\_text:** `str | None`

If passed, clients will display a button with specified text that switches the user to a private chat with the bot and sends the bot a start message with the parameter *switch\_pm\_parameter*

Deprecated since version API:6.7: <https://core.telegram.org/bots/api-changelog#april-21-2023>

## Usage

### As bot method

```
result: bool = await bot.answer_inline_query(...)
```

### Method as object

Imports:

- `from aiogram.methods.answer_inline_query import AnswerInlineQuery`
- `alias: from aiogram.methods import AnswerInlineQuery`

### With specific bot

```
result: bool = await bot(AnswerInlineQuery(...))
```

### As reply into Webhook in handler

```
return AnswerInlineQuery(...)
```

### As shortcut from received object

- `aiogram.types.inline_query.InlineQuery.answer()`

## answerWebAppQuery

Returns: `SentWebAppMessage`

```

class aiogram.methods.answer_web_app_query.AnswerWebAppQuery(*, web_app_query_id: str, result:
    InlineQueryResultCachedAudio |
    InlineQueryResultCachedDocument
    | InlineQueryResultCachedGif |
    InlineQueryResultCachedMpeg4Gif
    | InlineQueryResultCachedPhoto |
    InlineQueryResultCachedSticker |
    InlineQueryResultCachedVideo |
    InlineQueryResultCachedVoice |
    InlineQueryResultArticle |
    InlineQueryResultAudio |
    InlineQueryResultContact |
    InlineQueryResultGame |
    InlineQueryResultDocument |
    InlineQueryResultGif |
    InlineQueryResultLocation |
    InlineQueryResultMpeg4Gif |
    InlineQueryResultPhoto |
    InlineQueryResultVenue |
    InlineQueryResultVideo |
    InlineQueryResultVoice,
    **extra_data: Any)

```

Use this method to set the result of an interaction with a [Web App](#) and send a corresponding message on behalf of the user to the chat from which the query originated. On success, a `aiogram.types.sent_web_app_message.SentWebAppMessage` object is returned.

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

**web\_app\_query\_id:** `str`

Unique identifier for the query to be answered

**result:** `InlineQueryResultCachedAudio | InlineQueryResultCachedDocument |`  
`InlineQueryResultCachedGif | InlineQueryResultCachedMpeg4Gif |`  
`InlineQueryResultCachedPhoto | InlineQueryResultCachedSticker |`  
`InlineQueryResultCachedVideo | InlineQueryResultCachedVoice |`  
`InlineQueryResultArticle | InlineQueryResultAudio | InlineQueryResultContact |`  
`InlineQueryResultGame | InlineQueryResultDocument | InlineQueryResultGif |`  
`InlineQueryResultLocation | InlineQueryResultMpeg4Gif | InlineQueryResultPhoto |`  
`InlineQueryResultVenue | InlineQueryResultVideo | InlineQueryResultVoice`

A JSON-serialized object describing the message to be sent

## Usage

### As bot method

```
result: SentWebAppMessage = await bot.answer_web_app_query(...)
```

### Method as object

Imports:

- `from aiogram.methods.answer_web_app_query import AnswerWebAppQuery`
- `alias: from aiogram.methods import AnswerWebAppQuery`

### With specific bot

```
result: SentWebAppMessage = await bot(AnswerWebAppQuery(...))
```

### As reply into Webhook in handler

```
return AnswerWebAppQuery(...)
```

## 2.3.5 Enums

Here is list of all available enums:

### BotCommandScopeType

```
class aiogram.enums.bot_command_scope_type.BotCommandScopeType(value)
```

This object represents the scope to which bot commands are applied.

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

```
DEFAULT = 'default'
```

```
ALL_PRIVATE_CHATS = 'all_private_chats'
```

```
ALL_GROUP_CHATS = 'all_group_chats'
```

```
ALL_CHAT_ADMINISTRATORS = 'all_chat_administrators'
```

```
CHAT = 'chat'
```

```
CHAT_ADMINISTRATORS = 'chat_administrators'
```

```
CHAT_MEMBER = 'chat_member'
```

## ChatAction

**class** aiogram.enums.chat\_action.ChatAction(value)

This object represents bot actions.

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,
- choose\_sticker for stickers,
- find\_location for location data,
- record\_video\_note or upload\_video\_note for video notes.

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

```
TYPING = 'typing'

UPLOAD_PHOTO = 'upload_photo'

RECORD_VIDEO = 'record_video'

UPLOAD_VIDEO = 'upload_video'

RECORD_VOICE = 'record_voice'

UPLOAD_VOICE = 'upload_voice'

UPLOAD_DOCUMENT = 'upload_document'

CHOOSE_STICKER = 'choose_sticker'

FIND_LOCATION = 'find_location'

RECORD_VIDEO_NOTE = 'record_video_note'

UPLOAD_VIDEO_NOTE = 'upload_video_note'
```

## ChatMemberStatus

**class** aiogram.enums.chat\_member\_status.ChatMemberStatus(value)

This object represents chat member status.

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

```
CREATOR = 'creator'

ADMINISTRATOR = 'administrator'

MEMBER = 'member'

RESTRICTED = 'restricted'
```

```
LEFT = 'left'
KICKED = 'kicked'
```

## ChatType

```
class aiogram.enums.chat_type.ChatType(value)
    This object represents a chat type
    Source: https://core.telegram.org/bots/api#chat
    SENDER = 'sender'
    PRIVATE = 'private'
    GROUP = 'group'
    SUPERGROUP = 'supergroup'
    CHANNEL = 'channel'
```

## ContentType

```
class aiogram.enums.content_type.ContentType(value)
    This object represents a type of content in message
    UNKNOWN = 'unknown'
    ANY = 'any'
    TEXT = 'text'
    ANIMATION = 'animation'
    AUDIO = 'audio'
    DOCUMENT = 'document'
    PHOTO = 'photo'
    STICKER = 'sticker'
    STORY = 'story'
    VIDEO = 'video'
    VIDEO_NOTE = 'video_note'
    VOICE = 'voice'
    HAS_MEDIA_SPOILER = 'has_media_spoiler'
    CONTACT = 'contact'
    DICE = 'dice'
    GAME = 'game'
```



```
POLL = 'poll'
VENUE = 'venue'
LOCATION = 'location'
NEW_CHAT_MEMBERS = 'new_chat_members'
LEFT_CHAT_MEMBER = 'left_chat_member'
NEW_CHAT_TITLE = 'new_chat_title'
NEW_CHAT_PHOTO = 'new_chat_photo'
DELETE_CHAT_PHOTO = 'delete_chat_photo'
GROUP_CHAT_CREATED = 'group_chat_created'
SUPERGROUP_CHAT_CREATED = 'supergroup_chat_created'
CHANNEL_CHAT_CREATED = 'channel_chat_created'
MESSAGE_AUTO_DELETE_TIMER_CHANGED = 'message_auto_delete_timer_changed'
MIGRATE_TO_CHAT_ID = 'migrate_to_chat_id'
MIGRATE_FROM_CHAT_ID = 'migrate_from_chat_id'
PINNED_MESSAGE = 'pinned_message'
INVOICE = 'invoice'
SUCCESSFUL_PAYMENT = 'successful_payment'
USER_SHARED = 'user_shared'
CHAT_SHARED = 'chat_shared'
CONNECTED_WEBSITE = 'connected_website'
WRITE_ACCESS_ALLOWED = 'write_access_allowed'
PASSPORT_DATA = 'passport_data'
PROXIMITY_ALERT_TRIGGERED = 'proximity_alert_triggered'
FORUM_TOPIC_CREATED = 'forum_topic_created'
FORUM_TOPIC_EDITED = 'forum_topic_edited'
FORUM_TOPIC_CLOSED = 'forum_topic_closed'
FORUM_TOPIC_REOPENED = 'forum_topic_reopened'
GENERAL_FORUM_TOPIC_HIDDEN = 'general_forum_topic_hidden'
GENERAL_FORUM_TOPIC_UNHIDDEN = 'general_forum_topic_unhidden'
VIDEO_CHAT_SCHEDULED = 'video_chat_scheduled'
VIDEO_CHAT_STARTED = 'video_chat_started'
```

```
VIDEO_CHAT_ENDED = 'video_chat_ended'
```

```
VIDEO_CHAT_PARTICIPANTS_INVITED = 'video_chat_participants_invited'
```

```
WEB_APP_DATA = 'web_app_data'
```

## Currency

```
class aiogram.enums.currency.Currency(value)
```

Currencies supported by Telegram Bot API

Source: <https://core.telegram.org/bots/payments#supported-currencies>

```
AED = 'AED'
```

```
AFN = 'AFN'
```

```
ALL = 'ALL'
```

```
AMD = 'AMD'
```

```
ARS = 'ARS'
```

```
AUD = 'AUD'
```

```
AZN = 'AZN'
```

```
BAM = 'BAM'
```

```
BDT = 'BDT'
```

```
BGN = 'BGN'
```

```
BND = 'BND'
```

```
BOB = 'BOB'
```

```
BRL = 'BRL'
```

```
BYN = 'BYN'
```

```
CAD = 'CAD'
```

```
CHF = 'CHF'
```

```
CLP = 'CLP'
```

```
CNY = 'CNY'
```

```
COP = 'COP'
```

```
CRC = 'CRC'
```

```
CZK = 'CZK'
```

```
DKK = 'DKK'
```

```
DOP = 'DOP'
```

DZD = 'DZD'  
EGP = 'EGP'  
ETB = 'ETB'  
EUR = 'EUR'  
GBP = 'GBP'  
GEL = 'GEL'  
GTQ = 'GTQ'  
HKD = 'HKD'  
HNL = 'HNL'  
HRK = 'HRK'  
HUF = 'HUF'  
IDR = 'IDR'  
ILS = 'ILS'  
INR = 'INR'  
ISK = 'ISK'  
JMD = 'JMD'  
JPY = 'JPY'  
KES = 'KES'  
KGS = 'KGS'  
KRW = 'KRW'  
KZT = 'KZT'  
LBP = 'LBP'  
LKR = 'LKR'  
MAD = 'MAD'  
MDL = 'MDL'  
MNT = 'MNT'  
MUR = 'MUR'  
MVR = 'MVR'  
MXN = 'MXN'  
MYR = 'MYR'  
MZN = 'MZN'

NGN = 'NGN'  
NIO = 'NIO'  
NOK = 'NOK'  
NPR = 'NPR'  
NZD = 'NZD'  
PAB = 'PAB'  
PEN = 'PEN'  
PHP = 'PHP'  
PKR = 'PKR'  
PLN = 'PLN'  
PYG = 'PYG'  
QAR = 'QAR'  
RON = 'RON'  
RSD = 'RSD'  
RUB = 'RUB'  
SAR = 'SAR'  
SEK = 'SEK'  
SGD = 'SGD'  
THB = 'THB'  
TJS = 'TJS'  
TRY = 'TRY'  
TTD = 'TTD'  
TWD = 'TWD'  
TZS = 'TZS'  
UAH = 'UAH'  
UGX = 'UGX'  
USD = 'USD'  
UYU = 'UYU'  
UZS = 'UZS'  
VND = 'VND'  
YER = 'YER'  
ZAR = 'ZAR'

## DiceEmoji

```
class aiogram.enums.dice_emoji.DiceEmoji(value)
```

Emoji on which the dice throw animation is based

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

DICE = ''

DART = ''

BASKETBALL = ''

FOOTBALL = ''

SLOT\_MACHINE = ''

BOWLING = ''

## EncryptedPassportElement

```
class aiogram.enums.encrypted_passport_element.EncryptedPassportElement(value)
```

This object represents type of encrypted passport element.

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

PERSONAL\_DETAILS = 'personal\_details'

PASSPORT = 'passport'

DRIVER\_LICENSE = 'driver\_license'

IDENTITY\_CARD = 'identity\_card'

INTERNAL\_PASSPORT = 'internal\_passport'

ADDRESS = 'address'

UTILITY\_BILL = 'utility\_bill'

BANK\_STATEMENT = 'bank\_statement'

RENTAL\_AGREEMENT = 'rental\_agreement'

PASSPORT\_REGISTRATION = 'passport\_registration'

TEMPORARY\_REGISTRATION = 'temporary\_registration'

PHONE\_NUMBER = 'phone\_number'

EMAIL = 'email'

### InlineQueryResultType

**class** aiogram.enums.inline\_query\_result\_type.**InlineQueryResultType**(*value*)

Type of inline query result

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

**AUDIO** = 'audio'

**DOCUMENT** = 'document'

**GIF** = 'gif'

**MPEG4\_GIF** = 'mpeg4\_gif'

**PHOTO** = 'photo'

**STICKER** = 'sticker'

**VIDEO** = 'video'

**VOICE** = 'voice'

**ARTICLE** = 'article'

**CONTACT** = 'contact'

**GAME** = 'game'

**LOCATION** = 'location'

**VENUE** = 'venue'

### InputMediaType

**class** aiogram.enums.input\_media\_type.**InputMediaType**(*value*)

This object represents input media type

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

**ANIMATION** = 'animation'

**AUDIO** = 'audio'

**DOCUMENT** = 'document'

**PHOTO** = 'photo'

**VIDEO** = 'video'

### MaskPositionPoint

**class** aiogram.enums.mask\_position\_point.**MaskPositionPoint**(*value*)

The part of the face relative to which the mask should be placed.

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

**FOREHEAD** = 'forehead'

**EYES** = 'eyes'

**MOUTH** = 'mouth'

**CHIN** = 'chin'

### MenuButtonType

**class** aiogram.enums.menu\_button\_type.**MenuButtonType**(*value*)

This object represents an type of Menu button

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

**DEFAULT** = 'default'

**COMMANDS** = 'commands'

**WEB\_APP** = 'web\_app'

### MessageEntityType

**class** aiogram.enums.message\_entity\_type.**MessageEntityType**(*value*)

This object represents type of message entity

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

**MENTION** = 'mention'

**HASHTAG** = 'hashtag'

**CASHTAG** = 'cashtag'

**BOT\_COMMAND** = 'bot\_command'

**URL** = 'url'

**EMAIL** = 'email'

**PHONE\_NUMBER** = 'phone\_number'

**BOLD** = 'bold'

**ITALIC** = 'italic'

**UNDERLINE** = 'underline'

**STRIKETHROUGH** = 'strikethrough'

**SPOILER** = 'spoiler'

```
CODE = 'code'
PRE = 'pre'
TEXT_LINK = 'text_link'
TEXT_MENTION = 'text_mention'
CUSTOM_EMOJI = 'custom_emoji'
```

### ParseMode

```
class aiogram.enums.parse_mode.ParseMode(value)
    Formatting options
    Source: https://core.telegram.org/bots/api#formatting-options
    MARKDOWN_V2 = 'MarkdownV2'
    MARKDOWN = 'Markdown'
    HTML = 'HTML'
```

### PassportElementErrorType

```
class aiogram.enums.passport_element_error_type.PassportElementErrorType(value)
    This object represents a passport element error type.
    Source: https://core.telegram.org/bots/api#passportelementerror
    DATA = 'data'
    FRONT_SIDE = 'front_side'
    REVERSE_SIDE = 'reverse_side'
    SELFIE = 'selfie'
    FILE = 'file'
    FILES = 'files'
    TRANSLATION_FILE = 'translation_file'
    TRANSLATION_FILES = 'translation_files'
    UNSPECIFIED = 'unspecified'
```



## PollType

**class** aiogram.enums.poll\_type.**PollType**(*value*)

This object represents poll type

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

**REGULAR** = 'regular'

**QUIZ** = 'quiz'

## StickerFormat

**class** aiogram.enums.sticker\_format.**StickerFormat**(*value*)

Format of the sticker

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

**STATIC** = 'static'

**ANIMATED** = 'animated'

**VIDEO** = 'video'

## StickerType

**class** aiogram.enums.sticker\_type.**StickerType**(*value*)

The part of the face relative to which the mask should be placed.

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

**REGULAR** = 'regular'

**MASK** = 'mask'

**CUSTOM\_EMOJI** = 'custom\_emoji'

## TopicIconColor

**class** aiogram.enums.topic\_icon\_color.**TopicIconColor**(*value*)

Color of the topic icon in RGB format.

Source: [https://github.com/telegramdesktop/tdesktop/blob/991fe491c5ae62705d77aa8fdd44a79caf639c45/Telegram/SourceFiles/data/data\\_forum\\_topic.cpp#L51-L56](https://github.com/telegramdesktop/tdesktop/blob/991fe491c5ae62705d77aa8fdd44a79caf639c45/Telegram/SourceFiles/data/data_forum_topic.cpp#L51-L56)

**BLUE** = 7322096

**YELLOW** = 16766590

**VIOLET** = 13338331

**GREEN** = 9367192

**ROSE** = 16749490

**RED** = 16478047

## UpdateType

```
class aiogram.enums.update_type.UpdateType(value)
    This object represents the complete list of allowed update types
    Source: https://core.telegram.org/bots/api#update

    MESSAGE = 'message'

    EDITED_MESSAGE = 'edited_message'

    CHANNEL_POST = 'channel_post'

    EDITED_CHANNEL_POST = 'edited_channel_post'

    INLINE_QUERY = 'inline_query'

    CHOSEN_INLINE_RESULT = 'chosen_inline_result'

    CALLBACK_QUERY = 'callback_query'

    SHIPPING_QUERY = 'shipping_query'

    PRE_CHECKOUT_QUERY = 'pre_checkout_query'

    POLL = 'poll'

    POLL_ANSWER = 'poll_answer'

    MY_CHAT_MEMBER = 'my_chat_member'

    CHAT_MEMBER = 'chat_member'

    CHAT_JOIN_REQUEST = 'chat_join_request'
```

### 2.3.6 How to download file?

#### Download file manually

First, you must get the *file\_id* of the file you want to download. Information about files sent to the bot is contained in *Message*.

For example, download the document that came to the bot.

```
file_id = message.document.file_id
```

Then use the *getFile* method to get *file\_path*.

```
file = await bot.get_file(file_id)
file_path = file.file_path
```

After that, use the *download\_file* method from the bot object.

## download\_file(...)

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.

```
class aiogram.client.bot.Bot(token: str, session: BaseSession | None = None, parse_mode: str | None = None, disable_web_page_preview: bool | None = None, protect_content: bool | None = None)
```

```
async download_file(file_path: str, destination: BinaryIO | Path | str | None = None, timeout: int = 30, chunk_size: int = 65536, seek: bool = True) → BinaryIO | None
```

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** – File path on Telegram server (You can get it from `aiogram.types.File`)
- **destination** – Filename, file path or instance of `io.IOBase`. For e.g. `io.BytesIO`, defaults to `None`
- **timeout** – Total timeout in seconds, defaults to 30
- **chunk\_size** – File chunks size, defaults to 64 kb
- **seek** – Go to start of file when downloading is finished. Used only for destination with `typing.BinaryIO` type, defaults to `True`

There are two options where you can download the file: to **disk** or to **binary I/O object**.

### Download file to disk

To download file to disk, you must specify the file name or path where to download the file. In this case, the function will return nothing.

```
await bot.download_file(file_path, "text.txt")
```

### Download file to binary I/O object

To download file to binary I/O object, you must specify an object with the `typing.BinaryIO` type or use the default (`None`) value.

In the first case, the function will return your object:

```
my_object = MyBinaryIO()
result: MyBinaryIO = await bot.download_file(file_path, my_object)
# print(result is my_object) # True
```

If you leave the default value, an `io.BytesIO` object will be created and returned.

```
result: io.BytesIO = await bot.download_file(file_path)
```

### Download file in short way

Getting `file_path` manually every time is boring, so you should use the `download` method.

#### `download(...)`

Download file by `file_id` or `Downloadable` object to destination.

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

```
class aiogram.client.bot.Bot(token: str, session: BaseSession | None = None, parse_mode: str | None =
                             None, disable_web_page_preview: bool | None = None, protect_content: bool |
                             None = None)
```

```
    async download(file: str | Downloadable, destination: BinaryIO | Path | str | None = None, timeout: int =
                   30, chunk_size: int = 65536, seek: bool = True) → BinaryIO | None
```

Download file by `file_id` or `Downloadable` object to destination.

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

#### Parameters

- **file** – `file_id` or `Downloadable` object
- **destination** – Filename, file path or instance of `io.IOBase`. For e.g. `io.BytesIO`, defaults to `None`
- **timeout** – Total timeout in seconds, defaults to 30
- **chunk\_size** – File chunks size, defaults to 64 kb
- **seek** – Go to start of file when downloading is finished. Used only for destination with `typing.BinaryIO` type, defaults to `True`

It differs from `download_file` **only** in that it accepts `file_id` or an `Downloadable` object (object that contains the `file_id` attribute) instead of `file_path`.

You can download a file to `disk` or to a *binary I/O* object in the same way.

Example:

```
document = message.document
await bot.download(document)
```

### 2.3.7 How to upload file?

As says [official Telegram Bot API documentation](#) there are three ways to send files (photos, stickers, audio, media, etc.):

If the file is already stored somewhere on the Telegram servers or file is available by the URL, you don't need to reupload it.

But if you need to upload a new file just use subclasses of `InputFile`.

Here are the three different available builtin types of input file:

- `aiogram.types.input_file.FSInputFile` - uploading from file system

- `aiogram.types.input_file.BufferedInputFile` - uploading from buffer
- `aiogram.types.input_file.URLInputFile` - uploading from URL

#### Warning: Be respectful with Telegram

Instances of `InputFile` are reusable. That's mean you can create instance of `InputFile` and sent this file multiple times but Telegram does not recommend to do that and when you upload file once just save their `file_id` and use it in next times.

## Upload from file system

By first step you will need to import `InputFile` wrapper:

```
from aiogram.types import FSInputFile
```

Then you can use it:

```
cat = FSInputFile("cat.png")
agenda = FSInputFile("my-document.pdf", filename="agenda-2019-11-19.pdf")
```

```
class aiogram.types.input_file.FSInputFile(path: str | Path, filename: str | None = None, chunk_size: int
                                           = 65536)
```

```
    __init__(path: str | Path, filename: str | None = None, chunk_size: int = 65536)
```

Represents object for uploading files from filesystem

#### Parameters

- **path** – Path to file
- **filename** – Filename to be propagated to telegram. By default, will be parsed from path
- **chunk\_size** – Uploading chunk size

## Upload from buffer

Files can be also passed from buffer (For example you generate image using `Pillow` and you want to send it to Telegram):

Import wrapper:

```
from aiogram.types import BufferedInputFile
```

And then you can use it:

```
text_file = BufferedInputFile(b"Hello, world!", filename="file.txt")
```

```
class aiogram.types.input_file.BufferedInputFile(file: bytes, filename: str, chunk_size: int = 65536)
```

```
    __init__(file: bytes, filename: str, chunk_size: int = 65536)
```

Represents object for uploading files from filesystem

#### Parameters

- **file** – Bytes
- **filename** – Filename to be propagated to telegram.

- **chunk\_size** – Uploading chunk size

### Upload from url

If you need to upload a file from another server, but the direct link is bound to your server's IP, or you want to bypass native [upload limits](#) by URL, you can use `aiogram.types.input_file.URLInputFile`.

Import wrapper:

```
from aiogram.types import URLInputFile
```

And then you can use it:

```
image = URLInputFile(
    "https://www.python.org/static/community_logos/python-powered-h-140x182.png",
    filename="python-logo.png"
)
```

```
class aiogram.types.input_file.URLInputFile(url: str, headers: Dict[str, Any] | None = None, filename:
    str | None = None, chunk_size: int = 65536, timeout: int =
    30, bot: Bot | None = None)
```

## 2.4 Handling events

*aiogram* includes Dispatcher mechanism. Dispatcher is needed for handling incoming updates from Telegram.

With dispatcher you can do:

- Handle incoming updates;
- Filter incoming events before it will be processed by specific handler;
- Modify event and related data in middlewares;
- Separate bot functionality between different handlers, modules and packages

Dispatcher is also separated into two entities - Router and Dispatcher. Dispatcher is subclass of router and should be always is root router.

Telegram supports two ways of receiving updates:

- *Webhook* - you should configure your web server to receive updates from Telegram;
- *Long polling* - you should request updates from Telegram.

So, you can use both of them with *aiogram*.

## 2.4.1 Router

Usage:

```
from aiogram import Router
from aiogram.types import Message

my_router = Router(name=__name__)

@my_router.message()
async def message_handler(message: Message) -> Any:
    await message.answer('Hello from my router!')
```

**class** aiogram.dispatcher.router.**Router**(\*, name: str | None = None)

Bases: object

Router can route update, and it nested update types like messages, callback query, polls and all other event types.

Event handlers can be registered in observer by two ways:

- By observer method - router.<event\_type>.register(handler, <filters, ...>)
- By decorator - @router.<event\_type>(<filters, ...>)

**\_\_init\_\_**(\*, name: str | None = None) → None

### Parameters

**name** – Optional router name, can be useful for debugging

**include\_router**(router: Router) → Router

Attach another router.

### Parameters

**router** –

### Returns

**include\_routers**(\*routers: Router) → None

Attach multiple routers.

### Parameters

**routers** –

### Returns

**resolve\_used\_update\_types**(skip\_events: Set[str] | None = None) → List[str]

Resolve registered event names

Is useful for getting updates only for registered event types.

### Parameters

**skip\_events** – skip specified event names

### Returns

set of registered names

## Event observers

**Warning:** All handlers always should be asynchronous. The name of the handler function is not important. The event argument name is also not important but it is recommended to not overlap the name with contextual data in due to function can not accept two arguments with the same name.

Here is the list of available observers and examples of how to register handlers

In these examples only decorator-style registering handlers are used, but if you don't like @decorators just use `<event type>.register(...)` method instead.

## Message

**Attention:** Be attentive with filtering this event

You should expect that this event can be with different sets of attributes in different cases

(For example text, sticker and document are always of different content types of message)

Recommended way to check field availability before usage, for example via *magic filter*: `F.text` to handle text, `F.sticker` to handle stickers only and etc.

```
@router.message()
async def message_handler(message: types.Message) -> Any: pass
```

## Edited message

```
@router.edited_message()
async def edited_message_handler(edited_message: types.Message) -> Any: pass
```

## Channel post

```
@router.channel_post()
async def channel_post_handler(channel_post: types.Message) -> Any: pass
```

## Edited channel post

```
@router.edited_channel_post()
async def edited_channel_post_handler(edited_channel_post: types.Message) -> Any: pass
```



### Inline query

```
@router.inline_query()
async def inline_query_handler(inline_query: types.InlineQuery) -> Any: pass
```

### Chosen inline query

```
@router.chosen_inline_result()
async def chosen_inline_result_handler(chosen_inline_result: types.ChosenInlineResult) ->
    Any: pass
```

### Callback query

```
@router.callback_query()
async def callback_query_handler(callback_query: types.CallbackQuery) -> Any: pass
```

### Shipping query

```
@router.shipping_query()
async def shipping_query_handler(shipping_query: types.ShippingQuery) -> Any: pass
```

### Pre checkout query

```
@router.pre_checkout_query()
async def pre_checkout_query_handler(pre_checkout_query: types.PreCheckoutQuery) -> Any:
    pass
```

### Poll

```
@router.poll()
async def poll_handler(poll: types.Poll) -> Any: pass
```

### Poll answer

```
@router.poll_answer()
async def poll_answer_handler(poll_answer: types.PollAnswer) -> Any: pass
```

## Errors

```
@router.errors()
async def error_handler(exception: ErrorEvent) -> Any: pass
```

Is useful for handling errors from other handlers, error event described [here](#)

## Nested routers

### Warning:

**Routers by the way can be nested to an another routers with some limitations:**

1. Router **CAN NOT** include itself 1. Routers **CAN NOT** be used for circular including (router 1 include router 2, router 2 include router 3, router 3 include router 1)

Example:

Listing 1: module\_1.py

```
router2 = Router()

@router2.message()
...
```

Listing 2: module\_2.py

```
from module_2 import router2

router1 = Router()
router1.include_router(router2)
```

## Update

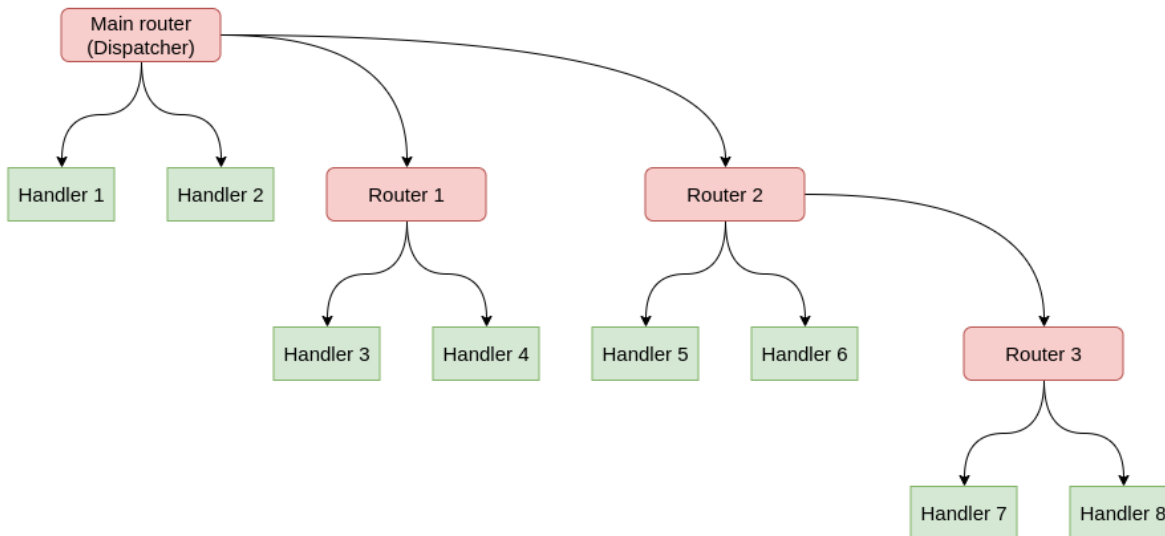
```
@dispatcher.update()
async def message_handler(update: types.Update) -> Any: pass
```

**Warning:** The only root Router (Dispatcher) can handle this type of event.

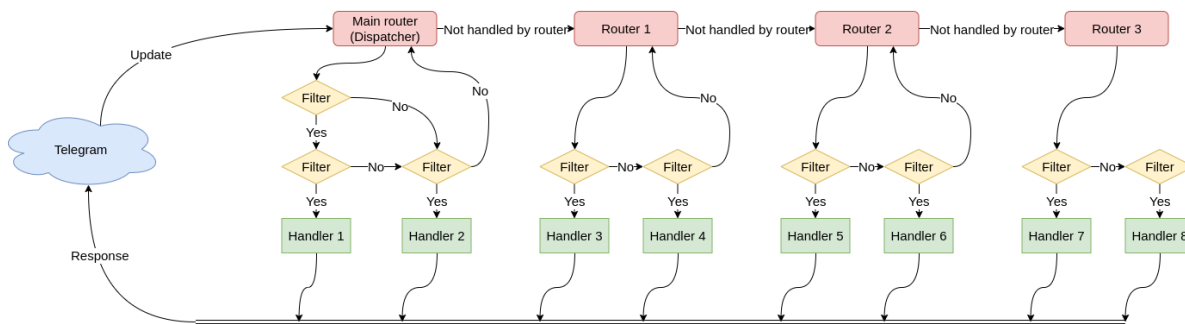
**Note:** Dispatcher already has default handler for this event type, so you can use it for handling all updates that are not handled by any other handlers.

## How it works?

For example, dispatcher has 2 routers, the last router also has one nested router:



In this case update propagation flow will have form:



## 2.4.2 Dispatcher

Dispatcher is root Router and in code Dispatcher can be used directly for routing updates or attach another routers into dispatcher.

Here is only listed base information about Dispatcher. All about writing handlers, filters and etc. you can find in next pages:

- [Router](#)
- [Filtering events](#)

```
class aiogram.dispatcher.dispatcher.Dispatcher(* , storage: BaseStorage | None = None, fsm_strategy:
    FSMStrategy = FSMStrategy.USER_IN_CHAT,
    events_isolation: BaseEventIsolation | None = None,
    disable_fsm: bool = False, name: str | None = None,
    **kwargs: Any)
```

Root router

```
__init__(* , storage: BaseStorage | None = None, fsm_strategy: FSMStrategy =  
    FSMStrategy.USER_IN_CHAT, events_isolation: BaseEventIsolation | None = None, disable_fsm:  
    bool = False, name: str | None = None, **kwargs: Any) → None
```

Root router

#### Parameters

- **storage** – Storage for FSM
- **fsm\_strategy** – FSM strategy
- **events\_isolation** – Events isolation
- **disable\_fsm** – Disable FSM, note that if you disable FSM then you should not use storage and events isolation
- **kwargs** – Other arguments, will be passed as keyword arguments to handlers

```
async feed_raw_update(bot: Bot, update: Dict[str, Any], **kwargs: Any) → Any
```

Main entry point for incoming updates with automatic Dict->Update serializer

#### Parameters

- **bot** –
- **update** –
- **kwargs** –

```
async feed_update(bot: Bot, update: Update, **kwargs: Any) → Any
```

Main entry point for incoming updates Response of this method can be used as Webhook response

#### Parameters

- **bot** –
- **update** –

```
run_polling(*bots: Bot, polling_timeout: int = 10, handle_as_tasks: bool = True, backoff_config:  
    BackoffConfig = BackoffConfig(min_delay=1.0, max_delay=5.0, factor=1.3, jitter=0.1),  
    allowed_updates: List[str] | _SentinelObject | None = sentinel.UNSET, handle_signals: bool  
    = True, close_bot_session: bool = True, **kwargs: Any) → None
```

Run many bots with polling

#### Parameters

- **bots** – Bot instances (one or mre)
- **polling\_timeout** – Long-polling wait time
- **handle\_as\_tasks** – Run task for each event and no wait result
- **backoff\_config** – backoff-retry config
- **allowed\_updates** – List of the update types you want your bot to receive
- **handle\_signals** – handle signals (SIGINT/SIGTERM)
- **close\_bot\_session** – close bot sessions on shutdown
- **kwargs** – contextual data

#### Returns

```
async start_polling(*bots: Bot, polling_timeout: int = 10, handle_as_tasks: bool = True,
                    backoff_config: BackoffConfig = BackoffConfig(min_delay=1.0, max_delay=5.0,
                        factor=1.3, jitter=0.1), allowed_updates: List[str] | _SentinelObject | None =
                        sentinel.UNSET, handle_signals: bool = True, close_bot_session: bool = True,
                    **kwargs: Any) → None
```

Polling runner

#### Parameters

- **bots** – Bot instances (one or more)
- **polling\_timeout** – Long-polling wait time
- **handle\_as\_tasks** – Run task for each event and no wait result
- **backoff\_config** – backoff-retry config
- **allowed\_updates** – List of the update types you want your bot to receive By default, all used update types are enabled (resolved from handlers)
- **handle\_signals** – handle signals (SIGINT/SIGTERM)
- **close\_bot\_session** – close bot sessions on shutdown
- **kwargs** – contextual data

#### Returns

**async stop\_polling**() → None

Execute this method if you want to stop polling programmatically

#### Returns

## Simple usage

Example:

```
dp = Dispatcher()

@dp.message()
async def message_handler(message: types.Message) -> None:
    await SendMessage(chat_id=message.from_user.id, text=message.text)
```

Including routers

Example:

```
dp = Dispatcher()
router1 = Router()
dp.include_router(router1)
```

## Handling updates

All updates can be propagated to the dispatcher by `Dispatcher.feed_update(bot=..., update=...)` method:

```
bot = Bot(...)
dp = Dispatcher()

...

result = await dp.feed_update(bot=bot, update=incoming_update)
```

### 2.4.3 Dependency injection

Dependency injection is a programming technique that makes a class independent of its dependencies. It achieves that by decoupling the usage of an object from its creation. This helps you to follow [SOLID's](#) dependency inversion and single responsibility principles.

#### How it works in aiogram

For each update `aiogram.dispatcher.dispatcher.Dispatcher` passes handling context data. Filters and middleware can also make changes to the context.

To access contextual data you should specify corresponding keyword parameter in handler or filter. For example, to get `aiogram.fsm.context.FSMContext` we do it like that:

```
@router.message(ProfileCompletion.add_photo, F.photo)
async def add_photo(
    message: types.Message, bot: Bot, state: FSMContext
) -> Any:
    ... # do something with photo
```

#### Injecting own dependencies

Aiogram provides several ways to complement / modify contextual data.

The first and easiest way is to simply specify the named arguments in `aiogram.dispatcher.dispatcher.Dispatcher` initialization, polling start methods or `aiogram.webhook.aihttp_server.SimpleRequestHandler` initialization if you use webhooks.

```
async def main() -> None:
    dp = Dispatcher(..., foo=42)
    return await dp.start_polling(
        bot, bar="Bazz"
    )
```

Analogy for webhook:

```
async def main() -> None:
    dp = Dispatcher(..., foo=42)
    handler = SimpleRequestHandler(dispatcher=dp, bot=bot, bar="Bazz")
    ... # starting webhook
```

`aiogram.dispatcher.dispatcher.Dispatcher`'s workflow data also can be supplemented by setting values as in a dictionary:

```
dp = Dispatcher(...)
dp["eggs"] = Spam()
```

The middlewares updates the context quite often. You can read more about them on this page:

- [Middlewares](#)

The last way is to return a dictionary from the filter:

```
from typing import Any, Dict, Optional, Union

from aiogram import Router
from aiogram.filters import Filter
from aiogram.types import Message, User

router = Router(name=__name__)

class HelloFilter(Filter):
    def __init__(self, name: Optional[str] = None) -> None:
        self.name = name

    async def __call__(
        self,
        message: Message,
        event_from_user: User
        # Filters also can accept keyword parameters like in handlers
    ) -> Union[bool, Dict[str, Any]]:
        if message.text.casefold() == "hello":
            # Returning a dictionary that will update the context data
            return {"name": event_from_user.mention_html(name=self.name)}
        return False

@router.message(HelloFilter())
async def my_handler(
    message: Message, name: str # Now we can accept "name" as named parameter
) -> Any:
    return message.answer("Hello, {name}!".format(name=name))
```

...or using [MagicFilter](#) with `.as_(...)` method.

## 2.4.4 Filtering events

Filters is needed for routing updates to the specific handler. Searching of handler is always stops on first match set of filters are pass.

*aiogram* has some builtin useful filters.

### Builtin filters

Here is list of builtin filters:

#### Command

##### Usage

1. Filter single variant of commands: `Command("start")`
2. Handle command by regexp pattern: `Command(re.compile(r"item_(d+)"))`
3. Match command by multiple variants: `Command("item", re.compile(r"item_(d+)"))`
4. Handle commands in public chats intended for other bots: `Command("command", ignore_mention=True)`
5. Use `aiogram.types.bot_command.BotCommand` object as command reference  
`Command(BotCommand(command="command", description="My awesome command"))`

<b>Warning:</b> Command cannot include spaces or any whitespace
---

```
class aiogram.filters.command.Command(*values: str | Pattern | BotCommand, commands: Sequence[str |
                                          Pattern | BotCommand] | str | Pattern | BotCommand | None =
                                          None, prefix: str = '/', ignore_case: bool = False, ignore_mention:
                                          bool = False, magic: MagicFilter | None = None)
```

This filter can be helpful for handling commands from the text messages.

Works only with `aiogram.types.message.Message` events which have the `text`.

```
__init__(*values: str | Pattern | BotCommand, commands: Sequence[str | Pattern | BotCommand] | str |
          Pattern | BotCommand | None = None, prefix: str = '/', ignore_case: bool = False,
          ignore_mention: bool = False, magic: MagicFilter | None = None)
```

List of commands (string or compiled regexp patterns)

#### Parameters

- **prefix** – Prefix for command. Prefix is always a single char but here you can pass all of allowed prefixes, for example: `"/!"` will work with commands prefixed by `"/"` or `"/!"`.
- **ignore\_case** – Ignore case (Does not work with regexp, use flags instead)
- **ignore\_mention** – Ignore bot mention. By default, bot can not handle commands intended for other bots
- **magic** – Validate command object via Magic filter after all checks done

When filter is passed the `aiogram.filters.command.CommandObject` will be passed to the handler argument `command`



```
class aiogram.filters.command.CommandObject(prefix: str = '/', command: str = "", mention: str | None =  
None, args: str | None = None, regexp_match: Match[str] |  
None = None, magic_result: Any | None = None)
```

Instance of this object is always has command and it prefix. Can be passed as keyword argument **command** to the handler

**prefix:** `str = '/'`

Command prefix

**command:** `str = ''`

Command without prefix and mention

**mention:** `str | None = None`

Mention (if available)

**args:** `str | None = None`

Command argument

**regexp\_match:** `Match[str] | None = None`

Will be presented match result if the command is presented as regexp in filter

**magic\_result:** `Any | None = None`

**property mentioned:** `bool`

This command has mention?

**property text:** `str`

Generate original text from object

## Allowed handlers

Allowed update types for this filter:

- *message*
- *edited\_message*

## ChatMemberUpdated

### Usage

Handle user leave or join events

```
from aiogram.filters import IS_MEMBER, IS_NOT_MEMBER

@router.chat_member(ChatMemberUpdatedFilter(IS_MEMBER >> IS_NOT_MEMBER))
async def on_user_leave(event: ChatMemberUpdated): ...

@router.chat_member(ChatMemberUpdatedFilter(IS_NOT_MEMBER >> IS_MEMBER))
async def on_user_join(event: ChatMemberUpdated): ...
```

Or construct your own terms via using pre-defined set of statuses and transitions.

## Explanation

```
class aiogram.filters.chat_member_updated.ChatMemberUpdatedFilter(member_status_changed:
                                                                    _MemberStatusMarker |
                                                                    _MemberStatusGroupMarker
                                                                    | _MemberStatusTransition)

    member_status_changed
```

You can import from `aiogram.filters` all available variants of *statuses*, *status groups* or *transitions*:

## Statuses

name	Description
CREATOR	Chat owner
ADMINISTRATOR	Chat administrator
MEMBER	Member of the chat
RESTRICTED	Restricted user (can be not member)
LEFT	Isn't member of the chat
KICKED	Kicked member by administrators

Statuses can be extended with *is\_member* flag by prefixing with + (for `is_member == True`) or - (for `is_member == False`) symbol, like `+RESTRICTED` or `-RESTRICTED`

## Status groups

The particular statuses can be combined via bitwise or operator, like `CREATOR | ADMINISTRATOR`

name	Description
IS_MEMBER	Combination of (CREATOR   ADMINISTRATOR   MEMBER   +RESTRICTED) statuses.
IS_ADMIN	Combination of (CREATOR   ADMINISTRATOR) statuses.
IS_NOT_MEMBER	Combination of (LEFT   KICKED   -RESTRICTED) statuses.

## Transitions

Transitions can be defined via bitwise shift operators `>>` and `<<`. Old chat member status should be defined in the left side for `>>` operator (right side for `<<`) and new status should be specified on the right side for `>>` operator (left side for `<<`)

The direction of transition can be changed via bitwise inversion operator: `~JOIN_TRANSITION` will produce swap of old and new statuses.

name	Description
JOIN_TRANSIT	Means status changed from IS_NOT_MEMBER to IS_MEMBER (IS_NOT_MEMBER >> IS_MEMBER)
LEAVE_TRANSI	Means status changed from IS_MEMBER to IS_NOT_MEMBER (~JOIN_TRANSITION)
PROMOTED_TRA	Means status changed from (MEMBER   RESTRICTED   LEFT   KICKED) >> ADMINISTRATOR ((MEMBER   RESTRICTED   LEFT   KICKED) >> ADMINISTRATOR)

---

**Note:** Note that if you define the status unions (via `|`) you will need to add brackets for the statement before use shift operator in due to operator priorities.

---

## Allowed handlers

Allowed update types for this filter:

- `my_chat_member`
- `chat_member`

## Magic filters

---

**Note:** This page still in progress. Has many incorrectly worded sentences.

---

Is external package maintained by *aiogram* core team.

By default installs with *aiogram* and also is available on [PyPi - magic-filter](#). That's mean you can install it and use with any other libraries and in own projects without depending *aiogram* installed.

## Usage

The **magic\_filter** package implements class shortly named `magic_filter.F` that's mean `F` can be imported from *aiogram* or *magic\_filter*. `F` is alias for `MagicFilter`.

---

**Note:** Note that *aiogram* has an small extension over *magic-filter* and if you want to use this extension you should import `magic` from *aiogram* instead of *magic\_filter* package

---

The `MagicFilter` object is callable, supports *some actions* and memorize the attributes chain and the action which should be checked on demand.

So that's mean you can chain attribute getters, describe simple data validations and then call the resulted object passing single object as argument, for example make attributes chain `F.foo.bar.baz` then add action `'F.foo.bar.baz == 'spam'` and then call the resulted object - `(F.foo.bar.baz == 'spam').resolve(obj)`

## Possible actions

Magic filter object supports some of basic logical operations over object attributes

## Exists or not None

Default actions.

```
F.photo # lambda message: message.photo
```

## Equals

```
F.text == 'hello' # lambda message: message.text == 'hello'
F.from_user.id == 42 # lambda message: message.from_user.id == 42
```

## Is one of

Can be used as method named `in_` or as matmul operator `@` with any iterable

```
F.from_user.id.in_({42, 1000, 123123}) # lambda query: query.from_user.id in {42, 1000, 123123}
F.data.in_({'foo', 'bar', 'baz'}) # lambda query: query.data in {'foo', 'bar', 'baz'}
```

## Contains

```
F.text.contains('foo') # lambda message: 'foo' in message.text
```

## String startswith/endswith

Can be applied only for text attributes

```
F.text.startswith('foo') # lambda message: message.text.startswith('foo')
F.text.endswith('bar') # lambda message: message.text.endswith('bar')
```

## Regexp

```
F.text.regexp(r'Hello, .+') # lambda message: re.match(r'Hello, .+', message.text)
```

## Custom function

Accepts any callable. Callback will be called when filter checks result

```
F.chat.func(lambda chat: chat.id == -42) # lambda message: (lambda chat: chat.id == -42)(message.chat)
```

## Inverting result

Any of available operation can be inverted by bitwise inversion - ~

```
~(F.text == 'spam') # lambda message: message.text != 'spam'
~F.text.startswith('spam') # lambda message: not message.text.startswith('spam')
```

## Combining

All operations can be combined via bitwise and/or operators - &/|

```
(F.from_user.id == 42) & (F.text == 'admin')
F.text.startswith('a') | F.text.endswith('b')
(F.from_user.id.in_({42, 777, 911})) & (F.text.startswith('!') | F.text.startswith('/'))
→ & F.text.contains('ban')
```

## Attribute modifiers - string manipulations

Make text upper- or lower-case

Can be used only with string attributes.

```
F.text.lower() == 'test' # lambda message: message.text.lower() == 'test'
F.text.upper().in_({'FOO', 'BAR'}) # lambda message: message.text.upper() in {'FOO', 'BAR'}
F.text.len() == 5 # lambda message: len(message.text) == 5
```

## Get filter result as handler argument

This part is not available in *magic-filter* directly but can be used with *aiogram*

```
from aiogram import F
...
@router.message(F.text.regex(r"^(\d+)$").as_("digits"))
async def any_digits_handler(message: Message, digits: Match[str]):
    await message.answer(html.quote(str(digits)))
```

## Usage in aiogram

```
@router.message(F.text == 'hello')
@router.inline_query(F.data == 'button:1')
@router.message(F.text.startswith('foo'))
@router.message(F.content_type.in_({'text', 'sticker'}))
@router.message(F.text.regex(r'\d+'))
...
# Many others cases when you will need to check any of available event attribute
```

## MagicData

### Usage

1. `MagicData(F.event.from_user.id == F.config.admin_id)` (Note that `config` should be passed from middleware)

### Explanation

**class** `aiogram.filters.magic_data.MagicData(magic_data: MagicFilter)`

This filter helps to filter event with contextual data

**magic\_data**

Can be imported:

- `from aiogram.filters import MagicData`

### Allowed handlers

Allowed update types for this filter:

- `message`
- `edited_message`
- `channel_post`
- `edited_channel_post`
- `inline_query`
- `chosen_inline_result`
- `callback_query`
- `shipping_query`
- `pre_checkout_query`
- `poll`
- `poll_answer`
- `my_chat_member`
- `chat_member`
- `chat_join_request`
- `error`

## Callback Data Factory & Filter

**class** aiogram.filters.callback\_data.CallbackData

Base class for callback data wrapper

This class should be used as super-class of user-defined callbacks.

The class-keyword **prefix** is required to define prefix and also the argument **sep** can be passed to define separator (default is **:**).

**pack()** → str

Generate callback data string

**Returns**

valid callback data for Telegram Bot API

**classmethod** **unpack**(value: str) → T

Parse callback data string

**Parameters**

**value** – value from Telegram

**Returns**

instance of CallbackData

**classmethod** **filter**(rule: MagicFilter | None = None) → CallbackQueryFilter

Generates a filter for callback query with rule

**Parameters**

**rule** – magic rule

**Returns**

instance of filter

## Usage

Create subclass of CallbackData:

```
class MyCallback(CallbackData, prefix="my"):
    foo: str
    bar: int
```

After that you can generate any callback based on this class, for example:

```
cb1 = MyCallback(foo="demo", bar=42)
cb1.pack() # returns 'my:demo:42'
cb1.unpack('my:demo:42') # returns <MyCallback(foo="demo", bar=42)>
```

So... Now you can use this class to generate any callbacks with defined structure

```
...
# Pass it into the markup
InlineKeyboardButton(
    text="demo",
    callback_data=MyCallback(foo="demo", bar="42").pack() # value should be packed to
    ↪ string
```

(continues on next page)

(continued from previous page)

```
)
...
```

... and handle by specific rules

```
# Filter callback by type and value of field :code:`foo`
@router.callback_query(MyCallback.filter(F.foo == "demo"))
async def my_callback_foo(query: CallbackQuery, callback_data: MyCallback):
    await query.answer(...)
    ...
    print("bar =", callback_data.bar)
```

Also can be used in *Keyboard builder*:

```
builder = InlineKeyboardBuilder()
builder.button(
    text="demo",
    callback_data=MyCallback(foo="demo", bar="42") # Value can be not packed to string,
    ↪ inplace, because builder knows what to do with callback instance
)
```

Another abstract example:

```
class Action(str, Enum):
    ban = "ban"
    kick = "kick"
    warn = "warn"

class AdminAction(CallbackData, prefix="adm"):
    action: Action
    chat_id: int
    user_id: int

...
# Inside handler
builder = InlineKeyboardBuilder()
for action in Action:
    builder.button(
        text=action.value.title(),
        callback_data=AdminAction(action=action, chat_id=chat_id, user_id=user_id),
    )
await bot.send_message(
    chat_id=admins_chat,
    text=f"What do you want to do with {html.quote(name)}",
    reply_markup=builder.as_markup(),
)
...

@router.callback_query(AdminAction.filter(F.action == Action.ban))
async def ban_user(query: CallbackQuery, callback_data: AdminAction, bot: Bot):
    await bot.ban_chat_member(
        chat_id=callback_data.chat_id,
        user_id=callback_data.user_id,
```

(continues on next page)



(continued from previous page)

```
    ...
)
```

## Known limitations

Allowed types and their subclasses:

- `str`
- `int`
- `bool`
- `float`
- `Decimal` (from `decimal` import `Decimal`)
- `Fraction` (from `fractions` import `Fraction`)
- `UUID` (from `uuid` import `UUID`)
- `Enum` (from `enum` import `Enum`, only for string enums)
- `IntEnum` (from `enum` import `IntEnum`, only for int enums)

---

**Note:** Note that the integer `Enum`'s should be always is subclasses of `IntEnum` in due to parsing issues.

---

## Exceptions

This filters can be helpful for handling errors from the text messages.

**class** `aiogram.filters.exception.ExceptionTypeFilter(*exceptions: Type[Exception])`

Allows to match exception by type

**exceptions**

**class** `aiogram.filters.exception.ExceptionMessageFilter(pattern: str | Pattern[str])`

Allow to match exception by message

**pattern**

## Allowed handlers

Allowed update types for this filters:

- `error`

## Writing own filters

Filters can be:

- Asynchronous function (`async def my_filter(*args, **kwargs): pass`)
- Synchronous function (`def my_filter(*args, **kwargs): pass`)
- Anonymous function (`lambda event: True`)
- Any awaitable object
- Subclass of `aiogram.filters.base.Filter`
- Instances of `MagicFilter`

and should return bool or dict. If the dictionary is passed as result of filter - resulted data will be propagated to the next filters and handler as keywords arguments.

## Base class for own filters

### `class aiogram.filters.base.Filter`

If you want to register own filters like builtin filters you will need to write subclass of this class with overriding the `__call__` method and adding filter attributes.

**abstract** `async __call__(*args: Any, **kwargs: Any) → bool | Dict[str, Any]`

This method should be overridden.

Accepts incoming event and should return boolean or dict.

#### Returns

bool or `Dict[str, Any]`

**update\_handler\_flags**(`flags: Dict[str, Any]`) → None

Also if you want to extend handler flags with using this filter you should implement this method

#### Parameters

**flags** – existing flags, can be updated directly

## Own filter example

For example if you need to make simple text filter:

```
from aiogram import Router
from aiogram.filters import Filter
from aiogram.types import Message

router = Router()

class MyFilter(Filter):
    def __init__(self, my_text: str) -> None:
        self.my_text = my_text

    async def __call__(self, message: Message) -> bool:
        return message.text == self.my_text
```

(continues on next page)

(continued from previous page)

```
@router.message(MyFilter("hello"))
async def my_handler(message: Message):
    ...
```

## Combining Filters

In general, all filters can be combined in two ways

### Recommended way

If you specify multiple filters in a row, it will be checked with an “and” condition:

```
@<router>.message(F.text.startswith("show"), F.text.endswith("example"))
```

Also, if you want to use two alternative ways to run the same handler (“or” condition) you can register the handler twice or more times as you like

```
@<router>.message(F.text == "hi")
@<router>.message(CommandStart())
```

Also sometimes you will need to invert the filter result, for example you have an *IsAdmin* filter and you want to check if the user is not an admin

```
@<router>.message(~IsAdmin())
```

### Another possible way

An alternative way is to combine using special functions (`and_f()`, `or_f()`, `invert_f()` from `aiogram.filters` module):

```
and_f(F.text.startswith("show"), F.text.endswith("example"))
or_f(F.text(text="hi"), CommandStart())
invert_f(IsAdmin())
and_f(<A>, or_f(<B>, <C>))
```

## 2.4.5 Long-polling

Long-polling is a technology that allows a Telegram server to send updates in case when you don’t have dedicated IP address or port to receive webhooks for example on a developer machine.

To use long-polling mode you should use `aiogram.dispatcher.dispatcher.Dispatcher.start_polling()` or `aiogram.dispatcher.dispatcher.Dispatcher.run_polling()` methods.

**Note:** You can use polling from only one polling process per single Bot token, in other case Telegram server will return an error.

---

**Note:** If you will need to scale your bot, you should use webhooks instead of long-polling.

---

---

**Note:** If you will use multibot mode, you should use webhook mode for all bots.

---

## Example

This example will show you how to create simple echo bot based on long-polling.

```
import asyncio
import logging
import sys
from os import getenv

from aiogram import Bot, Dispatcher, Router, types
from aiogram.enums import ParseMode
from aiogram.filters import CommandStart
from aiogram.types import Message
from aiogram.utils.markdown import hbold

# Bot token can be obtained via https://t.me/BotFather
TOKEN = getenv("BOT_TOKEN")

# All handlers should be attached to the Router (or Dispatcher)
dp = Dispatcher()

@dp.message(CommandStart())
async def command_start_handler(message: Message) -> None:
    """
    This handler receives messages with `/start` command
    """
    # Most event objects have aliases for API methods that can be called in events'
    context
    # For example if you want to answer to incoming message you can use `message.answer(
    ..)` alias
    # and the target chat will be passed to :ref:`aiogram.methods.send_message`
    SendMessage`
    # method automatically or call API method directly via
    # Bot instance: `bot.send_message(chat_id=message.chat.id, ...)`
    await message.answer(f"Hello, {hbold(message.from_user.full_name)}!")

@dp.message()
async def echo_handler(message: types.Message) -> None:
    """
    Handler will forward receive a message back to the sender

    By default, message handler will handle all message types (like a text, photo,
    sticker etc.)
```

(continues on next page)

(continued from previous page)

```

"""
try:
    # Send a copy of the received message
    await message.send_copy(chat_id=message.chat.id)
except TypeError:
    # But not all the types is supported to be copied so need to handle it
    await message.answer("Nice try!")

async def main() -> None:
    # Initialize Bot instance with a default parse mode which will be passed to all API_
    ↪ calls
    bot = Bot(TOKEN, parse_mode=ParseMode.HTML)
    # And the run events dispatching
    await dp.start_polling(bot)

if __name__ == "__main__":
    logging.basicConfig(level=logging.INFO, stream=sys.stdout)
    asyncio.run(main())

```

## 2.4.6 Webhook

Telegram Bot API supports webhook. If you set webhook for your bot, Telegram will send updates to the specified url. You can use `aiogram.methods.set_webhook.SetWebhook()` method to specify a url and receive incoming updates on it.

---

**Note:** If you use webhook, you can't use long polling at the same time.

---

Before start i'll recommend you to read [official Telegram's documentation about webhook](#)

After you read it, you can start to read this section.

Generally to use webhook with aiogram you should use any async web framework. Buy out of the box aiogram has an aiohttp integration, so we'll use it.

---

**Note:** You can use any async web framework you want, but you should write your own integration if you don't use aiohttp.

---

### aiohttp integration

Out of the box aiogram has aiohttp integration, so you can use it.

Here is available few ways to do it using different implementations of the webhook controller:

- `aiogram.webhook.aiohttp_server.BaseRequestHandler` - Abstract class for aiohttp webhook controller
- `aiogram.webhook.aiohttp_server.SimpleRequestHandler` - Simple webhook controller, uses single Bot instance

- `aiogram.webhook.aihttp_server.TokenBasedRequestHandler` - Token based webhook controller, uses multiple Bot instances and tokens

You can use it as is or inherit from it and override some methods.

```
class aiogram.webhook.aihttp_server.BaseRequestHandler(dispatcher: Dispatcher,  
                                                         handle_in_background: bool = False,  
                                                         **data: Any)
```

```
__init__(dispatcher: Dispatcher, handle_in_background: bool = False, **data: Any) → None
```

Base handler that helps to handle incoming request from aiohttp and propagate it to the Dispatcher

#### Parameters

- **dispatcher** – instance of `aiogram.dispatcher.dispatcher.Dispatcher`
- **handle\_in\_background** – immediately responds to the Telegram instead of a waiting end of a handler process

```
register(app: None, /, path: str, **kwargs: Any) → None
```

Register route and shutdown callback

#### Parameters

- **app** – instance of aiohttp Application
- **path** – route path
- **kwargs** –

```
abstract async resolve_bot(request: Request) → Bot
```

This method should be implemented in subclasses of this class.

Resolve Bot instance from request.

#### Parameters

**request** –

#### Returns

Bot instance

```
class aiogram.webhook.aihttp_server.SimpleRequestHandler(dispatcher: Dispatcher, bot: Bot,  
                                                         handle_in_background: bool = True,  
                                                         secret_token: str | None = None, **data:  
                                                         Any)
```

```
__init__(dispatcher: Dispatcher, bot: Bot, handle_in_background: bool = True, secret_token: str | None =  
        None, **data: Any) → None
```

Handler for single Bot instance

#### Parameters

- **dispatcher** – instance of `aiogram.dispatcher.dispatcher.Dispatcher`
- **handle\_in\_background** – immediately responds to the Telegram instead of a waiting end of handler process
- **bot** – instance of `aiogram.client.bot.Bot`

```
async close() → None
```

Close bot session

**register**(*app: None, /, path: str, \*\*kwargs: Any*) → None

Register route and shutdown callback

#### Parameters

- **app** – instance of aiohttp Application
- **path** – route path
- **kwargs** –

**async resolve\_bot**(*request: Request*) → Bot

This method should be implemented in subclasses of this class.

Resolve Bot instance from request.

#### Parameters

**request** –

#### Returns

Bot instance

```
class aiogram.webhook.aiohttp_server.TokenBasedRequestHandler(dispatcher: Dispatcher,
                                                             handle_in_background: bool =
                                                             True, bot_settings: Dict[str, Any] |
                                                             None = None, **data: Any)
```

```
__init__(dispatcher: Dispatcher, handle_in_background: bool = True, bot_settings: Dict[str, Any] | None =
None, **data: Any) → None
```

Handler that supports multiple bots the context will be resolved from path variable 'bot\_token'

---

**Note:** This handler is not recommended in due to token is available in URL and can be logged by reverse proxy server or other middleware.

---

#### Parameters

- **dispatcher** – instance of *aiogram.dispatcher.dispatcher.Dispatcher*
- **handle\_in\_background** – immediately responds to the Telegram instead of a waiting end of handler process
- **bot\_settings** – kwargs that will be passed to new Bot instance

**register**(*app: None, /, path: str, \*\*kwargs: Any*) → None

Validate path, register route and shutdown callback

#### Parameters

- **app** – instance of aiohttp Application
- **path** – route path
- **kwargs** –

**async resolve\_bot**(*request: Request*) → Bot

Get bot token from a path and create or get from cache Bot instance

#### Parameters

**request** –

#### Returns

## Security

Telegram supports two methods to verify incoming requests that they are from Telegram:

### Using a secret token

When you set webhook, you can specify a secret token and then use it to verify incoming requests.

### Using IP filtering

You can specify a list of IP addresses from which you expect incoming requests, and then use it to verify incoming requests.

It can be acy using firewall rules or nginx configuration or middleware on application level.

So, aiogram has an implementation of the IP filtering middleware for aiohttp.

```
aiogram.webhook.aiohttp_server.ip_filter_middleware(ip_filter: IPFilter) → Callable[[Request,  
                                           Callable[[Request],  
                                           Awaitable[StreamResponse]]], Awaitable[Any]]
```

#### Parameters

**ip\_filter** –

#### Returns

```
class aiogram.webhook.security.IPFilter(ips: Sequence[str | IPv4Network | IPv4Address] | None = None)  
    __init__(ips: Sequence[str | IPv4Network | IPv4Address] | None = None)
```

## Examples

### Behind reverse proxy

In this example we'll use aiohttp as web framework and nginx as reverse proxy.

```
"""  
This example shows how to use webhook on behind of any reverse proxy (nginx, traefik, ↵  
↵ingress etc.)  
"""  
  
import logging  
import sys  
from os import getenv  
  
from aiohttp import web  
  
from aiogram import Bot, Dispatcher, Router, types  
from aiogram.enums import ParseMode  
from aiogram.filters import CommandStart  
from aiogram.types import Message  
from aiogram.utils.markdown import hbold  
from aiogram.webhook.aiohttp_server import SimpleRequestHandler, setup_application
```

(continues on next page)



(continued from previous page)

```

# Bot token can be obtained via https://t.me/BotFather
TOKEN = getenv("BOT_TOKEN")

# Webserver settings
# bind localhost only to prevent any external access
WEB_SERVER_HOST = "127.0.0.1"
# Port for incoming request from reverse proxy. Should be any available port
WEB_SERVER_PORT = 8080

# Path to webhook route, on which Telegram will send requests
WEBHOOK_PATH = "/webhook"
# Secret key to validate requests from Telegram (optional)
WEBHOOK_SECRET = "my-secret"
# Base URL for webhook will be used to generate webhook URL for Telegram,
# in this example it is used public DNS with HTTPS support
BASE_WEBHOOK_URL = "https://aiogram.dev/"

# All handlers should be attached to the Router (or Dispatcher)
router = Router()

@router.message(CommandStart())
async def command_start_handler(message: Message) -> None:
    """
    This handler receives messages with `/start` command
    """
    # Most event objects have aliases for API methods that can be called in events
    ↪ context
    # For example if you want to answer to incoming message you can use `message.answer(
    ↪ ..)` alias
    # and the target chat will be passed to :ref:`aiogram.methods.send_message`
    ↪ SendMessage`
    # method automatically or call API method directly via
    # Bot instance: `bot.send_message(chat_id=message.chat.id, ...)`
    await message.answer(f"Hello, {bhold(message.from_user.full_name)}!")

@router.message()
async def echo_handler(message: types.Message) -> None:
    """
    Handler will forward receive a message back to the sender

    By default, message handler will handle all message types (like text, photo, sticker
    ↪ etc.)
    """
    try:
        # Send a copy of the received message
        await message.send_copy(chat_id=message.chat.id)
    except TypeError:
        # But not all the types is supported to be copied so need to handle it
        await message.answer("Nice try!")

```

(continues on next page)

(continued from previous page)

```

async def on_startup(bot: Bot) -> None:
    # If you have a self-signed SSL certificate, then you will need to send a public
    # certificate to Telegram
    await bot.set_webhook(f"{BASE_WEBHOOK_URL}{WEBHOOK_PATH}", secret_token=WEBHOOK_
    ↪SECRET)

def main() -> None:
    # Dispatcher is a root router
    dp = Dispatcher()
    # ... and all other routers should be attached to Dispatcher
    dp.include_router(router)

    # Register startup hook to initialize webhook
    dp.startup.register(on_startup)

    # Initialize Bot instance with a default parse mode which will be passed to all API_
    ↪calls
    bot = Bot(TOKEN, parse_mode=ParseMode.HTML)

    # Create aiohttp.web.Application instance
    app = web.Application()

    # Create an instance of request handler,
    # aiogram has few implementations for different cases of usage
    # In this example we use SimpleRequestHandler which is designed to handle simple_
    ↪cases
    webhook_requests_handler = SimpleRequestHandler(
        dispatcher=dp,
        bot=bot,
        secret_token=WEBHOOK_SECRET,
    )
    # Register webhook handler on application
    webhook_requests_handler.register(app, path=WEBHOOK_PATH)

    # Mount dispatcher startup and shutdown hooks to aiohttp application
    setup_application(app, dp, bot=bot)

    # And finally start webserver
    web.run_app(app, host=WEB_SERVER_HOST, port=WEB_SERVER_PORT)

if __name__ == "__main__":
    logging.basicConfig(level=logging.INFO, stream=sys.stdout)
    main()

```

When you use nginx as reverse proxy, you should set *proxy\_pass* to your aiohttp server address.

```

location /webhook {
    proxy_set_header Host $http_host;

```

(continues on next page)

(continued from previous page)

```

proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
proxy_redirect off;
proxy_buffering off;
proxy_pass http://127.0.0.1:8080;
}

```

### Without reverse proxy (not recommended)

In case you want can't use reverse proxy, you can use aiohttp's ssl context.

Also this example contains usage with self-signed certificate.

```

"""
This example shows how to use webhook with SSL certificate.
"""

import logging
import ssl
import sys
from os import getenv

from aiohttp import web

from aiogram import Bot, Dispatcher, Router, types
from aiogram.enums import ParseMode
from aiogram.filters import CommandStart
from aiogram.types import FSInputFile, Message
from aiogram.utils.markdown import hbold
from aiogram.webhook.aiohttp_server import SimpleRequestHandler, setup_application

# Bot token can be obtained via https://t.me/BotFather
TOKEN = getenv("BOT_TOKEN")

# Webserver settings
# bind localhost only to prevent any external access
WEB_SERVER_HOST = "127.0.0.1"
# Port for incoming request from reverse proxy. Should be any available port
WEB_SERVER_PORT = 8080

# Path to webhook route, on which Telegram will send requests
WEBHOOK_PATH = "/webhook"
# Secret key to validate requests from Telegram (optional)
WEBHOOK_SECRET = "my-secret"
# Base URL for webhook will be used to generate webhook URL for Telegram,
# in this example it is used public address with TLS support
BASE_WEBHOOK_URL = "https://aiogram.dev"

# Path to SSL certificate and private key for self-signed certificate.
WEBHOOK_SSL_CERT = "/path/to/cert.pem"
WEBHOOK_SSL_PRIV = "/path/to/private.key"

# All handlers should be attached to the Router (or Dispatcher)

```

(continues on next page)

(continued from previous page)

```

router = Router()

@router.message(CommandStart())
async def command_start_handler(message: Message) -> None:
    """
    This handler receives messages with `/start` command
    """
    # Most event objects have aliases for API methods that can be called in events'
    ↪ context
    # For example if you want to answer to incoming message you can use `message.answer(
    ↪ ..)` alias
    # and the target chat will be passed to :ref:`aiogram.methods.send_message`.
    ↪ SendMessage`
    # method automatically or call API method directly via
    # Bot instance: `bot.send_message(chat_id=message.chat.id, ...)`
    await message.answer(f"Hello, {hbold(message.from_user.full_name)}!")

@router.message()
async def echo_handler(message: types.Message) -> None:
    """
    Handler will forward receive a message back to the sender

    By default, message handler will handle all message types (like text, photo, sticker
    ↪ etc.)
    """
    try:
        # Send a copy of the received message
        await message.send_copy(chat_id=message.chat.id)
    except TypeError:
        # But not all the types is supported to be copied so need to handle it
        await message.answer("Nice try!")

async def on_startup(bot: Bot) -> None:
    # In case when you have a self-signed SSL certificate, you need to send the
    ↪ certificate
    # itself to Telegram servers for validation purposes
    # (see https://core.telegram.org/bots/self-signed)
    # But if you have a valid SSL certificate, you SHOULD NOT send it to Telegram
    ↪ servers.
    await bot.set_webhook(
        f"{BASE_WEBHOOK_URL}{WEBHOOK_PATH}",
        certificate=FSInputFile(WEBHOOK_SSL_CERT),
        secret_token=WEBHOOK_SECRET,
    )

def main() -> None:
    # Dispatcher is a root router
    dp = Dispatcher()

```

(continues on next page)

(continued from previous page)

```

# ... and all other routers should be attached to Dispatcher
dp.include_router(router)

# Register startup hook to initialize webhook
dp.startup.register(on_startup)

# Initialize Bot instance with a default parse mode which will be passed to all API
↳ calls
bot = Bot(TOKEN, parse_mode=ParseMode.HTML)

# Create aiohttp.web.Application instance
app = web.Application()

# Create an instance of request handler,
# aiogram has few implementations for different cases of usage
# In this example we use SimpleRequestHandler which is designed to handle simple
↳ cases
webhook_requests_handler = SimpleRequestHandler(
    dispatcher=dp,
    bot=bot,
    secret_token=WEBHOOK_SECRET,
)
# Register webhook handler on application
webhook_requests_handler.register(app, path=WEBHOOK_PATH)

# Mount dispatcher startup and shutdown hooks to aiohttp application
setup_application(app, dp, bot=bot)

# Generate SSL context
context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)
context.load_cert_chain(WEBHOOK_SSL_CERT, WEBHOOK_SSL_PRIV)

# And finally start webserver
web.run_app(app, host=WEB_SERVER_HOST, port=WEB_SERVER_PORT, ssl_context=context)

if __name__ == "__main__":
    logging.basicConfig(level=logging.INFO, stream=sys.stdout)
    main()

```

### With using other web framework

You can pass incoming request to aiogram's webhook controller from any web framework you want.

Read more about it in `aiogram.dispatcher.dispatcher.Dispatcher.feed_webhook_update()` or `aiogram.dispatcher.dispatcher.Dispatcher.feed_update()` methods.

```

update = Update.model_validate(await request.json(), context={"bot": bot})
await dispatcher.feed_update(update)

```

**Note:** If you want to use reply into webhook, you should check that result of the `feed_update` methods is an instance

of API method and build `multipart/form-data` or `application/json` response body manually.

---

## 2.4.7 Finite State Machine

A finite-state machine (FSM) or finite-state automaton (FSA, plural: automata), finite automaton, or simply a state machine, is a mathematical model of computation.

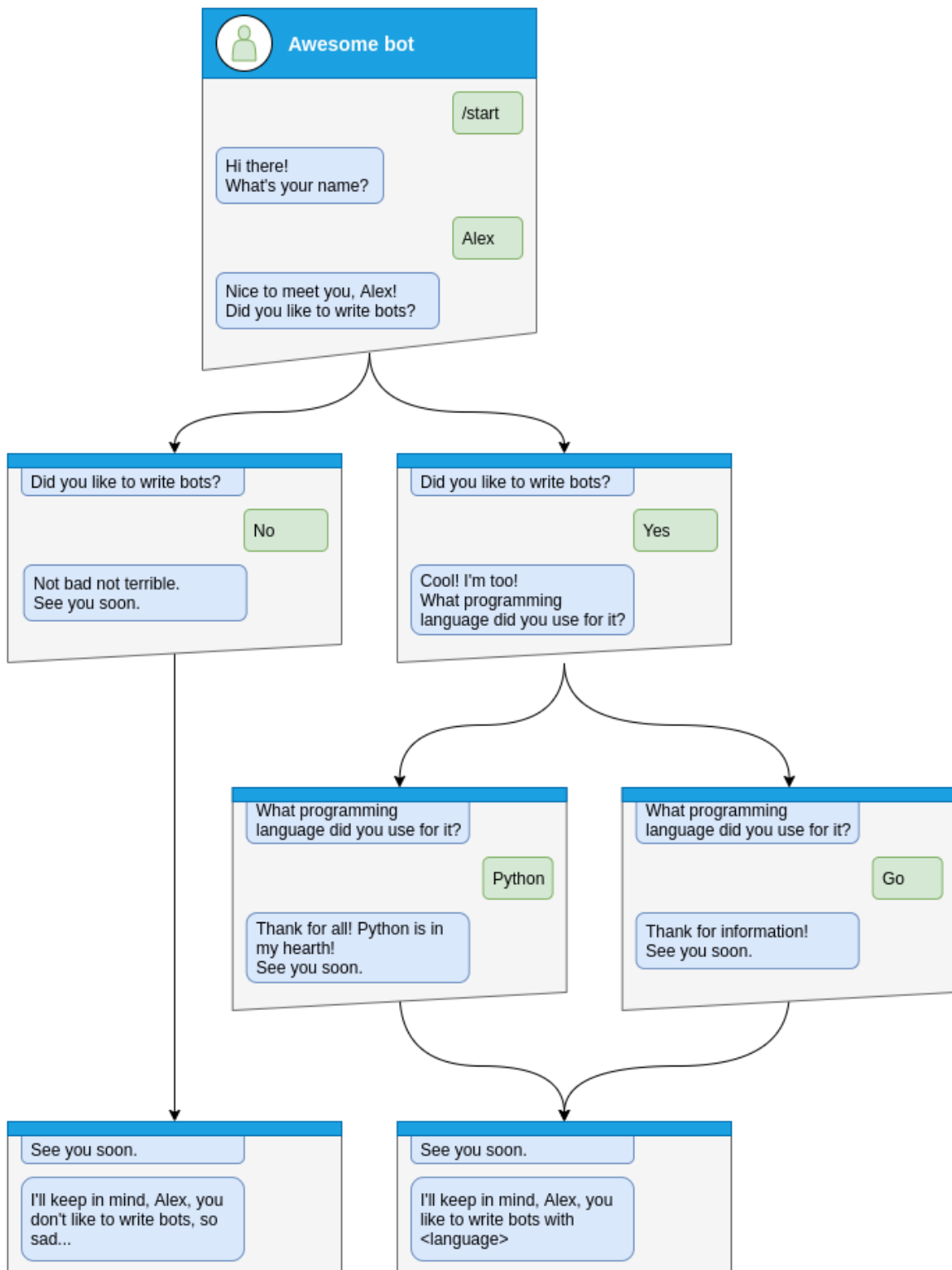
It is an abstract machine that can be in exactly one of a finite number of states at any given time. The FSM can change from one state to another in response to some inputs; the change from one state to another is called a transition.

An FSM is defined by a list of its states, its initial state, and the inputs that trigger each transition.

Source: [Wikipedia](#)

### Usage example

Not all functionality of the bot can be implemented as single handler, for example you will need to collect some data from user in separated steps you will need to use FSM.



Let's see how to do that step-by-step

## Step by step

Before handle any states you will need to specify what kind of states you want to handle

```
class Form(StatesGroup):  
    name = State()  
    like_bots = State()  
    language = State()
```

And then write handler for each state separately from the start of dialog

Here is dialog can be started only via command `/start`, so lets handle it and make transition user to state `Form.name`

```
@form_router.message(CommandStart())  
async def command_start(message: Message, state: FSMContext) -> None:  
    await state.set_state(Form.name)  
    await message.answer(  
        "Hi there! What's your name?",  
        reply_markup=ReplyKeyboardRemove(),  
    )
```

After that you will need to save some data to the storage and make transition to next step.

```
@form_router.message(Form.name)  
async def process_name(message: Message, state: FSMContext) -> None:  
    await state.update_data(name=message.text)  
    await state.set_state(Form.like_bots)  
    await message.answer(  
        f"Nice to meet you, {html.quote(message.text)}!\nDid you like to write bots?",  
        reply_markup=ReplyKeyboardMarkup(  
            keyboard=[  
                [  
                    KeyboardButton(text="Yes"),  
                    KeyboardButton(text="No"),  
                ]  
            ],  
            resize_keyboard=True,  
        ),  
    )
```

At the next steps user can make different answers, it can be *yes*, *no* or any other

Handle yes and soon we need to handle `Form.language` state

```
@form_router.message(Form.like_bots, F.text.casefold() == "yes")  
async def process_like_write_bots(message: Message, state: FSMContext) -> None:  
    await state.set_state(Form.language)  
  
    await message.reply(  
        "Cool! I'm too!\nWhat programming language did you use for it?",  
        reply_markup=ReplyKeyboardRemove(),  
    )
```

Handle no



```
@form_router.message(Form.like_bots, F.text.casefold() == "no")
async def process_dont_like_write_bots(message: Message, state: FSMContext) -> None:
    data = await state.get_data()
    await state.clear()
    await message.answer(
        "Not bad not terrible.\nSee you soon.",
        reply_markup=ReplyKeyboardRemove(),
    )
    await show_summary(message=message, data=data, positive=False)
```

And handle any other answers

```
@form_router.message(Form.like_bots)
async def process_unknown_write_bots(message: Message) -> None:
    await message.reply("I don't understand you :(")
```

All possible cases of `like_bots` step was covered, let's implement finally step

```
@form_router.message(Form.language)
async def process_language(message: Message, state: FSMContext) -> None:
    data = await state.update_data(language=message.text)
    await state.clear()

    if message.text.casefold() == "python":
        await message.reply(
            "Python, you say? That's the language that makes my circuits light up! "
        )

    await show_summary(message=message, data=data)
```

```
async def show_summary(message: Message, data: Dict[str, Any], positive: bool = True) ->
↳None:
    name = data["name"]
    language = data.get("language", "<something unexpected>")
    text = f"I'll keep in mind that, {html.quote(name)}, "
    text += (
        f"you like to write bots with {html.quote(language)}."
        if positive
        else "you don't like to write bots, so sad..."
    )
    await message.answer(text=text, reply_markup=ReplyKeyboardRemove())
```

And now you have covered all steps from the image, but you can make possibility to cancel conversation, lets do that via command or text

```
@form_router.message(Command("cancel"))
@form_router.message(F.text.casefold() == "cancel")
async def cancel_handler(message: Message, state: FSMContext) -> None:
    """
    Allow user to cancel any action
    """
    current_state = await state.get_state()
    if current_state is None:
```

(continues on next page)

(continued from previous page)

```

    return

    logging.info("Cancelling state %r", current_state)
    await state.clear()
    await message.answer(
        "Cancelled.",
        reply_markup=ReplyKeyboardRemove(),
    )

```

### Complete example

```

1  import asyncio
2  import logging
3  import sys
4  from os import getenv
5  from typing import Any, Dict
6
7  from aiogram import Bot, Dispatcher, F, Router, html
8  from aiogram.enums import ParseMode
9  from aiogram.filters import Command, CommandStart
10 from aiogram.fsm.context import FSMContext
11 from aiogram.fsm.state import State, StatesGroup
12 from aiogram.types import (
13     KeyboardButton,
14     Message,
15     ReplyKeyboardMarkup,
16     ReplyKeyboardRemove,
17 )
18
19 TOKEN = getenv("BOT_TOKEN")
20
21 form_router = Router()
22
23
24 class Form(StatesGroup):
25     name = State()
26     like_bots = State()
27     language = State()
28
29
30 @form_router.message(CommandStart())
31 async def command_start(message: Message, state: FSMContext) -> None:
32     await state.set_state(Form.name)
33     await message.answer(
34         "Hi there! What's your name?",
35         reply_markup=ReplyKeyboardRemove(),
36     )
37
38
39 @form_router.message(Command("cancel"))

```

(continues on next page)

(continued from previous page)

```

40 @form_router.message(F.text.casefold() == "cancel")
41 async def cancel_handler(message: Message, state: FSMContext) -> None:
42     """
43     Allow user to cancel any action
44     """
45     current_state = await state.get_state()
46     if current_state is None:
47         return
48
49     logging.info("Cancelling state %r", current_state)
50     await state.clear()
51     await message.answer(
52         "Cancelled.",
53         reply_markup=ReplyKeyboardRemove(),
54     )
55
56
57 @form_router.message(Form.name)
58 async def process_name(message: Message, state: FSMContext) -> None:
59     await state.update_data(name=message.text)
60     await state.set_state(Form.like_bots)
61     await message.answer(
62         f"Nice to meet you, {html.quote(message.text)}!\nDid you like to write bots?",
63         reply_markup=ReplyKeyboardMarkup(
64             keyboard=[
65                 [
66                     KeyboardButton(text="Yes"),
67                     KeyboardButton(text="No"),
68                 ]
69             ],
70             resize_keyboard=True,
71         ),
72     )
73
74
75 @form_router.message(Form.like_bots, F.text.casefold() == "no")
76 async def process_dont_like_write_bots(message: Message, state: FSMContext) -> None:
77     data = await state.get_data()
78     await state.clear()
79     await message.answer(
80         "Not bad not terrible.\nSee you soon.",
81         reply_markup=ReplyKeyboardRemove(),
82     )
83     await show_summary(message=message, data=data, positive=False)
84
85
86 @form_router.message(Form.like_bots, F.text.casefold() == "yes")
87 async def process_like_write_bots(message: Message, state: FSMContext) -> None:
88     await state.set_state(Form.language)
89
90     await message.reply(
91         "Cool! I'm too!\nWhat programming language did you use for it?",

```

(continues on next page)

(continued from previous page)

```

92     reply_markup=ReplyKeyboardRemove(),
93 )
94
95
96 @form_router.message(Form.like_bots)
97 async def process_unknown_write_bots(message: Message) -> None:
98     await message.reply("I don't understand you :(")
99
100
101 @form_router.message(Form.language)
102 async def process_language(message: Message, state: FSMContext) -> None:
103     data = await state.update_data(language=message.text)
104     await state.clear()
105
106     if message.text.casefold() == "python":
107         await message.reply(
108             "Python, you say? That's the language that makes my circuits light up! "
109         )
110
111     await show_summary(message=message, data=data)
112
113
114 async def show_summary(message: Message, data: Dict[str, Any], positive: bool = True) ->
115     None:
116     name = data["name"]
117     language = data.get("language", "<something unexpected>")
118     text = f"I'll keep in mind that, {html.quote(name)}, "
119     text += (
120         f"you like to write bots with {html.quote(language)}."
121         if positive
122         else "you don't like to write bots, so sad..."
123     )
124     await message.answer(text=text, reply_markup=ReplyKeyboardRemove())
125
126
127 async def main():
128     bot = Bot(token=TOKEN, parse_mode=ParseMode.HTML)
129     dp = Dispatcher()
130     dp.include_router(form_router)
131
132     await dp.start_polling(bot)
133
134 if __name__ == "__main__":
135     logging.basicConfig(level=logging.INFO, stream=sys.stdout)
136     asyncio.run(main())

```

[Read more](#)

## Storages

### Storages out of the box

#### MemoryStorage

**class** aiogram.fsm.storage.memory.**MemoryStorage**

Default FSM storage, stores all data in dict and loss everything on shutdown

**Warning:** Is not recommended using in production in due to you will lose all data when your bot restarts

`__init__()` → None

#### RedisStorage

**class** aiogram.fsm.storage.redis.**RedisStorage**(*redis: ~redis.asyncio.client.Redis, key\_builder: ~aiogram.fsm.storage.redis.KeyBuilder | None = None, state\_ttl: int | ~datetime.timedelta | None = None, data\_ttl: int | ~datetime.timedelta | None = None, json\_loads: ~typing.Callable[[...], ~typing.Any] = <function loads>, json\_dumps: ~typing.Callable[[...], str] = <function dumps>)*

Redis storage required redis package installed (pip install redis)

`__init__`(*redis: ~redis.asyncio.client.Redis, key\_builder: ~aiogram.fsm.storage.redis.KeyBuilder | None = None, state\_ttl: int | ~datetime.timedelta | None = None, data\_ttl: int | ~datetime.timedelta | None = None, json\_loads: ~typing.Callable[[...], ~typing.Any] = <function loads>, json\_dumps: ~typing.Callable[[...], str] = <function dumps>)* → None

##### Parameters

- **redis** – Instance of Redis connection
- **key\_builder** – builder that helps to convert contextual key to string
- **state\_ttl** – TTL for state records
- **data\_ttl** – TTL for data records

**classmethod** `from_url`(*url: str, connection\_kwargs: Dict[str, Any] | None = None, \*\*kwargs: Any*) → *RedisStorage*

Create an instance of *RedisStorage* with specifying the connection string

##### Parameters

- **url** – for example `redis://user:password@host:port/db`
- **connection\_kwargs** – see redis docs
- **kwargs** – arguments to be passed to *RedisStorage*

##### Returns

an instance of *RedisStorage*

Keys inside storage can be customized via key builders:

**class** aiogram.fsm.storage.redis.**KeyBuilder**

Base class for Redis key builder

**abstract build**(key: *StorageKey*, part: *Literal['data', 'state', 'lock']*) → str

This method should be implemented in subclasses

**Parameters**

- **key** – contextual key
- **part** – part of the record

**Returns**

key to be used in Redis queries

**class** aiogram.fsm.storage.redis.**DefaultKeyBuilder**(\*, prefix: str = 'fsm', separator: str = ':',  
with\_bot\_id: bool = False, with\_destiny: bool = False)

Simple Redis key builder with default prefix.

Generates a colon-joined string with prefix, chat\_id, user\_id, optional bot\_id and optional destiny.

**build**(key: *StorageKey*, part: *Literal['data', 'state', 'lock']*) → str

This method should be implemented in subclasses

**Parameters**

- **key** – contextual key
- **part** – part of the record

**Returns**

key to be used in Redis queries

## Writing own storages

**class** aiogram.fsm.storage.base.**BaseStorage**

Base class for all FSM storages

**abstract async set\_state**(key: *StorageKey*, state: str | State | None = None) → None

Set state for specified key

**Parameters**

- **key** – storage key
- **state** – new state

**abstract async get\_state**(key: *StorageKey*) → str | None

Get key state

**Parameters**

**key** – storage key

**Returns**

current state

**abstract async set\_data**(key: *StorageKey*, data: *Dict[str, Any]*) → None

Write data (replace)

**Parameters**

- **key** – storage key
- **data** – new data

**abstract async get\_data**(key: *StorageKey*) → *Dict[str, Any]*

Get current data for key

**Parameters**

**key** – storage key

**Returns**

current data

**async update\_data**(key: *StorageKey*, data: *Dict[str, Any]*) → *Dict[str, Any]*

Update data in the storage for key (like dict.update)

**Parameters**

- **key** – storage key
- **data** – partial data

**Returns**

new data

**abstract async close**() → None

Close storage (database connection, file or etc.)

## 2.4.8 Middlewares

**aiogram** provides powerful mechanism for customizing event handlers via middlewares.

Middlewares in bot framework seems like Middlewares mechanism in web-frameworks like [aiohttp](#), [fastapi](#), [Django](#) or etc.) with small difference - here is implemented two layers of middlewares (before and after filters).

---

**Note:** Middleware is function that triggered on every event received from Telegram Bot API in many points on processing pipeline.

---

### Base theory

As many books and other literature in internet says:

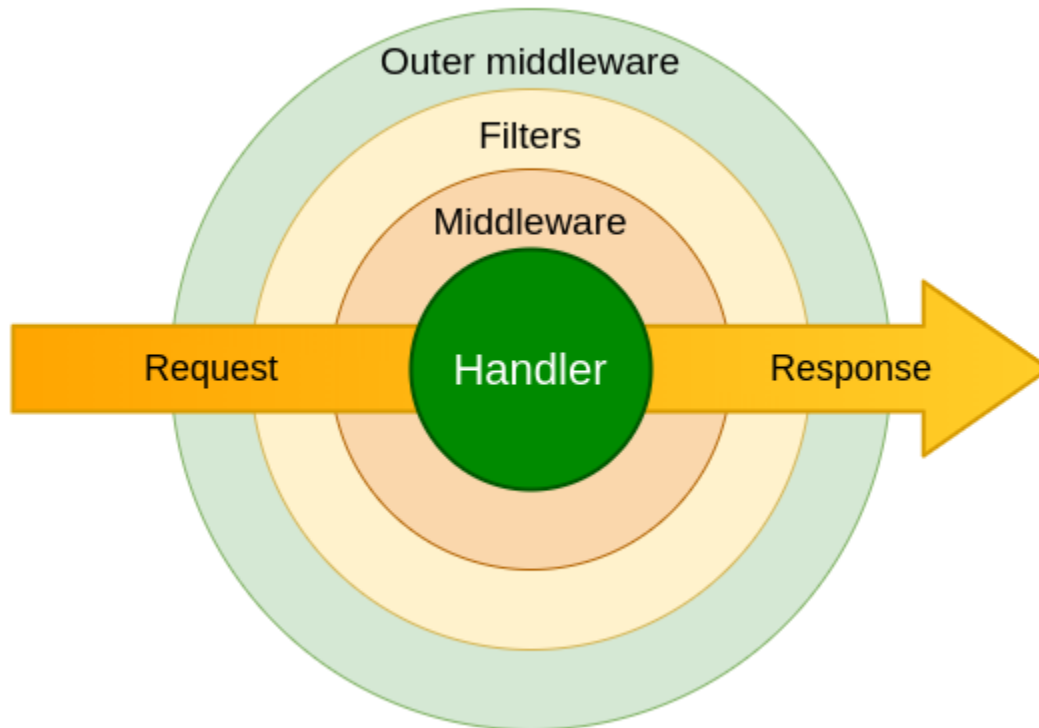
Middleware is reusable software that leverages patterns and frameworks to bridge the gap between the functional requirements of applications and the underlying operating systems, network protocol stacks, and databases.

Middleware can modify, extend or reject processing event in many places of pipeline.

## Basics

Middleware instance can be applied for every type of Telegram Event (Update, Message, etc.) in two places

1. Outer scope - before processing filters (`<router>.<event>.outer_middleware(...)`)
2. Inner scope - after processing filters but before handler (`<router>.<event>.middleware(...)`)



<b>Attention:</b> Middleware should be subclass of <code>BaseMiddleware</code> ( <code>from aiogram import BaseMiddleware</code> ) or any async callable
--

## Arguments specification

```
class aiogram.dispatcher.middlewares.base.BaseMiddleware
```

Bases: ABC

Generic middleware class

```
abstract async __call__(handler: Callable[[TelegramObject, Dict[str, Any]], Awaitable[Any]], event: TelegramObject, data: Dict[str, Any]) → Any
```

Execute middleware

### Parameters

- **handler** – Wrapped handler in middlewares chain
- **event** – Incoming event (Subclass of `aiogram.types.base.TelegramObject`)



- **data** – Contextual data. Will be mapped to handler arguments

**Returns**

Any

**Examples**

**Danger:** Middleware should always call `await handler(event, data)` to propagate event for next middleware/handler

**Class-based**

```
from aiogram import BaseMiddleware
from aiogram.types import Message

class CounterMiddleware(BaseMiddleware):
    def __init__(self) -> None:
        self.counter = 0

    async def __call__(
        self,
        handler: Callable[[Message, Dict[str, Any]], Awaitable[Any]],
        event: Message,
        data: Dict[str, Any]
    ) -> Any:
        self.counter += 1
        data['counter'] = self.counter
        return await handler(event, data)
```

and then

```
router = Router()
router.message.middleware(CounterMiddleware())
```

**Function-based**

```
@dispatcher.update.outer_middleware()
async def database_transaction_middleware(
    handler: Callable[[Update, Dict[str, Any]], Awaitable[Any]],
    event: Update,
    data: Dict[str, Any]
) -> Any:
    async with database.transaction():
        return await handler(event, data)
```

## Facts

1. Middlewares from outer scope will be called on every incoming event
2. Middlewares from inner scope will be called only when filters pass
3. Inner middlewares is always calls for `aiogram.types.update.Update` event type in due to all incoming updates going to specific event type handler through built in update handler

## 2.4.9 Errors

### Handling errors

Is recommended way that you should use errors inside handlers using try-except block, but in common cases you can use global errors handler at router or dispatcher level.

If you specify errors handler for router - it will be used for all handlers inside this router.

If you specify errors handler for dispatcher - it will be used for all handlers inside all routers.

```
@router.error(ExceptionTypeFilter(MyCustomException), F.update.message.as_("message"))
async def handle_my_custom_exception(event: ErrorEvent, message: Message):
    # do something with error
    await message.answer("Oops, something went wrong!")

@router.error()
async def error_handler(event: ErrorEvent):
    logger.critical("Critical error caused by %s", event.exception, exc_info=True)
    # do something with error
    ...
```

### ErrorEvent

```
class aiogram.types.error_event.ErrorEvent(*, update: Update, exception: Exception, **extra_data: Any)
```

Internal event, should be used to receive errors while processing Updates from Telegram

Source: <https://core.telegram.org/bots/api#error-event>

**update:** `Update`

Received update

**exception:** `Exception`

Exception

## Error types

**exception** aiogram.exceptions.**AiogramError**

Base exception for all aiogram errors.

**exception** aiogram.exceptions.**DetailedAiogramError**(*message: str*)

Base exception for all aiogram errors with detailed message.

**exception** aiogram.exceptions.**CallbackAnswerException**

Exception for callback answer.

**exception** aiogram.exceptions.**UnsupportedKeywordArgument**(*message: str*)

Exception raised when a keyword argument is passed as filter.

**exception** aiogram.exceptions.**TelegramAPIError**(*method: TelegramMethod, message: str*)

Base exception for all Telegram API errors.

**exception** aiogram.exceptions.**TelegramNetworkError**(*method: TelegramMethod, message: str*)

Base exception for all Telegram network errors.

**exception** aiogram.exceptions.**TelegramRetryAfter**(*method: TelegramMethod, message: str, retry\_after: int*)

Exception raised when flood control exceeds.

**exception** aiogram.exceptions.**TelegramMigrateToChat**(*method: TelegramMethod, message: str, migrate\_to\_chat\_id: int*)

Exception raised when chat has been migrated to a supergroup.

**exception** aiogram.exceptions.**TelegramBadRequest**(*method: TelegramMethod, message: str*)

Exception raised when request is malformed.

**exception** aiogram.exceptions.**TelegramNotFound**(*method: TelegramMethod, message: str*)

Exception raised when chat, message, user, etc. not found.

**exception** aiogram.exceptions.**TelegramConflictError**(*method: TelegramMethod, message: str*)

Exception raised when bot token is already used by another application in polling mode.

**exception** aiogram.exceptions.**TelegramUnauthorizedError**(*method: TelegramMethod, message: str*)

Exception raised when bot token is invalid.

**exception** aiogram.exceptions.**TelegramForbiddenError**(*method: TelegramMethod, message: str*)

Exception raised when bot is kicked from chat or etc.

**exception** aiogram.exceptions.**TelegramServerError**(*method: TelegramMethod, message: str*)

Exception raised when Telegram server returns 5xx error.

**exception** aiogram.exceptions.**RestartingTelegram**(*method: TelegramMethod, message: str*)

Exception raised when Telegram server is restarting.

It seems like this error is not used by Telegram anymore, but it's still here for backward compatibility.

**Currently, you should expect that Telegram can raise RetryAfter (with timeout 5 seconds) error instead of this one.**

**exception** aiogram.exceptions.**TelegramEntityTooLarge**(*method: TelegramMethod, message: str*)

Exception raised when you are trying to send a file that is too large.

**exception** aiogram.exceptions.**ClientDecodeError**(*message: str, original: Exception, data: Any*)

Exception raised when client can't decode response. (Malformed response, etc.)

## 2.4.10 Flags

Flags is a markers for handlers that can be used in *middlewares* or special *utilities* to make classification of the handlers. Flags can be added to the handler via *decorators*, *handlers registration* or *filters*.

### Via decorators

For example mark handler with *chat\_action* flag

```
from aiogram import flags

@flags.chat_action
async def my_handler(message: Message)
```

Or just for rate-limit or something else

```
from aiogram import flags

@flags.rate_limit(rate=2, key="something")
async def my_handler(message: Message)
```

### Via handler registration method

```
@router.message(..., flags={'chat_action': 'typing', 'rate_limit': {'rate': 5}})
```

### Via filters

```
class Command(Filter):
    ...

    def update_handler_flags(self, flags: Dict[str, Any]) -> None:
        commands = flags.setdefault("commands", [])
        commands.append(self)
```

### Use in middlewares

`aiogram.dispatcher.flags.check_flags(handler: HandlerObject | Dict[str, Any], magic: MagicFilter) -> Any`

Check flags via magic filter

#### Parameters

- **handler** – handler object or data
- **magic** – instance of the magic

#### Returns

the result of magic filter check

`aiogram.dispatcher.flags.extract_flags(handler: HandlerObject | Dict[str, Any]) → Dict[str, Any]`

Extract flags from handler or middleware context data

**Parameters**

**handler** – handler object or data

**Returns**

dictionary with all handler flags

`aiogram.dispatcher.flags.get_flag(handler: HandlerObject | Dict[str, Any], name: str, *, default: Any | None = None) → Any`

Get flag by name

**Parameters**

- **handler** – handler object or data
- **name** – name of the flag
- **default** – default value (None)

**Returns**

value of the flag or default

## Example in middlewares

```
async def my_middleware(handler, event, data):
    typing = get_flag(data, "typing") # Check that handler marked with `typing` flag
    if not typing:
        return await handler(event, data)

    async with ChatActionSender.typing(chat_id=event.chat.id):
        return await handler(event, data)
```

## Use in utilities

For example you can collect all registered commands with handler description and then it can be used for generating commands help

```
def collect_commands(router: Router) -> Generator[Tuple[Command, str], None, None]:
    for handler in router.message.handlers:
        if "commands" not in handler.flags: # ignore all handler without commands
            continue
        # the Command filter adds the flag with list of commands attached to the handler
        for command in handler.flags["commands"]:
            yield command, handler.callback.__doc__ or ""
    # Recursively extract commands from nested routers
    for sub_router in router.sub_routers:
        yield from collect_commands(sub_router)
```

### 2.4.11 Class based handlers

A handler is a async callable which takes a event with contextual data and returns a response.

In **aiogram** it can be more than just an async function, these allow you to use classes which can be used as Telegram event handlers to structure your event handlers and reuse code by harnessing inheritance and mixins.

There are some base class based handlers what you need to use in your own handlers:

#### BaseHandler

Base handler is generic abstract class and should be used in all other class-based handlers.

Import: `from aiogram.handler import BaseHandler`

By default you will need to override only method `async def handle(self) -> Any: ...`

This class is also have an default initializer and you don't need to change it. Initializer accepts current event and all contextual data and which can be accessed from the handler through attributes: `event: TelegramEvent` and `data: Dict[Any, str]`

If instance of the bot is specified in context data or current context it can be accessed through `bot` class attribute.

#### Example

```
class MyHandler(BaseHandler[Message]):
    async def handle(self) -> Any:
        await self.event.answer("Hello!")
```

#### CallbackQueryHandler

`class aiogram.handlers.callback_query.CallbackQueryHandler(event: T, **kwargs: Any)`

There is base class for callback query handlers.

Example:

```
from aiogram.handlers import CallbackQueryHandler

...

@router.callback_query()
class MyHandler(CallbackQueryHandler):
    async def handle(self) -> Any: ...
```

**property from\_user:** *User*

Is alias for `event.from_user`

**property message:** *Message* | None

Is alias for `event.message`

**property callback\_data:** str | None

Is alias for `event.data`

## ChosenInlineResultHandler

There is base class for chosen inline result handlers.

### Simple usage

```
from aiogram.handlers import ChosenInlineResultHandler

...

@router.chosen_inline_result()
class MyHandler(ChosenInlineResultHandler):
    async def handle(self) -> Any: ...
```

### Extension

This base handler is subclass of *BaseHandler* with some extensions:

- `self.chat` is alias for `self.event.chat`
- `self.from_user` is alias for `self.event.from_user`

## ErrorHandler

There is base class for error handlers.

### Simple usage

```
from aiogram.handlers import ErrorHandler

...

@router.errors()
class MyHandler(ErrorHandler):
    async def handle(self) -> Any:
        log.exception(
            "Cause unexpected exception %s: %s",
            self.exception_name,
            self.exception_message
        )
```

## Extension

This base handler is subclass of *BaseHandler* with some extensions:

- `self.exception_name` is alias for `self.event.__class__.__name__`
- `self.exception_message` is alias for `str(self.event)`

## InlineQueryHandler

There is base class for inline query handlers.

### Simple usage

```
from aiogram.handlers import InlineQueryHandler

...

@router.inline_query()
class MyHandler(InlineQueryHandler):
    async def handle(self) -> Any: ...
```

## Extension

This base handler is subclass of *BaseHandler* with some extensions:

- `self.chat` is alias for `self.event.chat`
- `self.query` is alias for `self.event.query`

## MessageHandler

There is base class for message handlers.

### Simple usage

```
from aiogram.handlers import MessageHandler

...

@router.message()
class MyHandler(MessageHandler):
    async def handle(self) -> Any:
        return SendMessage(chat_id=self.chat.id, text="PASS")
```



## Extension

This base handler is subclass of [BaseHandler](basics.md#basehandler) with some extensions:

- `self.chat` is alias for `self.event.chat`
- `self.from_user` is alias for `self.event.from_user`

## PollHandler

There is base class for poll handlers.

### Simple usage

```
from aiogram.handlers import PollHandler

...

@router.poll()
class MyHandler(PollHandler):
    async def handle(self) -> Any: ...
```

## Extension

This base handler is subclass of *BaseHandler* with some extensions:

- `self.question` is alias for `self.event.question`
- `self.options` is alias for `self.event.options`

## PreCheckoutQueryHandler

There is base class for callback query handlers.

### Simple usage

```
from aiogram.handlers import PreCheckoutQueryHandler

...

@router.pre_checkout_query()
class MyHandler(PreCheckoutQueryHandler):
    async def handle(self) -> Any: ...
```

## Extension

This base handler is subclass of *BaseHandler* with some extensions:

- `self.from_user` is alias for `self.event.from_user`

## ShippingQueryHandler

There is base class for callback query handlers.

### Simple usage

```
from aiogram.handlers import ShippingQueryHandler

...

@router.shipping_query()
class MyHandler(ShippingQueryHandler):
    async def handle(self) -> Any: ...
```

## Extension

This base handler is subclass of *BaseHandler* with some extensions:

- `self.from_user` is alias for `self.event.from_user`

## ChatMemberHandler

There is base class for chat member updated events.

### Simple usage

```
from aiogram.handlers import ChatMemberHandler

...

@router.chat_member()
@router.my_chat_member()
class MyHandler(ChatMemberHandler):
    async def handle(self) -> Any: ...
```

## Extension

This base handler is subclass of *BaseHandler* with some extensions:

- `self.chat` is alias for `self.event.chat`

## 2.5 Utils

### 2.5.1 Keyboard builder

Keyboard builder helps to dynamically generate markup.

**Note:** Note that if you have static markup, it's best to define it explicitly rather than using builder, but if you have dynamic markup configuration, feel free to use builder as you wish.

#### Usage example

For example you want to generate inline keyboard with 10 buttons

```
builder = InlineKeyboardBuilder()

for index in range(1, 11):
    builder.button(text=f"Set {index}", callback_data=f"set:{index}")
```

then adjust this buttons to some grid, for example first line will have 3 buttons, the next lines will have 2 buttons

```
builder.adjust(3, 2)
```

also you can attach another builder to this one

```
another_builder = InlineKeyboardBuilder(...)... # Another builder with some buttons
builder.attach(another_builder)
```

or you can attach some already generated markup

```
markup = InlineKeyboardMarkup(inline_keyboard=[...]) # Some markup
builder.attach(InlineKeyboardBuilder.from_markup(markup))
```

and finally you can export this markup to use it in your message

```
await message.answer("Some text here", reply_markup=builder.as_markup())
```

Reply keyboard builder has the same interface

**Warning:** Note that you can't attach reply keyboard builder to inline keyboard builder and vice versa

## Inline Keyboard

**class** aiogram.utils.keyboard.**InlineKeyboardBuilder**(markup: List[List[InlineKeyboardButton]] | None = None)

Inline keyboard builder inherits all methods from generic builder

**button**(text: str, url: str | None = None, login\_url: LoginUrl | None = None, callback\_data: str | CallbackData | None = None, switch\_inline\_query: str | None = None, switch\_inline\_query\_current\_chat: str | None = None, callback\_game: CallbackGame | None = None, pay: bool | None = None, \*\*kwargs: Any) → aiogram.utils.keyboard.InlineKeyboardBuilder

Add new inline button to markup

**as\_markup**() → aiogram.types.inline\_keyboard\_markup.InlineKeyboardMarkup

Construct an InlineKeyboardMarkup

**\_\_init\_\_**(markup: List[List[InlineKeyboardButton]] | None = None) → None

**add**(\*buttons: ButtonType) → KeyboardBuilder[ButtonType]

Add one or many buttons to markup.

### Parameters

**buttons** –

### Returns

**adjust**(\*sizes: int, repeat: bool = False) → KeyboardBuilder[ButtonType]

Adjust previously added buttons to specific row sizes.

By default, when the sum of passed sizes is lower than buttons count the last one size will be used for tail of the markup. If repeat=True is passed - all sizes will be cycled when available more buttons count than all sizes

### Parameters

- **sizes** –
- **repeat** –

### Returns

**property buttons**: Generator[ButtonType, None, None]

Get flatten set of all buttons

### Returns

**copy**() → InlineKeyboardBuilder

Make full copy of current builder with markup

### Returns

**export**() → List[List[ButtonType]]

Export configured markup as list of lists of buttons

```
>>> builder = KeyboardBuilder(button_type=InlineKeyboardButton)
>>> ... # Add buttons to builder
>>> markup = InlineKeyboardMarkup(inline_keyboard=builder.export())
```

### Returns

**classmethod** `from_markup`(*markup*: [InlineKeyboardMarkup](#)) → [InlineKeyboardBuilder](#)

Create builder from existing markup

**Parameters**

**markup** –

**Returns**

**row**(*\*buttons*: [ButtonType](#), *width*: *int* = 8) → [KeyboardBuilder](#)[[ButtonType](#)]

Add row to markup

When too much buttons is passed it will be separated to many rows

**Parameters**

• **buttons** –

• **width** –

**Returns**

## Reply Keyboard

**class** `aiogram.utils.keyboard.ReplyKeyboardBuilder`(*markup*: [List](#)[[List](#)[[KeyboardButton](#)]] | *None* = *None*)

Reply keyboard builder inherits all methods from generic builder

**button**(*text*: *str*, *request\_contact*: *bool* | *None* = *None*, *request\_location*: *bool* | *None* = *None*, *request\_poll*: [KeyboardButtonPollType](#) | *None* = *None*, *\*\*kwargs*: *Any*) → [aiogram.utils.keyboard.ReplyKeyboardBuilder](#)

Add new button to markup

**as\_markup**() → [aiogram.types.reply\\_keyboard\\_markup.ReplyKeyboardMarkup](#)

Construct an [ReplyKeyboardMarkup](#)

**\_\_init\_\_**(*markup*: [List](#)[[List](#)[[KeyboardButton](#)]] | *None* = *None*) → *None*

**add**(*\*buttons*: [ButtonType](#)) → [KeyboardBuilder](#)[[ButtonType](#)]

Add one or many buttons to markup.

**Parameters**

**buttons** –

**Returns**

**adjust**(*\*sizes*: *int*, *repeat*: *bool* = *False*) → [KeyboardBuilder](#)[[ButtonType](#)]

Adjust previously added buttons to specific row sizes.

By default, when the sum of passed sizes is lower than buttons count the last one size will be used for tail of the markup. If `repeat=True` is passed - all sizes will be cycled when available more buttons count than all sizes

**Parameters**

• **sizes** –

• **repeat** –

**Returns**

**property buttons:** `Generator[ButtonType, None, None]`

Get flatten set of all buttons

**Returns**

**copy()** → *ReplyKeyboardBuilder*

Make full copy of current builder with markup

**Returns**

**export()** → `List[List[ButtonType]]`

Export configured markup as list of lists of buttons

```
>>> builder = KeyboardBuilder(button_type=InlineKeyboardButton)
>>> ... # Add buttons to builder
>>> markup = InlineKeyboardMarkup(inline_keyboard=builder.export())
```

**Returns**

**classmethod from\_markup**(*markup: ReplyKeyboardMarkup*) → *ReplyKeyboardBuilder*

Create builder from existing markup

**Parameters**

**markup** –

**Returns**

**row**(\**buttons: ButtonType*, *width: int = 8*) → `KeyboardBuilder[ButtonType]`

Add row to markup

When too much buttons is passed it will be separated to many rows

**Parameters**

• **buttons** –

• **width** –

**Returns**

## 2.5.2 Translation

In order to make you bot translatable you have to add a minimal number of hooks to your Python code.

These hooks are called translation strings.

The aiogram translation utils is build on top of [GNU gettext Python module](#) and [Babel library](#).

### Installation

Babel is required to make simple way to extract translation strings from your code

Can be installed from pip directly:

```
pip install Babel
```

or as *aiogram* extra dependency:

```
pip install aiogram[i18n]
```

### Make messages translatable

In order to gettext need to know what the strings should be translated you will need to write translation strings.

For example:

```
from aiogram import html
from aiogram.utils.i18n import gettext as _

async def my_handler(message: Message) -> None:
    await message.answer(
        _("Hello, {name}!").format(
            name=html.quote(message.from_user.full_name)
        )
    )
```

**Danger:** f-strings can't be used as translations string because any dynamic variables should be added to message after getting translated message

Also if you want to use translated string in keyword- or magic- filters you will need to use lazy gettext calls:

```
from aiogram import F
from aiogram.utils.i18n import lazy_gettext as __

@router.message(F.text == __("My menu entry"))
...
```

**Danger:** Lazy gettext calls should always be used when the current language is not know at the moment

**Danger:** Lazy gettext can't be used as value for API methods or any Telegram Object (like [aiogram.types.inline\\_keyboard\\_button.InlineKeyboardButton](#) or etc.)

### Configuring engine

After you messages is already done to use gettext your bot should know how to detect user language

On top of your application the instance of `aiogram.utils.i18n.I18n` should be created

```
i18n = I18n(path="locales", default_locale="en", domain="messages")
```

After that you will need to choose one of builtin I18n middleware or write your own.

Builtin middlewares:

### SimpleI18nMiddleware

```
class aiogram.utils.i18n.middleware.SimpleI18nMiddleware(i18n: I18n, i18n_key: str | None = 'i18n',
                                                         middleware_key: str =
                                                         'i18n_middleware')
```

Simple I18n middleware.

Chooses language code from the User object received in event

```
__init__(i18n: I18n, i18n_key: str | None = 'i18n', middleware_key: str = 'i18n_middleware') → None
```

Create an instance of middleware

#### Parameters

- **i18n** – instance of I18n
- **i18n\_key** – context key for I18n instance
- **middleware\_key** – context key for this middleware

### ConstI18nMiddleware

```
class aiogram.utils.i18n.middleware.ConstI18nMiddleware(locale: str, i18n: I18n, i18n_key: str | None
                                                         = 'i18n', middleware_key: str =
                                                         'i18n_middleware')
```

Const middleware chooses statically defined locale

```
__init__(locale: str, i18n: I18n, i18n_key: str | None = 'i18n', middleware_key: str = 'i18n_middleware')
→ None
```

Create an instance of middleware

#### Parameters

- **i18n** – instance of I18n
- **i18n\_key** – context key for I18n instance
- **middleware\_key** – context key for this middleware

### FSMI18nMiddleware

```
class aiogram.utils.i18n.middleware.FSMI18nMiddleware(i18n: I18n, key: str = 'locale', i18n_key: str |
                                                         None = 'i18n', middleware_key: str =
                                                         'i18n_middleware')
```

This middleware stores locale in the FSM storage

```
__init__(i18n: I18n, key: str = 'locale', i18n_key: str | None = 'i18n', middleware_key: str =
'i18n_middleware') → None
```

Create an instance of middleware

#### Parameters

- **i18n** – instance of I18n
- **i18n\_key** – context key for I18n instance
- **middleware\_key** – context key for this middleware



**async set\_locale**(*state: FSMContext, locale: str*) → None

Write new locale to the storage

#### Parameters

- **state** – instance of FSMContext
- **locale** – new locale

## I18nMiddleware

or define you own based on abstract I18nMiddleware middleware:

```
class aiogram.utils.i18n.middleware.I18nMiddleware(i18n: I18n, i18n_key: str | None = 'i18n',  
                                                middleware_key: str = 'i18n_middleware')
```

Abstract I18n middleware.

```
__init__(i18n: I18n, i18n_key: str | None = 'i18n', middleware_key: str = 'i18n_middleware') → None
```

Create an instance of middleware

#### Parameters

- **i18n** – instance of I18n
- **i18n\_key** – context key for I18n instance
- **middleware\_key** – context key for this middleware

```
abstract async get_locale(event: TelegramObject, data: Dict[str, Any]) → str
```

Detect current user locale based on event and context.

**This method must be defined in child classes**

#### Parameters

- **event** –
- **data** –

#### Returns

```
setup(router: Router, exclude: Set[str] | None = None) → BaseMiddleware
```

Register middleware for all events in the Router

#### Parameters

- **router** –
- **exclude** –

#### Returns

## Deal with Babel

### Step 1 Extract messages

```
pybabel extract --input-dirs=. -o locales/messages.pot
```

Here is `--input-dirs=.` - path to code and the `locales/messages.pot` is template where messages will be extracted and `messages` is translation domain.

---

**Note:** Some useful options:

- Extract texts with pluralization support `-k __:1,2`
  - Add comments for translators, you can use another tag if you want (TR) `--add-comments=NOTE`
  - Disable comments with string location in code `--no-location`
  - Set project name `--project=MySuperBot`
  - Set version `--version=2.2`
- 

### Step 2: Init language

```
pybabel init -i locales/messages.pot -d locales -D messages -l en
```

- `-i locales/messages.pot` - pre-generated template
- `-d locales` - translations directory
- `-D messages` - translations domain
- `-l en` - language. Can be changed to any other valid language code (For example `-l uk` for ukrainian language)

### Step 3: Translate texts

To open `.po` file you can use basic text editor or any PO editor, e.g. [Poedit](#)

Just open the file named `locales/{language}/LC_MESSAGES/messages.po` and write translations

### Step 4: Compile translations

```
pybabel compile -d locales -D messages
```

## Step 5: Updating messages

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

- Step 5.1: regenerate pot file: command from step 1
- **Step 5.2: update po files**

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

- Step 5.3: update your translations: location and tools you know from step 3
- Step 5.4: compile mo files: command from step 4

## 2.5.3 Chat action sender

### Sender

```
class aiogram.utils.chat_action.ChatActionSender(*, bot: Bot, chat_id: str | int, message_thread_id:
    int | None = None, action: str = 'typing', interval:
    float = 5.0, initial_sleep: float = 0.0)
```

This utility helps to automatically send chat action until long actions is done to take acknowledge bot users the bot is doing something and not crashed.

Provides simply to use context manager.

Technically sender start background task with infinity loop which works until action will be finished and sends the [chat action](#) every 5 seconds.

```
__init__(*, bot: Bot, chat_id: str | int, message_thread_id: int | None = None, action: str = 'typing',
    interval: float = 5.0, initial_sleep: float = 0.0) → None
```

#### Parameters

- **bot** – instance of the bot
- **chat\_id** – target chat id
- **action** – chat action type
- **interval** – interval between iterations
- **initial\_sleep** – sleep before first iteration

```
classmethod choose_sticker(chat_id: int | str, bot: Bot, interval: float = 5.0, initial_sleep: float = 0.0)
    → ChatActionSender
```

Create instance of the sender with *choose\_sticker* action

```
classmethod find_location(chat_id: int | str, bot: Bot, interval: float = 5.0, initial_sleep: float = 0.0)
    → ChatActionSender
```

Create instance of the sender with *find\_location* action

```
classmethod record_video(chat_id: int | str, bot: Bot, interval: float = 5.0, initial_sleep: float = 0.0) →
    ChatActionSender
```

Create instance of the sender with *record\_video* action

```
classmethod record_video_note(chat_id: int | str, bot: Bot, interval: float = 5.0, initial_sleep: float =
    0.0) → ChatActionSender
```

Create instance of the sender with *record\_video\_note* action

**classmethod** `record_voice(chat_id: int | str, bot: Bot, interval: float = 5.0, initial_sleep: float = 0.0) → ChatActionSender`

Create instance of the sender with `record_voice` action

**classmethod** `typing(chat_id: int | str, bot: Bot, interval: float = 5.0, initial_sleep: float = 0.0) → ChatActionSender`

Create instance of the sender with `typing` action

**classmethod** `upload_document(chat_id: int | str, bot: Bot, interval: float = 5.0, initial_sleep: float = 0.0) → ChatActionSender`

Create instance of the sender with `upload_document` action

**classmethod** `upload_photo(chat_id: int | str, bot: Bot, interval: float = 5.0, initial_sleep: float = 0.0) → ChatActionSender`

Create instance of the sender with `upload_photo` action

**classmethod** `upload_video(chat_id: int | str, bot: Bot, interval: float = 5.0, initial_sleep: float = 0.0) → ChatActionSender`

Create instance of the sender with `upload_video` action

**classmethod** `upload_video_note(chat_id: int | str, bot: Bot, interval: float = 5.0, initial_sleep: float = 0.0) → ChatActionSender`

Create instance of the sender with `upload_video_note` action

**classmethod** `upload_voice(chat_id: int | str, bot: Bot, interval: float = 5.0, initial_sleep: float = 0.0) → ChatActionSender`

Create instance of the sender with `upload_voice` action

## Usage

```
async with ChatActionSender.typing(bot=bot, chat_id=message.chat.id):  
    # Do something...  
    # Perform some long calculations  
    await message.answer(result)
```

## Middleware

**class** `aiogram.utils.chat_action.ChatActionMiddleware`

Helps to automatically use chat action sender for all message handlers

## Usage

Before use should be registered for the `message` event

```
<router or dispatcher>.message.middleware(ChatActionMiddleware())
```

After this action all handlers which works longer than `initial_sleep` will produce the `'typing'` chat action.

Also sender can be customized via flags feature for particular handler.

Change only action type:

```
@router.message(...)
@flags.chat_action("sticker")
async def my_handler(message: Message): ...
```

Change sender configuration:

```
@router.message(...)
@flags.chat_action(initial_sleep=2, action="upload_document", interval=3)
async def my_handler(message: Message): ...
```

## 2.5.4 WebApp

Telegram Bot API 6.0 announces a revolution in the development of chatbots using WebApp feature.

You can read more details on it in the official [blog](#) and [documentation](#).

*aiogram* implements simple utils to remove headache with the data validation from Telegram WebApp on the backend side.

### Usage

For example from frontend you will pass `application/x-www-form-urlencoded` POST request with `_auth` field in body and wants to return User info inside response as `application/json`

```
from aiogram.utils.web_app import safe_parse_webapp_init_data
from aiohttp.web_request import Request
from aiohttp.web_response import json_response

async def check_data_handler(request: Request):
    bot: Bot = request.app["bot"]

    data = await request.post() # application/x-www-form-urlencoded
    try:
        data = safe_parse_webapp_init_data(token=bot.token, init_data=data["_auth"])
    except ValueError:
        return json_response({"ok": False, "err": "Unauthorized"}, status=401)
    return json_response({"ok": True, "data": data.user.dict()})
```

### Functions

`aiogram.utils.web_app.check_webapp_signature(token: str, init_data: str) → bool`

Check incoming WebApp init data signature

Source: <https://core.telegram.org/bots/webapps#validating-data-received-via-the-web-app>

#### Parameters

- **token** – bot Token
- **init\_data** – data from frontend to be validated

#### Returns

```
aiogram.utils.web_app.parse_webapp_init_data(init_data: str, *, loads: ~typing.Callable[[...],  
~typing.Any] = <function loads>) → WebAppInitData
```

Parse WebApp init data and return it as WebAppInitData object

This method doesn't make any security check, so you shall not trust to this data, use `safe_parse_webapp_init_data` instead.

#### Parameters

- **init\_data** – data from frontend to be parsed
- **loads** –

#### Returns

```
aiogram.utils.web_app.safe_parse_webapp_init_data(token: str, init_data: str, *, loads:  
~typing.Callable[[...], ~typing.Any] = <function  
loads>) → WebAppInitData
```

Validate raw WebApp init data and return it as WebAppInitData object

Raise `ValueError` when data is invalid

#### Parameters

- **token** – bot token
- **init\_data** – data from frontend to be parsed and validated
- **loads** –

#### Returns

## Types

```
class aiogram.utils.web_app.WebAppInitData(**extra_data: Any)
```

This object contains data that is transferred to the Web App when it is opened. It is empty if the Web App was launched from a keyboard button.

Source: <https://core.telegram.org/bots/webapps#webappinitdata>

**query\_id:** str | None

A unique identifier for the Web App session, required for sending messages via the `answerWebAppQuery` method.

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'defer_build': True, 'extra': 'allow', 'frozen': True, 'populate_by_name': True,  
'use_enum_values': True, 'validate_assignment': True}
```

Configuration for the model, should be a dictionary conforming to `[ConfigDict][pydantic.config.ConfigDict]`.

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'auth_date':  
FieldInfo(annotation=datetime, required=True), 'hash': FieldInfo(annotation=str,  
required=True), 'query_id': FieldInfo(annotation=Union[str, NoneType],  
required=False), 'receiver': FieldInfo(annotation=Union[WebAppUser, NoneType],  
required=False), 'start_param': FieldInfo(annotation=Union[str, NoneType],  
required=False), 'user': FieldInfo(annotation=Union[WebAppUser, NoneType],  
required=False)}
```

Metadata about the fields defined on the model, mapping of field names to `[FieldInfo][pydantic.fields.FieldInfo]`.

This replaces *Model.\_\_fields\_\_* from Pydantic V1.

**user:** *WebAppUser* | None

An object containing data about the current user.

**receiver:** *WebAppUser* | None

An object containing data about the chat partner of the current user in the chat where the bot was launched via the attachment menu. Returned only for Web Apps launched via the attachment menu.

**start\_param:** str | None

The value of the startattach parameter, passed via link. Only returned for Web Apps when launched from the attachment menu via link. The value of the start\_param parameter will also be passed in the GET-parameter tgWebAppStartParam, so the Web App can load the correct interface right away.

**auth\_date:** datetime

Unix time when the form was opened.

**hash:** str

A hash of all passed parameters, which the bot server can use to check their validity.

**class** aiogram.utils.web\_app.WebAppUser(\*\*extra\_data: Any)

This object contains the data of the Web App user.

Source: <https://core.telegram.org/bots/webapps#webappuser>

**id:** int

A unique identifier for the user or bot. This number may have more than 32 significant bits and some programming languages may have difficulty/silent defects in interpreting it. It has at most 52 significant bits, so a 64-bit integer or a double-precision float type is safe for storing this identifier.

**is\_bot:** bool | None

True, if this user is a bot. Returns in the receiver field only.

**first\_name:** str

First name of the user or bot.

**last\_name:** str | None

Last name of the user or bot.

**username:** str | None

Username of the user or bot.

**language\_code:** str | None

IETF language tag of the user's language. Returns in user field only.

**photo\_url:** str | None

URL of the user's profile photo. The photo can be in .jpeg or .svg formats. Only returned for Web Apps launched from the attachment menu.

**model\_config:** ClassVar[ConfigDict] = {'arbitrary\_types\_allowed': True, 'defer\_build': True, 'extra': 'allow', 'frozen': True, 'populate\_by\_name': True, 'use\_enum\_values': True, 'validate\_assignment': True}

Configuration for the model, should be a dictionary conforming to [ConfigDict][pydantic.config.ConfigDict].

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'first_name':
FieldInfo(annotation=str, required=True), 'id': FieldInfo(annotation=int,
required=True), 'is_bot': FieldInfo(annotation=Union[bool, NoneType],
required=False), 'language_code': FieldInfo(annotation=Union[str, NoneType],
required=False), 'last_name': FieldInfo(annotation=Union[str, NoneType],
required=False), 'photo_url': FieldInfo(annotation=Union[str, NoneType],
required=False), 'username': FieldInfo(annotation=Union[str, NoneType],
required=False)}
```

Metadata about the fields defined on the model, mapping of field names to `[FieldInfo][pydantic.fields.FieldInfo]`.

This replaces `Model.__fields__` from Pydantic V1.

## 2.5.5 Callback answer

Helper for callback query handlers, can be useful in bots with a lot of callback handlers to automatically take answer to all requests.

### Simple usage

For use, it is enough to register the inner middleware `aiogram.utils.callback_answer.CallbackAnswerMiddleware` in dispatcher or specific router:

```
dispatcher.callback_query.middleware(CallbackAnswerMiddleware())
```

After that all handled callback queries will be answered automatically after processing the handler.

### Advanced usage

In some cases you need to have some non-standard response parameters, this can be done in several ways:

#### Global defaults

Change default parameters while initializing middleware, for example change answer to *pre* mode and text “OK”:

```
dispatcher.callback_query.middleware(CallbackAnswerMiddleware(pre=True, text="OK"))
```

Look at `aiogram.utils.callback_answer.CallbackAnswerMiddleware` to get all available parameters

### Handler specific

By using *flags* you can change the behavior for specific handler

```
@router.callback_query(<filters>)
@flags.callback_answer(text="Thanks", cache_time=30)
async def my_handler(query: CallbackQuery):
    ...
```

Flag arguments is the same as in `aiogram.utils.callback_answer.CallbackAnswerMiddleware` with additional one disabled to disable answer.



## A special case

It is not always correct to answer the same in every case, so there is an option to change the answer inside the handler. You can get an instance of `aiogram.utils.callback_answer.CallbackAnswer` object inside handler and change whatever you want.

**Danger:** Note that is impossible to change callback answer attributes when you use `pre=True` mode.

```
@router.callback_query(<filters>)
async def my_handler(query: CallbackQuery, callback_answer: CallbackAnswer):
    ...
    if <everything is ok>:
        callback_answer.text = "All is ok"
    else:
        callback_answer.text = "Something wrong"
        callback_answer.cache_time = 10
```

## Combine that all at once

For example you want to answer in most of cases before handler with text "" but at some cases need to answer after the handler with custom text, so you can do it:

```
dispatcher.callback_query.middleware(CallbackAnswerMiddleware(pre=True, text=""))

@router.callback_query(<filters>)
@flags.callback_answer(pre=False, cache_time=30)
async def my_handler(query: CallbackQuery):
    ...
    if <everything is ok>:
        callback_answer.text = "All is ok"
```

## Description of objects

```
class aiogram.utils.callback_answer.CallbackAnswerMiddleware(pre: bool = False, text: str | None =
    None, show_alert: bool | None =
    None, url: str | None = None,
    cache_time: int | None = None)
```

Bases: `BaseMiddleware`

```
__init__(pre: bool = False, text: str | None = None, show_alert: bool | None = None, url: str | None = None,
    cache_time: int | None = None) → None
```

Inner middleware for callback query handlers, can be useful in bots with a lot of callback handlers to automatically take answer to all requests

### Parameters

- **pre** – send answer before execute handler
- **text** – answer with text
- **show\_alert** – show alert

- **url** – game url
- **cache\_time** – cache answer for some time

```
class aiogram.utils.callback_answer.CallbackAnswer(answered: bool, disabled: bool = False, text: str | None = None, show_alert: bool | None = None, url: str | None = None, cache_time: int | None = None)
```

Bases: object

```
__init__(answered: bool, disabled: bool = False, text: str | None = None, show_alert: bool | None = None, url: str | None = None, cache_time: int | None = None) → None
```

Callback answer configuration

#### Parameters

- **answered** – this request is already answered by middleware
- **disabled** – answer will not be performed
- **text** – answer with text
- **show\_alert** – show alert
- **url** – game url
- **cache\_time** – cache answer for some time

```
disable() → None
```

Deactivate answering for this handler

```
property disabled: bool
```

Indicates that automatic answer is disabled in this handler

```
property answered: bool
```

Indicates that request is already answered by middleware

```
property text: str | None
```

Response text :return:

```
property show_alert: bool | None
```

Whether to display an alert

```
property url: str | None
```

Game url

```
property cache_time: int | None
```

Response cache time

## 2.5.6 Formatting

Make your message formatting flexible and simple

This instrument works on top of Message entities instead of using HTML or Markdown markups, you can easily construct your message and sent it to the Telegram without the need to remember tag parity (opening and closing) or escaping user input.

## Usage

### Basic scenario

Construct your message and send it to the Telegram.

```
content = Text("Hello, ", Bold(message.from_user.full_name), "!")
await message.answer(**content.as_kwargs())
```

Is the same as the next example, but without usage markup

```
await message.answer(
    text=f"Hello, <b>{html.quote(message.from_user.full_name)}!",
    parse_mode=ParseMode.HTML
)
```

Literally when you execute `as_kwargs` method the `Text` object is converted into text `Hello, Alex!` with entities list `[MessageEntity(type='bold', offset=7, length=4)]` and passed into dict which can be used as `**kwargs` in API call.

The complete list of elements is listed [on this page below](#).

### Advanced scenario

On top of base elements can be implemented content rendering structures, so, out of the box aiogram has a few already implemented functions that helps you to format your messages:

`aiogram.utils.formatting.as_line(*items: Any, end: str = '\n', sep: str = '') → Text`

Wrap multiple nodes into line with `\n` at the end of line.

#### Parameters

- **items** – Text or Any
- **end** – ending of the line, by default is `\n`
- **sep** – separator between items, by default is empty string

#### Returns

Text

`aiogram.utils.formatting.as_list(*items: Any, sep: str = '\n') → Text`

Wrap each element to separated lines

#### Parameters

- **items** –
- **sep** –

#### Returns

`aiogram.utils.formatting.as_marked_list(*items: Any, marker: str = '- ') → Text`

Wrap elements as marked list

#### Parameters

- **items** –
- **marker** – line marker, by default is `'- '`

**Returns**

Text

`aiogram.utils.formatting.as_numbered_list(*items: Any, start: int = 1, fmt: str = '{}. ') → Text`

Wrap elements as numbered list

**Parameters**

- **items** –
- **start** – initial number, by default 1
- **fmt** – number format, by default '{}. '

**Returns**

Text

`aiogram.utils.formatting.as_section(title: Any, *body: Any) → Text`

Wrap elements as simple section, section has title and body

**Parameters**

- **title** –
- **body** –

**Returns**

Text

`aiogram.utils.formatting.as_marked_section(title: Any, *body: Any, marker: str = '- ') → Text`

Wrap elements as section with marked list

**Parameters**

- **title** –
- **body** –
- **marker** –

**Returns**`aiogram.utils.formatting.as_numbered_section(title: Any, *body: Any, start: int = 1, fmt: str = '{}. ') → Text`

Wrap elements as section with numbered list

**Parameters**

- **title** –
- **body** –
- **start** –
- **fmt** –

**Returns**`aiogram.utils.formatting.as_key_value(key: Any, value: Any) → Text`

Wrap elements pair as key-value line. (&lt;b&gt;{key}&lt;/b&gt; {value})

**Parameters**

- **key** –
- **value** –

**Returns**

Text

and lets complete them all:

```
content = as_list(
    as_marked_section(
        Bold("Success:"),
        "Test 1",
        "Test 3",
        "Test 4",
        marker=" ",
    ),
    as_marked_section(
        Bold("Failed:"),
        "Test 2",
        marker=" ",
    ),
    as_marked_section(
        Bold("Summary:"),
        as_key_value("Total", 4),
        as_key_value("Success", 3),
        as_key_value("Failed", 1),
        marker=" ",
    ),
    HashTag("#test"),
    sep="\n\n",
)
```

Will be rendered into:

**Success:**

Test 1

Test 3

Test 4

**Failed:**

Test 2

**Summary:****Total:** 4**Success:** 3**Failed:** 1

#test

Or as HTML:

```
<b>Success:</b>
Test 1
Test 3
Test 4
```

(continues on next page)

(continued from previous page)

```

<b>Failed:</b>
  Test 2

<b>Summary:</b>
  <b>Total:</b> 4
  <b>Success:</b> 3
  <b>Failed:</b> 1

#test

```

## Available methods

**class** aiogram.utils.formatting.Text(\*body: Any, \*\*params: Any)

Bases: Iterable[Any]

Simple text element

**\_\_init\_\_**(\*body: Any, \*\*params: Any) → None

**render**(\*, \_offset: int = 0, \_sort: bool = True, \_collect\_entities: bool = True) → Tuple[str, List[MessageEntity]]

Render elements tree as text with entities list

### Returns

**as\_kwargs**(\*, text\_key: str = 'text', entities\_key: str = 'entities', replace\_parse\_mode: bool = True, parse\_mode\_key: str = 'parse\_mode') → Dict[str, Any]

Render elements tree as keyword arguments for usage in the API call, for example:

```

entities = Text(...)
await message.answer(**entities.as_kwargs())

```

### Parameters

- **text\_key** –
- **entities\_key** –
- **replace\_parse\_mode** –
- **parse\_mode\_key** –

### Returns

**as\_html**() → str

Render elements tree as HTML markup

**as\_markdown**() → str

Render elements tree as MarkdownV2 markup

## Available elements

**class** aiogram.utils.formatting.**Text**(\*body: Any, \*\*params: Any)

Bases: Iterable[Any]

Simple text element

**class** aiogram.utils.formatting.**HashTag**(\*body: Any, \*\*params: Any)

Bases: *Text*

Hashtag element.

**Warning:** The value should always start with '#' symbol

Will be wrapped into *aiogram.types.message\_entity.MessageEntity* with type *aiogram.enums.message\_entity\_type.MessageEntityType.HASHTAG*

**class** aiogram.utils.formatting.**CashTag**(\*body: Any, \*\*params: Any)

Bases: *Text*

Cashtag element.

**Warning:** The value should always start with '\$' symbol

Will be wrapped into *aiogram.types.message\_entity.MessageEntity* with type *aiogram.enums.message\_entity\_type.MessageEntityType.CASHTAG*

**class** aiogram.utils.formatting.**BotCommand**(\*body: Any, \*\*params: Any)

Bases: *Text*

Bot command element.

**Warning:** The value should always start with '/' symbol

Will be wrapped into *aiogram.types.message\_entity.MessageEntity* with type *aiogram.enums.message\_entity\_type.MessageEntityType.BOT\_COMMAND*

**class** aiogram.utils.formatting.**Url**(\*body: Any, \*\*params: Any)

Bases: *Text*

Url element.

Will be wrapped into *aiogram.types.message\_entity.MessageEntity* with type *aiogram.enums.message\_entity\_type.MessageEntityType.URL*

**class** aiogram.utils.formatting.**Email**(\*body: Any, \*\*params: Any)

Bases: *Text*

Email element.

Will be wrapped into *aiogram.types.message\_entity.MessageEntity* with type *aiogram.enums.message\_entity\_type.MessageEntityType.EMAIL*

**class** aiogram.utils.formatting.**PhoneNumber**(\*body: Any, \*\*params: Any)

Bases: *Text*

Phone number element.

Will be wrapped into *aiogram.types.message\_entity.MessageEntity* with type *aiogram.enums.message\_entity\_type.MessageEntityType.PHONE\_NUMBER*

**class** aiogram.utils.formatting.**Bold**(\*body: Any, \*\*params: Any)

Bases: *Text*

Bold element.

Will be wrapped into *aiogram.types.message\_entity.MessageEntity* with type *aiogram.enums.message\_entity\_type.MessageEntityType.BOLD*

**class** aiogram.utils.formatting.**Italic**(\*body: Any, \*\*params: Any)

Bases: *Text*

Italic element.

Will be wrapped into *aiogram.types.message\_entity.MessageEntity* with type *aiogram.enums.message\_entity\_type.MessageEntityType.ITALIC*

**class** aiogram.utils.formatting.**Underline**(\*body: Any, \*\*params: Any)

Bases: *Text*

Underline element.

Will be wrapped into *aiogram.types.message\_entity.MessageEntity* with type *aiogram.enums.message\_entity\_type.MessageEntityType.UNDERLINE*

**class** aiogram.utils.formatting.**Strikethrough**(\*body: Any, \*\*params: Any)

Bases: *Text*

Strikethrough element.

Will be wrapped into *aiogram.types.message\_entity.MessageEntity* with type *aiogram.enums.message\_entity\_type.MessageEntityType.STRIKETHROUGH*

**class** aiogram.utils.formatting.**Spoiler**(\*body: Any, \*\*params: Any)

Bases: *Text*

Spoiler element.

Will be wrapped into *aiogram.types.message\_entity.MessageEntity* with type *aiogram.enums.message\_entity\_type.MessageEntityType.SPOILER*

**class** aiogram.utils.formatting.**Code**(\*body: Any, \*\*params: Any)

Bases: *Text*

Code element.

Will be wrapped into *aiogram.types.message\_entity.MessageEntity* with type *aiogram.enums.message\_entity\_type.MessageEntityType.CODE*

**class** aiogram.utils.formatting.**Pre**(\*body: Any, language: str | None = None, \*\*params: Any)

Bases: *Text*

Pre element.

Will be wrapped into *aiogram.types.message\_entity.MessageEntity* with type *aiogram.enums.message\_entity\_type.MessageEntityType.PRE*



**class** aiogram.utils.formatting.**TextLink**(\*body: Any, url: str, \*\*params: Any)

Bases: *Text*

Text link element.

Will be wrapped into *aiogram.types.message\_entity.MessageEntity* with type *aiogram.enums.message\_entity\_type.MessageEntityType.TEXT\_LINK*

**class** aiogram.utils.formatting.**TextMention**(\*body: Any, user: User, \*\*params: Any)

Bases: *Text*

Text mention element.

Will be wrapped into *aiogram.types.message\_entity.MessageEntity* with type *aiogram.enums.message\_entity\_type.MessageEntityType.TEXT\_MENTION*

**class** aiogram.utils.formatting.**CustomEmoji**(\*body: Any, custom\_emoji\_id: str, \*\*params: Any)

Bases: *Text*

Custom emoji element.

Will be wrapped into *aiogram.types.message\_entity.MessageEntity* with type *aiogram.enums.message\_entity\_type.MessageEntityType.CUSTOM\_EMOJI*

## 2.6 Changelog

### 2.6.1 3.0.0 (2023-09-01)

#### Bugfixes

- Replaced `datetime.datetime` with *DateTime* type wrapper across types to make dumped JSONs object more compatible with data that is sent by Telegram. #1277
- Fixed magic `.as_()` operation for values that can be interpreted as *False* (e.g. *0*). #1281
- Italic markdown from utils now uses correct decorators #1282
- Fixed method `Message.send_copy` for stickers. #1284
- Fixed `Message.send_copy` method, which was not working properly with stories, so not you can copy stories too (forwards messages). #1286
- Fixed error overlapping when validation error is caused by `remove_unset` root validator in base types and methods. #1290

### 2.6.2 3.0.0rc2 (2023-08-18)

#### Bugfixes

- Fixed missing message content types (`ContentType.USER_SHARED`, `ContentType.CHAT_SHARED`) #1252
- Fixed nested hashtag, cashtag and email message entities not being parsed correctly when these entities are inside another entity. #1259
- Moved global filters check placement into router to add chance to pass context from global filters into handlers in the same way as it possible in other places #1266

## Improved Documentation

- Added error handling example *examples/error\_handling.py* #1099
- Added a few words about skipping pending updates #1251
- Added a section on Dependency Injection technology #1253
- This update includes the addition of a multi-file bot example to the repository. #1254
- Refactored examples code to use aiogram enumerations and enhanced chat messages with markdown beautification's for a more user-friendly display. #1256
- Supplemented “Finite State Machine” section in Migration FAQ #1264
- Removed extra param in docstring of TelegramEventObserver's filter method and fixed typo in I18n documentation. #1268

## Misc

- Enhanced the warning message in dispatcher to include a JSON dump of the update when update type is not known. #1269
- Added support for Bot API 6.8 #1275

### 2.6.3 3.0.0rc1 (2023-08-06)

#### Features

- Added Currency enum. You can use it like this:

```
from aiogram.enums import Currency

await bot.send_invoice(
    ...,
    currency=Currency.USD,
    ...
)
```

#1194

- Updated keyboard builders with new methods for integrating buttons and keyboard creation more seamlessly. Added functionality to create buttons from existing markup and attach another builder. This improvement aims to make the keyboard building process more user-friendly and flexible. #1236
- Added support for message\_thread\_id in ChatActionSender #1249

## Bugfixes

- Fixed polling startup when “bot” key is passed manually into dispatcher workflow data [#1242](#)
- Added codegen configuration for lost shortcuts:
  - `ShippingQuery.answer`
  - `PreCheckoutQuery.answer`
  - `Message.delete_reply_markup`

[#1244](#)

## Improved Documentation

- Added documentation for webhook and polling modes. [#1241](#)

## Misc

- Reworked `InputFile` reading, removed `__aiter__` method, added *bot: Bot* argument to the `.read(...)` method, so, from now `URLInputFile` can be used without specifying bot instance. [#1238](#)
- Code-generated `__init__` typehints in types and methods to make IDE happy without additional pydantic plugin [#1245](#)

## 2.6.4 3.0.0b9 (2023-07-30)

### Features

- Added new shortcuts for `aiogram.types.chat_member_updated.ChatMemberUpdated` to send message to chat that member joined/left. [#1234](#)
- Added new shortcuts for `aiogram.types.chat_join_request.ChatJoinRequest` to make easier access to sending messages to users who wants to join to chat. [#1235](#)

### Bugfixes

- Fixed bot assignment in the `Message.send_copy` shortcut [#1232](#)
- Added model validation to remove `UNSET` before field validation. This change was necessary to correctly handle `parse_mode` where ‘UNSET’ is used as a sentinel value. Without the removal of ‘UNSET’, it would create issues when passed to model initialization from `Bot.method_name`. ‘UNSET’ was also added to typing. [#1233](#)
- Updated pydantic to 2.1 with few bugfixes

## Improved Documentation

- Improved docs, added basic migration guide (will be expanded later) [#1143](#)

## Deprecations and Removals

- Removed the use of the context instance (`Bot.get_current`) from all placements that were used previously. This is to avoid the use of the context instance in the wrong place. [#1230](#)

## 2.6.5 3.0.0b8 (2023-07-17)

### Features

- Added possibility to use custom events in routers (If router does not support custom event it does not break and passes it to included routers). [#1147](#)
- Added support for FSM in Forum topics.

The strategy can be changed in dispatcher:

```
from aiogram.fsm.strategy import FSMStrategy
...
dispatcher = Dispatcher(
    fsm_strategy=FSMStrategy.USER_IN_TOPIC,
    storage=..., # Any persistent storage
)
```

---

**Note:** If you have implemented you own storages you should extend record key generation with new one attribute - `thread_id`

---

[#1161](#)

- Improved CallbackData serialization.
  - Minimized UUID (hex without dashes)
  - Replaced bool values with int (true=1, false=0)

[#1163](#)

- Added a tool to make text formatting flexible and easy. More details on the [corresponding documentation page](#) [#1172](#)
- Added X-Telegram-Bot-API-Secret-Token header check [#1173](#)
- Made `allowed_updates` list to revolve automatically in `start_polling` method if not set explicitly. [#1178](#)
- Added possibility to pass custom headers to `URLInputFile` object [#1191](#)

## Bugfixes

- Change type of result in `InlineQueryResult` enum for `InlineQueryResultCachedMpeg4Gif` and `InlineQueryResultMpeg4Gif` to more correct according to documentation.

Change regexp for entities parsing to more correct (`InlineQueryResultType.yml`). #1146

- Fixed signature of startup/shutdown events to include the `**dispatcher.workflow_data` as the handler arguments. #1155
- Added missing `FORUM_TOPIC_EDITED` value to `content_type` property #1160
- Fixed compatibility with Python 3.8-3.9 (from previous release) #1162
- Fixed the markdown spoiler parser. #1176
- Fixed workflow data propagation #1196
- Fixed the serialization error associated with nested subtypes like `InputMedia`, `ChatMember`, etc.

The previously generated code resulted in an invalid schema under `pydantic v2`, which has stricter type parsing. Hence, subtypes without the specification of all subtype unions were generating an empty object. This has been rectified now. #1213

## Improved Documentation

- Changed small grammar typos for `upload_file` #1133

## Deprecations and Removals

- Removed text filter in due to is planned to remove this filter few versions ago.  
Use `F.text` instead #1170

## Misc

- Added full support of Bot API 6.6

**Danger:** Note that this issue has breaking changes described in in the Bot API changelog, this changes is not breaking in the API but breaking inside aiogram because Beta stage is not finished.

#1139

- Added full support of Bot API 6.7

**Warning:** Note that arguments `switch_pm_parameter` and `switch_pm_text` was deprecated and should be changed to `button` argument as described in API docs.

#1168

- Updated `Pydantic` to V2

**Warning:** Be careful, not all libraries is already updated to using V2

#1202

- Added global defaults `disable_web_page_preview` and `protect_content` in addition to `parse_mode` to the Bot instance, reworked internal request builder mechanism. [#1142](#)
- Removed bot parameters from storages [#1144](#)
- Replaced `ContextVar`'s with a new feature called [Validation Context](#) in Pydantic to improve the clarity, usability, and versatility of handling the Bot instance within method shortcuts.

**Danger: Breaking:** The 'bot' argument now is required in *URLInputFile*

#1210

- Updated magic-filter with new features
  - Added hint for `len(F)` error
  - Added not in operation

#1221

### 2.6.6 3.0.0b7 (2023-02-18)

**Warning:** Note that this version has incompatibility with Python 3.8-3.9 in case when you create an instance of `Dispatcher` outside of the any coroutine.

Sorry for the inconvenience, it will be fixed in the next version.

This code will not work:

```
dp = Dispatcher()

def main():
    ...
    dp.run_polling(...)

main()
```

But if you change it like this it should works as well:

```
router = Router()

async def main():
    dp = Dispatcher()
    dp.include_router(router)
    ...
    dp.start_polling(...)

asyncio.run(main())
```

## Features

- Added missing shortcuts, new enums, reworked old stuff

**Breaking** All previously added enums is re-generated in new place - *aiogram.enums* instead of *aiogram.types*

**Added enums:** *aiogram.enums.bot\_command\_scope\_type.BotCommandScopeType*,

```
aiogram.enums.chat_action.ChatAction,          aiogram.enums.chat_member_status.
ChatMemberStatus,          aiogram.enums.chat_type.ChatType,          aiogram.enums.
content_type.ContentType,    aiogram.enums.dice_emoji.DiceEmoji,    aiogram.enums.
inline_query_result_type.InlineQueryResultType,    aiogram.enums.input_media_type.
InputMediaType,    aiogram.enums.mask_position_point.MaskPositionPoint,    aiogram.
enums.menu_button_type.MenuButtonType,          aiogram.enums.message_entity_type.
MessageEntityType,    aiogram.enums.parse_mode.ParseMode,    aiogram.enums.poll_type.
PollType,    aiogram.enums.sticker_type.StickerType,    aiogram.enums.topic_icon_color.
TopicIconColor,    aiogram.enums.update_type.UpdateType,
```

**Added shortcuts:**

- **Chat** *aiogram.types.chat.Chat.Chat.get\_administrators()*,  
*aiogram.types.chat.Chat.delete\_message()*, *aiogram.types.chat.Chat.*  
*revoke\_invite\_link()*, *aiogram.types.chat.Chat.edit\_invite\_link()*,  
*aiogram.types.chat.Chat.create\_invite\_link()*, *aiogram.types.chat.Chat.*  
*export\_invite\_link()*, *aiogram.types.chat.Chat.do()*, *aiogram.types.chat.Chat.*  
*delete\_sticker\_set()*, *aiogram.types.chat.Chat.set\_sticker\_set()*, *aiogram.types.*  
*chat.Chat.get\_member()*, *aiogram.types.chat.Chat.get\_member\_count()*, *aiogram.*  
*types.chat.Chat.leave()*, *aiogram.types.chat.Chat.unpin\_all\_messages()*, *aiogram.*  
*types.chat.Chat.unpin\_message()*, *aiogram.types.chat.Chat.pin\_message()*,  
*aiogram.types.chat.Chat.set\_administrator\_custom\_title()*, *aiogram.types.chat.*  
*Chat.set\_permissions()*, *aiogram.types.chat.Chat.promote()*, *aiogram.types.chat.*  
*Chat.restrict()*, *aiogram.types.chat.Chat.unban()*, *aiogram.types.chat.Chat.*  
*ban()*, *aiogram.types.chat.Chat.set\_description()*, *aiogram.types.chat.Chat.*  
*set\_title()*, *aiogram.types.chat.Chat.delete\_photo()*, *aiogram.types.chat.Chat.*  
*set\_photo()*,
- **Sticker:** *aiogram.types.sticker.Sticker.set\_position\_in\_set()*,  
*aiogram.types.sticker.Sticker.delete\_from\_set()*,
- **User:** *aiogram.types.user.User.get\_profile\_photos()*

#952

- Added *callback answer* feature #1091
- Added a method that allows you to compactly register routers #1117

## Bugfixes

- Check status code when downloading file #816
- Fixed *ignore\_case* parameter in *aiogram.filters.command.Command* filter #1106

## Misc

- Added integration with new code-generator named [Butcher #1069](#)
- Added full support of [Bot API 6.4 #1088](#)
- Updated package metadata, moved build internals from Poetry to Hatch, added contributing guides. [#1095](#)
- Added full support of [Bot API 6.5](#)

**Danger:** Note that `aiogram.types.chat_permissions.ChatPermissions` is updated without backward compatibility, so now this object has no `can_send_media_messages` attribute

[#1112](#)

- Replaced error `TypeError: TelegramEventObserver.__call__() got an unexpected keyword argument '<name>'` with a more understandable one for developers and with a link to the documentation. [#1114](#)
- Added possibility to reply into webhook with files [#1120](#)
- Reworked graceful shutdown. Added method to stop polling. Now polling started from dispatcher can be stopped by signals gracefully without errors (on Linux and Mac). [#1124](#)

## 2.6.7 3.0.0b6 (2022-11-18)

### Features

- (again) Added possibility to combine filters with an *and/or* operations.  
Read more in “[Combining filters](#)” documentation section [#1018](#)

- Added following methods to Message class:
  - `Message.forward(...)`
  - `Message.edit_media(...)`
  - `Message.edit_live_location(...)`
  - `Message.stop_live_location(...)`
  - `Message.pin(...)`
  - `Message.unpin()`

[#1030](#)

- Added following methods to User class:
  - `User.mention_markdown(...)`
  - `User.mention_html(...)`

[#1049](#)

- Added full support of [Bot API 6.3 #1057](#)



## Bugfixes

- Fixed `Message.send_invoice` and `Message.reply_invoice`, added missing arguments #1047
  - Fixed copy and forward in:
    - `Message.answer(...)`
    - `Message.copy_to(...)`
- #1064

## Improved Documentation

- Fixed UA translations in `index.po` #1017
- Fix typehints for `Message`, `reply_media_group` and `answer_media_group` methods #1029
- Removed an old now non-working feature #1060

## Misc

- Enabled testing on Python 3.11 #1044
- Added a mandatory dependency `certifi` in due to in some cases on systems that doesn't have updated certificates the requests to Bot API fails with reason `[SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: self signed certificate in certificate chain` #1066

## 2.6.8 3.0.0b5 (2022-10-02)

### Features

- Add PyPy support and run tests under PyPy #985
- Added message text to aiogram exceptions representation #988
- Added warning about using magic filter from `magic_filter` instead of *aiogram's* ones. Is recommended to use *from aiogram import F* instead of *from magic\_filter import F* #990
- Added more detailed error when server response can't be deserialized. This feature will help to debug unexpected responses from the Server #1014

### Bugfixes

- Reworked error event, introduced `aiogram.types.error_event.ErrorEvent` object. #898
- Fixed escaping markdown in `aiogram.utils.markdown` module #903
- Fixed polling crash when Telegram Bot API raises HTTP 429 status-code. #995
- Fixed empty mention in command parsing, now it will be `None` instead of an empty string #1013

## Improved Documentation

- Initialized Docs translation (added Ukrainian language) [#925](#)

## Deprecations and Removals

- Removed filters factory as described in corresponding issue. [#942](#)

## Misc

- Now Router/Dispatcher accepts only keyword arguments. [#982](#)

## 2.6.9 3.0.0b4 (2022-08-14)

### Features

- Add class helper ChatAction for constants that Telegram BotAPI uses in sendChatAction request. In my opinion, this will help users and will also improve compatibility with 2.x version where similar class was called “ChatActions”. [#803](#)
- Added possibility to combine filters or invert result

Example:

```
Text(text="demo") | Command(commands=["demo"])
MyFilter() & AnotherFilter()
~StateFilter(state='my-state')
```

[#894](#)

- Fixed type hints for redis TTL params. [#922](#)
- Added `full_name` shortcut for `Chat` object [#929](#)

### Bugfixes

- Fixed false-positive coercing of Union types in API methods [#901](#)
- Added 3 missing content types:
  - `proximity_alert_triggered`
  - `supergroup_chat_created`
  - `channel_chat_created`

[#906](#)

- Fixed the ability to compare the state, now comparison to copy of the state will return `True`. [#927](#)
- Fixed default lock kwargs in RedisEventIsolation. [#972](#)

## Misc

- Restrict including routers with strings [#896](#)
- Changed `CommandPatterType` to `CommandPatternType` in `aiogram/dispatcher/filters/command.py` [#907](#)
- Added full support of [Bot API 6.1](#) [#936](#)

- **Breaking!** More flat project structure

These packages was moved, imports in your code should be fixed:

- `aiogram.dispatcher.filters` -> `aiogram.filters`
- `aiogram.dispatcher.fsm` -> `aiogram.fsm`
- `aiogram.dispatcher.handler` -> `aiogram.handler`
- `aiogram.dispatcher.webhook` -> `aiogram.webhook`
- `aiogram.dispatcher.flags/*` -> `aiogram.dispatcher.flags` (single module instead of package)

[#938](#)

- Removed deprecated `router.<event>_handler` and `router.register_<event>_handler` methods. [#941](#)
- Deprecated filters factory. It will be removed in next Beta (3.0b5) [#942](#)
- `MessageEntity` method `get_text` was removed and `extract` was renamed to `extract_from` [#944](#)
- Added full support of [Bot API 6.2](#) [#975](#)

## 2.6.10 3.0.0b3 (2022-04-19)

### Features

- Added possibility to get command magic result as handler argument [#889](#)
- Added full support of [Telegram Bot API 6.0](#) [#890](#)

### Bugfixes

- Fixed I18n lazy-proxy. Disabled caching. [#839](#)
- Added parsing of spoiler message entity [#865](#)
- Fixed default `parse_mode` for `Message.copy_to()` method. [#876](#)
- Fixed `CallbackData` factory parsing `IntEnum`'s [#885](#)

## Misc

- Added automated check that pull-request adds a changes description to **CHANGES** directory [#873](#)
- Changed `Message.html_text` and `Message.md_text` attributes behaviour when message has no text. The empty string will be used instead of raising error. [#874](#)
- Used `redis-py` instead of `aioredis` package in due to this packages was merged into single one [#882](#)
- Solved common naming problem with middlewares that confusing too much developers - now you can't see the `middleware` and `middlewares` attributes at the same point because this functionality encapsulated to special interface. [#883](#)

## 2.6.11 3.0.0b2 (2022-02-19)

### Features

- Added possibility to pass additional arguments into the aiohttp webhook handler to use this arguments inside handlers as the same as it possible in polling mode. [#785](#)
- Added possibility to add handler flags via decorator (like *pytest.mark* decorator but *aiogram.flags*) [#836](#)
- Added ChatActionSender utility to automatically sends chat action while long process is running. It also can be used as message middleware and can be customized via `chat_action` flag. [#837](#)

### Bugfixes

- Fixed unexpected behavior of sequences in the StateFilter. [#791](#)
- Fixed exceptions filters [#827](#)

### Misc

- Logger name for processing events is changed to `aiogram.events`. [#830](#)
- Added full support of Telegram Bot API 5.6 and 5.7 [#835](#)
- **BREAKING** Events isolation mechanism is moved from FSM storages to standalone managers [#838](#)

## 2.6.12 3.0.0b1 (2021-12-12)

### Features

- Added new custom operation for MagicFilter named `as_`  
Now you can use it to get magic filter result as handler argument

```
from aiogram import F

...

@router.message(F.text.regexp(r"^(\d+)$").as_("digits"))
async def any_digits_handler(message: Message, digits: Match[str]):
    await message.answer(html.quote(str(digits)))

@router.message(F.photo[-1].as_("photo"))
async def download_photos_handler(message: Message, photo: PhotoSize, bot: Bot):
    content = await bot.download(photo)
```

[#759](#)

## Bugfixes

- Fixed: Missing ChatMemberHandler import in aiogram/dispatcher/handler #751

## Misc

- Check destiny in case of no with\_destiny enabled in RedisStorage key builder #776
- Added full support of Bot API 5.5 #777
- Stop using feature from #336. From now settings of client-session should be placed as initializer arguments instead of changing instance attributes. #778
- Make TelegramAPIServer files wrapper in local mode bi-directional (server-client, client-server) Now you can convert local path to server path and server path to local path. #779

## 2.6.13 3.0.0a18 (2021-11-10)

### Features

- Breaking: Changed the signature of the session middlewares Breaking: Renamed AiohttpSession.make\_request method parameter from call to method to match the naming in the base class Added middleware for logging outgoing requests #716
- Improved description of filters resolving error. For example when you try to pass wrong type of argument to the filter but don't know why filter is not resolved now you can get error like this:

```
aiogram.exceptions.FiltersResolveError: Unknown keyword filters: {'content_types'}
Possible cases:
- 1 validation error for ContentTypesFilter
  content_types
    Invalid content types {'42'} is not allowed here (type=value_error)
```

#717

- **Breaking internal API change** Reworked FSM Storage record keys propagation #723
- Implemented new filter named MagicData(magic\_data) that helps to filter event by data from middlewares or other filters

For example your bot is running with argument named config that contains the application config then you can filter event by value from this config:

```
@router.message(magic_data=F.event.from_user.id == F.config.admin_id)
...
```

#724

## Bugfixes

- Fixed I18n context inside error handlers #726
- Fixed bot session closing before emit shutdown #734
- Fixed: bound filter resolving does not require children routers #736

## Misc

- Enabled testing on Python 3.10 Removed `async_lru` dependency (is incompatible with Python 3.10) and replaced usage with protected property #719
- Converted README.md to README.rst and use it as base file for docs #725
- Rework filters resolving:
  - Automatically apply Bound Filters with default values to handlers
  - Fix data transfer from parent to included routers filters#727
- Added full support of Bot API 5.4 <https://core.telegram.org/bots/api-changelog#november-5-2021> #744

### 2.6.14 3.0.0a17 (2021-09-24)

## Misc

- Added `html_text` and `md_text` to Message object #708
- Refactored I18n, added context managers for I18n engine and current locale #709

### 2.6.15 3.0.0a16 (2021-09-22)

## Features

- Added support of local Bot API server files downloading  
When Local API is enabled files can be downloaded via `bot.download/bot.download_file` methods. #698
- Implemented I18n & L10n support #701

## Misc

- Covered by tests and docs KeyboardBuilder util #699
- **Breaking!!!**. Refactored and renamed exceptions.
  - Exceptions module was moved from `aiogram.utils.exceptions` to `aiogram.exceptions`
  - Added prefix *Telegram* for all error classes#700
- Replaced all pragma: no cover marks via global `.coveragerc` config #702

- Updated dependencies.

**Breaking for framework developers** Now all optional dependencies should be installed as extra: *poetry install -E fast -E redis -E proxy -E i18n -E docs* #703

## 2.6.16 3.0.0a15 (2021-09-10)

### Features

- Ability to iterate over all states in StatesGroup. Aiogram already had in check for states group so this is relative feature. #666

### Bugfixes

- Fixed incorrect type checking in the `aiogram.utils.keyboard.KeyboardBuilder` #674

### Misc

- Disable ContentType filter by default #668
- Moved update type detection from Dispatcher to Update object #669
- Updated **pre-commit** config #681
- Reworked **handlers\_in\_use** util. Function moved to Router as method `.resolve_used_update_types()` #682

## 2.6.17 3.0.0a14 (2021-08-17)

### Features

- add aliases for edit/delete reply markup to Message #662
- Reworked outer middleware chain. Prevent to call many times the outer middleware for each nested router #664

### Bugfixes

- Prepare parse mode for InputMessageContent in AnswerInlineQuery method #660

### Improved Documentation

- Added integration with towncrier #602

## Misc

- Added `.editorconfig` #650
- Redis storage speedup globals #651
- add `allow_sending_without_reply` param to `Message` reply aliases #663

### 2.6.18 2.14.3 (2021-07-21)

- Fixed `ChatMember` type detection via adding customizable object serialization mechanism (#624, #623)

### 2.6.19 2.14.2 (2021-07-26)

- Fixed `MemoryStorage` cleaner (#619)
- Fixed unused default locale in `I18nMiddleware` (#562, #563)

### 2.6.20 2.14 (2021-07-27)

- Full support of Bot API 5.3 (#610, #614)
- Fixed `Message.send_copy` method for polls (#603)
- Updated pattern for `GroupDeactivated` exception (#549)
- Added `caption_entities` field in `InputMedia` base class (#583)
- Fixed HTML text decorations for tag `pre` (#597 fixes issues #596 and #481)
- Fixed `Message.get_full_command` method for messages with caption (#576)
- Improved `MongoStorage`: remove documents with empty data from `aiogram_data` collection to save memory. (#609)

### 2.6.21 2.13 (2021-04-28)

- Added full support of Bot API 5.2 (#572)
- Fixed usage of `provider_data` argument in `sendInvoice` method call
- Fixed builtin command filter args (#556) (#558)
- Allowed to use `State` instances FSM storage directly (#542)
- Added possibility to get `i18n` locale without `User` instance (#546)
- Fixed returning type of `Bot.*_chat_invite_link()` methods #548 (#549)
- Fixed deep-linking util (#569)
- Small changes in documentation - describe limits in docstrings corresponding to the current limit. (#565)
- Fixed internal call to deprecated `'is_private'` method (#553)
- Added possibility to use `allowed_updates` argument in Polling mode (#564)



### 2.6.22 2.12.1 (2021-03-22)

- Fixed `TypeError`: Value should be instance of 'User' not 'NoneType' (#527)
- Added missing `Chat.message_auto_delete_time` field (#535)
- Added `MediaGroup` filter (#528)
- Added `Chat.delete_message` shortcut (#526)
- Added mime types parsing for `aiogram.types.Document` (#431)
- Added warning in `TelegramObject.__setattr__` when Telegram adds a new field (#532)
- Fixed `examples/chat_type_filter.py` (#533)
- Removed redundant definitions in framework code (#531)

### 2.6.23 2.12 (2021-03-14)

- Full support for Telegram Bot API 5.1 (#519)
- Fixed sending playlist of audio files and documents (#465, #468)
- Fixed `FSMContextProxy.setdefault` method (#491)
- Fixed `Message.answer_location` and `Message.reply_location` unable to send live location (#497)
- Fixed `user_id` and `chat_id` getters from the context at Dispatcher `check_key`, `release_key` and `throttle` methods (#520)
- Fixed `Chat.update_chat` method and all similar situations (#516)
- Fixed `MediaGroup` attach methods (#514)
- Fixed state filter for inline keyboard query callback in groups (#508, #510)
- Added missing `ContentTypes.DICE` (#466)
- Added missing `vcard` argument to `InputContactMessageContent` constructor (#473)
- Add missing exceptions: `MessageIdInvalid`, `CantRestrictChatOwner` and `UserIsAnAdministratorOfTheChat` (#474, #512)
- Added `answer_chat_action` to the `Message` object (#501)
- Added dice to `message.send_copy` method (#511)
- Removed deprecation warning from `Message.send_copy`
- Added an example of integration between externally created `aiohttp` Application and `aiogram` (#433)
- Added `split_separator` argument to `safe_split_text` (#515)
- Fixed some typos in docs and examples (#489, #490, #498, #504, #514)

### 2.6.24 2.11.2 (2021-11-10)

- Fixed default parse mode
- Added missing “supports\_streaming” argument to answer\_video method [#462](#)

### 2.6.25 2.11.1 (2021-11-10)

- Fixed files URL template
- Fix MessageEntity serialization for API calls [#457](#)
- When entities are set, default parse\_mode become disabled ([#461](#))
- Added parameter supports\_streaming to reply\_video, remove redundant docstrings ([#459](#))
- Added missing parameter to promoteChatMember alias ([#458](#))

### 2.6.26 2.11 (2021-11-08)

- Added full support of Telegram Bot API 5.0 ([#454](#))
- **Added possibility to more easy specify custom API Server (example)**
  - WARNING: API method close was named in Bot class as close\_bot in due to Bot instance already has method with the same name. It will be changed in aiogram 3.0
- Added alias to Message object Message.copy\_to with deprecation of Message.send\_copy
- ChatType.SUPER\_GROUP renamed to ChatType.SUPERGROUP ([#438](#))

### 2.6.27 2.10.1 (2021-09-14)

- Fixed critical bug with getting asyncio event loop in executor. ([#424](#)) `AttributeError: 'NoneType' object has no attribute 'run_until_complete'`

### 2.6.28 2.10 (2021-09-13)

- Breaking change: Stop using \_MainThread event loop in bot/dispatcher instances ([#397](#))
- Breaking change: Replaced aiomongo with motor ([#368](#), [#380](#))
- Fixed: TelegramObject’s aren’t destroyed after update handling [#307](#) ([#371](#))
- Add setting current context of Telegram types ([#369](#))
- Fixed markdown escaping issues ([#363](#))
- Fixed HTML characters escaping ([#409](#))
- Fixed italic and underline decorations when parse entities to Markdown
- Fixed [#413](#): parse entities positioning ([#414](#))
- Added missing thumb parameter ([#362](#))
- Added public methods to register filters and middlewares ([#370](#))
- Added ChatType builtin filter ([#356](#))

- Fixed IDFilter checking message from channel (#376)
- Added missed answer\_poll and reply\_poll (#384)
- Added possibility to ignore message caption in commands filter (#383)
- Fixed addStickerToSet method
- Added preparing thumb in send\_document method (#391)
- Added exception MessageToPinNotFound (#404)
- Fixed handlers parameter-spec solving (#408)
- Fixed CallbackQuery.answer() returns nothing (#420)
- CHOSEN\_INLINE\_RESULT is a correct API-term (#415)
- Fixed missing attributes for Animation class (#422)
- Added missed emoji argument to reply\_dice (#395)
- Added is\_chat\_creator method to ChatMemberStatus (#394)
- Added missed ChatPermissions to \_\_all\_\_ (#393)
- Added is\_forward method to Message (#390)
- Fixed usage of deprecated is\_private function (#421)

and many others documentation and examples changes:

- Updated docstring of RedisStorage2 (#423)
- Updated I18n example (added docs and fixed typos) (#419)
- A little documentation revision (#381)
- Added comments about correct errors\_handlers usage (#398)
- Fixed typo rexex -> regex (#386)
- Fixed docs Quick start page code blocks (#417)
- fixed type hints of callback\_data (#400)
- Prettify readme, update downloads stats badge (#406)

### 2.6.29 2.9.2 (2021-06-13)

- Fixed Message.get\_full\_command() #352
- Fixed markdown util #353

### 2.6.30 2.9 (2021-06-08)

- Added full support of Telegram Bot API 4.9
- Fixed user context at poll\_answer update (#322)
- Fix Chat.set\_description (#325)
- Add lazy session generator (#326)
- Fix text decorations (#315, #316, #328)
- Fix missing InlineQueryResultPhoto parse\_mode field (#331)

- Fix fields from parent object in `KeyboardButton` (#344 fixes #343)
- Add possibility to get bot id without calling `get_me` (#296)

### 2.6.31 2.8 (2021-04-26)

- Added full support of Bot API 4.8
- Added `Message.answer_dice` and `Message.reply_dice` methods (#306)

### 2.6.32 2.7 (2021-04-07)

- Added full support of Bot API 4.7 (#294 #289)
- Added default parse mode for `send_animation` method (#293 #292)
- Added new API exception when poll requested in public chats (#270)
- Make correct User and Chat `get_mention` methods (#277)
- Small changes and other minor improvements

### 2.6.33 2.6.1 (2021-01-25)

- Fixed reply `KeyboardButton` initializer with `request_poll` argument (#266)
- Added helper for poll types (`aiogram.types.PollType`)
- Changed behavior of `Telegram_object.as_*` and `.to_*` methods. It will no more mutate the object. (#247)

### 2.6.34 2.6 (2021-01-23)

- Full support of Telegram Bot API v4.6 (Polls 2.0) #265
- Added new filter - `IsContactSender` (commit)
- Fixed proxy extra dependencies version #262

### 2.6.35 2.5.3 (2021-01-05)

- #255 Updated `CallbackData` factory validity check. More correct for non-latin symbols
- #256 Fixed `renamed_argument` decorator error
- #257 One more fix of `CommandStart` filter

### 2.6.36 2.5.2 (2021-01-01)

- Get back `quote_html` and `escape_md` functions

### 2.6.37 2.5.1 (2021-01-01)

- Hot-fix of `CommandStart` filter

### 2.6.38 2.5 (2021-01-01)

- Added full support of Telegram Bot API 4.5 (#250, #251)
- #239 Fixed `check_token` method
- #238, #241: Added deep-linking utils
- #248 Fixed support of `aiohttp-socks`
- Updated `setup.py`. No more use of internal pip API
- Updated links to documentations (<https://docs.aiogram.dev>)
- Other small changes and minor improvements (#223 and others...)

### 2.6.39 2.4 (2021-10-29)

- Added `Message.send_copy` method (forward message without forwarding)
- Safe close of `aiohttp` client session (no more exception when application is shutdown)
- No more “adWanced” words in project #209
- Arguments `user` and `chat` is renamed to `user_id` and `chat_id` in `Dispatcher.throttle` method #196
- Fixed `set_chat_permissions` #198
- Fixed `Dispatcher` polling task does not process cancellation #199, #201
- Fixed compatibility with latest `asyncio` version #200
- Disabled caching by default for `lazy_gettext` method of `I18nMiddleware` #203
- Fixed HTML user mention parser #205
- Added `IsReplyFilter` #210
- Fixed `send_poll` method arguments #211
- Added `OrderedHelper` #215
- Fix incorrect completion order. #217

### 2.6.40 2.3 (2021-08-16)

- Full support of Telegram Bot API 4.4
- Fixed [#143](#)
- Added new filters from issue [#151](#): [#172](#), [#176](#), [#182](#)
- Added expire argument to RedisStorage2 and other storage fixes [#145](#)
- Fixed JSON and Pickle storages [#138](#)
- Implemented MongoStorage [#153](#) based on aiomongo (soon motor will be also added)
- Improved tests
- Updated examples
- Warning: Updated auth widget util. [#190](#)
- Implemented throttle decorator [#181](#)

### 2.6.41 2.2 (2021-06-09)

- Provides latest Telegram Bot API (4.3)
- Updated docs for filters
- Added opportunity to use different bot tokens from single bot instance (via context manager, [#100](#))
- IMPORTANT: Fixed Typo: data -> bucket in update\_bucket for RedisStorage2 ([#132](#))

### 2.6.42 2.1 (2021-04-18)

- Implemented all new features from Telegram Bot API 4.2
- `is_member` and `is_admin` methods of `ChatMember` and `ChatMemberStatus` was renamed to `is_chat_member` and `is_chat_admin`
- Remover func filter
- Added some useful Message edit functions (`Message.edit_caption`, `Message.edit_media`, `Message.edit_reply_markup`) ([#121](#), [#103](#), [#104](#), [#112](#))
- Added requests timeout for all methods ([#110](#))
- Added `answer*` methods to `Message` object ([#112](#))
- Made some improvements of `CallbackData` factory
- Added deep-linking parameter filter to `CommandStart` filter
- Implemented opportunity to use DNS over socks ([#97](#) -> [#98](#))
- Implemented logging filter for extending `LogRecord` attributes (Will be usefull with external logs collector utils like GrayLog, Kibana and etc.)
- Updated `requirements.txt` and `dev_requirements.txt` files
- Other small changes and minor improvements

### 2.6.43 2.0.1 (2021-12-31)

- Implemented CallbackData factory ([example](#))
- Implemented methods for answering to inline query from context and reply with animation to the messages. [#85](#)
- Fixed installation from tar.gz [#84](#)
- More exceptions (ChatIdIsEmpty and NotEnoughRightsToRestrict)

### 2.6.44 2.0 (2021-10-28)

This update will break backward compability with Python 3.6 and works only with Python 3.7+: - contextvars (PEP-567); - New syntax for annotations (PEP-563).

Changes: - 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 `I18nMiddleware`; - Errors handlers now should accept only two arguments (current update and exception); - Used `aiohttp_socks` instead of `aiosocksy` for Socks4/5 proxy; - `types.ContentType` was divided to `types.ContentType` and `types.ContentTypes`; - Allowed to use `rapidjson` instead of `ujson/json`; - `.current()` method in bot and dispatcher objects was renamed to `get_current()`;

Full changelog - You can read more details about this release in migration FAQ: [https://aiogram.readthedocs.io/en/latest/migration\\_1\\_to\\_2.html](https://aiogram.readthedocs.io/en/latest/migration_1_to_2.html)

### 2.6.45 1.4 (2021-08-03)

- Bot API 4.0 ([#57](#))

### 2.6.46 1.3.3 (2021-07-16)

- Fixed markup-entities parsing;
- Added more API exceptions;
- Now `InlineQueryResultLocation` has `live_period`;
- Added more message content types;
- Other small changes and minor improvements.

### 2.6.47 1.3.2 (2021-05-27)

- Fixed crashing of polling process. (i think)
- Added `parse_mode` field into input query results according to Bot API Docs.
- Added new methods for Chat object. ([#42](#), [#43](#))
- **Warning:** disabled connections limit for bot `aiohttp` session.
- **Warning:** Destroyed “temp sessions” mechanism.
- Added new error types.
- Refactored detection of error type.

- Small fixes of executor util.
- Fixed RethinkDBStorage

### 2.6.48 1.3.1 (2018-05-27)

### 2.6.49 1.3 (2021-04-22)

- Allow to use Socks5 proxy (need manually install `aiosocksy`).
- Refactored `aiogram.utils.executor` module.
- **[Warning]** Updated requirements list.

### 2.6.50 1.2.3 (2018-04-14)

- Fixed API errors detection
- Fixed compability of `setup.py` with pip 10.0.0

### 2.6.51 1.2.2 (2018-04-08)

- Added more error types.
- Implemented method `InputFile.from_url(url: str)` for downloading files.
- Implemented big part of API method tests.
- Other small changes and mminor improvements.

### 2.6.52 1.2.1 (2018-03-25)

- Fixed handling Venue's [#27, #26]
- Added `parse_mode` to all medias (Bot API 3.6 support) [#23]
- Now regexp filter can be used with callback query data [#19]
- Improvements in `InlineKeyboardMarkup` & `ReplyKeyboardMarkup` objects [#21]
- Other bug & typo fixes and minor improvements.

### 2.6.53 1.2 (2018-02-23)

- Full provide Telegram Bot API 3.6
- Fixed critical error: `Fatal Python error: PyImport_GetModuleDict: no module dictionary!`
- Implemented connection pool in RethinkDB driver
- Typo fixes of documentstion
- Other bug fixes and minor improvements.



### 2.6.54 1.1 (2018-01-27)

- Added more methods for data types (like `message.reply_sticker(...)` or `file.download(...)`)
- Typo fixes of documentstion
- Allow to set default parse mode for messages (`Bot( ..., parse_mode='HTML')`)
- Allowed to cancel event from the `Middleware.on_pre_process_<event type>`
- Fixed sending files with correct names.
- Fixed `MediaGroup`
- Added RethinkDB storage for FSM (`aiogram.contrib.fsm_storage.rethinkdb`)

### 2.6.55 1.0.4 (2018-01-10)

### 2.6.56 1.0.3 (2018-01-07)

- Added middlewares mechanism.
- Added example for middlewares and throttling manager.
- Added logging middleware (`aiogram.contrib.middlewares.logging.LoggingMiddleware`)
- Fixed handling errors in async tasks (marked as 'async\_task')
- Small fixes and other minor improvements.

### 2.6.57 1.0.2 (2017-11-29)

### 2.6.58 1.0.1 (2017-11-21)

- Implemented `types.InputFile` for more easy sending local files
- **Danger!** Fixed typo in word pooling. Now whatever all methods with that word marked as deprecated and original methods is renamed to pooling. Check it in you'r code before updating!
- Fixed helper for chat actions (`types.ChatActions`)
- Added [example](#) for media group.

### 2.6.59 1.0 (2017-11-19)

- Remaked data types serialoization/deserialization mechanism (Speed up).
- Fully rewrited all Telegram data types.
- Bot object was fully rewritted (regenerated).
- Full provide Telegram Bot API 3.4+ (with `sendMediaGroup`)
- Warning: Now `BaseStorage.close()` is awaitable! (FSM)
- Fixed compability with uvloop.
- More employments for `aiogram.utils.context`.
- Allowed to disable `ujson`.

- Other bug fixes and minor improvements.
- Migrated from Bitbucket to Github.

**2.6.60 0.4.1 (2017-08-03)**

**2.6.61 0.4 (2017-08-05)**

**2.6.62 0.3.4 (2017-08-04)**

**2.6.63 0.3.3 (2017-07-05)**

**2.6.64 0.3.2 (2017-07-04)**

**2.6.65 0.3.1 (2017-07-04)**

**2.6.66 0.2b1 (2017-06-00)**

**2.6.67 0.1 (2017-06-03)**

## 2.7 Contributing

You're welcome to contribute to aiogram!

*aiogram* is an open-source project, and anyone can contribute to it in any possible way

### 2.7.1 Developing

Before making any changes in the framework code, it is necessary to fork the project and clone the project to your PC and know how to do a pull-request.

How to work with pull-request you can read in the [GitHub docs](#)

Also in due to this project is written in Python, you will need Python to be installed (is recommended to use latest Python versions, but any version starting from 3.8 can be used)

#### Use virtualenv

You can create a virtual environment in a directory using venv module (it should be pre-installed by default):

This action will create a `.venv` directory with the Python binaries and then you will be able to install packages into that isolated environment.

## Activate the environment

Linux / macOS:

```
source .venv/bin/activate
```

Windows cmd

```
.\.venv\Scripts\activate
```

Windows PowerShell

```
.\.venv\Scripts\activate.ps1
```

To check it worked, use described command, it should show the `pip` version and location inside the isolated environment

```
pip -V
```

Also make sure you have the latest `pip` version in your virtual environment to avoid errors on next steps:

```
python -m pip install --upgrade pip
```

## Setup project

After activating the environment install *aiogram* from sources and their dependencies.

Linux / macOS:

```
pip install -e ."[dev,test,docs,fast,redis,proxy,i18n]"
```

Windows:

```
pip install -e .[dev,test,docs,fast,redis,proxy,i18n]
```

It will install *aiogram* in editable mode into your virtual environment and all dependencies.

## Making changes in code

At this point you can make any changes in the code that you want, it can be any fixes, implementing new features or experimenting.

## Format the code (code-style)

Note that this project is Black-formatted, so you should follow that code-style, too be sure You're correctly doing this let's reformat the code automatically:

```
black aiogram tests examples
isort aiogram tests examples
```

### Run tests

All changes should be tested:

```
pytest tests
```

Also if you are doing something with Redis-storage, you will need to test everything works with Redis:

```
pytest --redis redis://<host>:<port>/<db> tests
```

### Docs

We are using *Sphinx* to render docs in different languages, all sources located in *docs* directory, you can change the sources and to test it you can start live-preview server and look what you are doing:

```
sphinx-autobuild --watch aiogram/ docs/ docs/_build/
```

### Docs translations

Translation of the documentation is very necessary and cannot be done without the help of the community from all over the world, so you are welcome to translate the documentation into different languages.

Before start, let's up to date all texts:

```
cd docs
make gettext
sphinx-intl update -p _build/gettext -l <language_code>
```

Change the `<language_code>` in example below to the target language code, after that you can modify texts inside `docs/locale/<language_code>/LC_MESSAGES` as `*.po` files by using any text-editor or specialized utilites for GNU Gettext, for example via [poedit](#).

To view results:

```
sphinx-autobuild --watch aiogram/ docs/ docs/_build/ -D language=<language_code>
```

### Describe changes

Describe your changes in one or more sentences so that bot developers know what's changed in their favorite framework - create `<code>.<category>.rst` file and write the description.

`<code>` is Issue or Pull-request number, after release link to this issue will be published to the *Changelog* page.

`<category>` is a changes category marker, it can be one of:

- `feature` - when you are implementing new feature
- `bugfix` - when you fix a bug
- `doc` - when you improve the docs
- `removal` - when you remove something from the framework
- `misc` - when changed something inside the Core or project configuration

If you have troubles with changing category feel free to ask Core-contributors to help with choosing it.

## Complete

After you have made all your changes, publish them to the repository and create a pull request as mentioned at the beginning of the article and wait for a review of these changes.

### 2.7.2 Star on GitHub

You can “star” repository on GitHub - <https://github.com/aiogram/aiogram> (click the star button at the top right)

Adding stars makes it easier for other people to find this project and understand how useful it is.

### 2.7.3 Guides

You can write guides how to develop Bots on top of aiogram and publish it into YouTube, Medium, GitHub Books, any Courses platform or any other platform that you know.

This will help more people learn about the framework and learn how to use it

### 2.7.4 Take answers

The developers is always asks for any question in our chats or any other platforms like GitHub Discussions, StackOverflow and others, feel free to answer to this questions.

### 2.7.5 Funding

The development of the project is free and not financed by commercial organizations, it is my personal initiative (@JRootJunior) and I am engaged in the development of the project in my free time.

So, if you want to financially support the project, or, for example, give me a pizza or a beer, you can do it on [OpenCollective](#).



## PYTHON MODULE INDEX

### a

aiogram.dispatcher.flags, 438  
aiogram.enums.bot\_command\_scope\_type, 376  
aiogram.enums.chat\_action, 377  
aiogram.enums.chat\_member\_status, 377  
aiogram.enums.chat\_type, 378  
aiogram.enums.content\_type, 378  
aiogram.enums.currency, 380  
aiogram.enums.dice\_emoji, 383  
aiogram.enums.encrypted\_passport\_element, 383  
aiogram.enums.inline\_query\_result\_type, 384  
aiogram.enums.input\_media\_type, 384  
aiogram.enums.mask\_position\_point, 385  
aiogram.enums.menu\_button\_type, 385  
aiogram.enums.message\_entity\_type, 385  
aiogram.enums.parse\_mode, 386  
aiogram.enums.passport\_element\_error\_type, 386  
aiogram.enums.poll\_type, 387  
aiogram.enums.sticker\_format, 387  
aiogram.enums.sticker\_type, 387  
aiogram.enums.topic\_icon\_color, 387  
aiogram.enums.update\_type, 388  
aiogram.exceptions, 437  
aiogram.handlers.callback\_query, 440  
aiogram.methods.add\_sticker\_to\_set, 337  
aiogram.methods.answer\_callback\_query, 236  
aiogram.methods.answer\_inline\_query, 372  
aiogram.methods.answer\_pre\_checkout\_query, 329  
aiogram.methods.answer\_shipping\_query, 330  
aiogram.methods.answer\_web\_app\_query, 375  
aiogram.methods.approve\_chat\_join\_request, 237  
aiogram.methods.ban\_chat\_member, 238  
aiogram.methods.ban\_chat\_sender\_chat, 240  
aiogram.methods.close, 241  
aiogram.methods.close\_forum\_topic, 242  
aiogram.methods.close\_general\_forum\_topic, 243  
aiogram.methods.copy\_message, 243  
aiogram.methods.create\_chat\_invite\_link, 245  
aiogram.methods.create\_forum\_topic, 247  
aiogram.methods.create\_invoice\_link, 331  
aiogram.methods.create\_new\_sticker\_set, 338  
aiogram.methods.decline\_chat\_join\_request, 248  
aiogram.methods.delete\_chat\_photo, 249  
aiogram.methods.delete\_chat\_sticker\_set, 250  
aiogram.methods.delete\_forum\_topic, 251  
aiogram.methods.delete\_message, 361  
aiogram.methods.delete\_my\_commands, 252  
aiogram.methods.delete\_sticker\_from\_set, 339  
aiogram.methods.delete\_sticker\_set, 340  
aiogram.methods.delete\_webhook, 356  
aiogram.methods.edit\_chat\_invite\_link, 253  
aiogram.methods.edit\_forum\_topic, 254  
aiogram.methods.edit\_general\_forum\_topic, 255  
aiogram.methods.edit\_message\_caption, 362  
aiogram.methods.edit\_message\_live\_location, 364  
aiogram.methods.edit\_message\_media, 366  
aiogram.methods.edit\_message\_reply\_markup, 367  
aiogram.methods.edit\_message\_text, 368  
aiogram.methods.export\_chat\_invite\_link, 256  
aiogram.methods.forward\_message, 257  
aiogram.methods.get\_chat, 259  
aiogram.methods.get\_chat\_administrators, 259  
aiogram.methods.get\_chat\_member, 260  
aiogram.methods.get\_chat\_member\_count, 261  
aiogram.methods.get\_chat\_menu\_button, 262  
aiogram.methods.get\_custom\_emoji\_stickers, 341  
aiogram.methods.get\_file, 263  
aiogram.methods.get\_forum\_topic\_icon\_stickers, 264  
aiogram.methods.get\_game\_high\_scores, 352  
aiogram.methods.get\_me, 264  
aiogram.methods.get\_my\_commands, 265  
aiogram.methods.get\_my\_default\_administrator\_rights, 266  
aiogram.methods.get\_my\_description, 267  
aiogram.methods.get\_my\_name, 268

[aiogram.methods.get\\_my\\_short\\_description](#), 268  
[aiogram.methods.get\\_sticker\\_set](#), 342  
[aiogram.methods.get\\_updates](#), 356  
[aiogram.methods.get\\_user\\_profile\\_photos](#), 269  
[aiogram.methods.get\\_webhook\\_info](#), 358  
[aiogram.methods.hide\\_general\\_forum\\_topic](#), 270  
[aiogram.methods.leave\\_chat](#), 271  
[aiogram.methods.log\\_out](#), 272  
[aiogram.methods.pin\\_chat\\_message](#), 273  
[aiogram.methods.promote\\_chat\\_member](#), 274  
[aiogram.methods.reopen\\_forum\\_topic](#), 276  
[aiogram.methods.reopen\\_general\\_forum\\_topic](#), 277  
[aiogram.methods.restrict\\_chat\\_member](#), 278  
[aiogram.methods.revoke\\_chat\\_invite\\_link](#), 279  
[aiogram.methods.send\\_animation](#), 280  
[aiogram.methods.send\\_audio](#), 282  
[aiogram.methods.send\\_chat\\_action](#), 285  
[aiogram.methods.send\\_contact](#), 286  
[aiogram.methods.send\\_dice](#), 288  
[aiogram.methods.send\\_document](#), 289  
[aiogram.methods.send\\_game](#), 353  
[aiogram.methods.send\\_invoice](#), 334  
[aiogram.methods.send\\_location](#), 291  
[aiogram.methods.send\\_media\\_group](#), 293  
[aiogram.methods.send\\_message](#), 295  
[aiogram.methods.send\\_photo](#), 297  
[aiogram.methods.send\\_poll](#), 299  
[aiogram.methods.send\\_sticker](#), 342  
[aiogram.methods.send\\_venue](#), 301  
[aiogram.methods.send\\_video](#), 303  
[aiogram.methods.send\\_video\\_note](#), 305  
[aiogram.methods.send\\_voice](#), 307  
[aiogram.methods.set\\_chat\\_administrator\\_custom\\_title](#), 309  
[aiogram.methods.set\\_chat\\_description](#), 310  
[aiogram.methods.set\\_chat\\_menu\\_button](#), 311  
[aiogram.methods.set\\_chat\\_permissions](#), 312  
[aiogram.methods.set\\_chat\\_photo](#), 313  
[aiogram.methods.set\\_chat\\_sticker\\_set](#), 314  
[aiogram.methods.set\\_chat\\_title](#), 315  
[aiogram.methods.set\\_custom\\_emoji\\_sticker\\_set\\_thumbnail](#), 344  
[aiogram.methods.set\\_game\\_score](#), 354  
[aiogram.methods.set\\_my\\_commands](#), 316  
[aiogram.methods.set\\_my\\_default\\_administrator\\_rights](#), 317  
[aiogram.methods.set\\_my\\_description](#), 319  
[aiogram.methods.set\\_my\\_name](#), 320  
[aiogram.methods.set\\_my\\_short\\_description](#), 321  
[aiogram.methods.set\\_passport\\_data\\_errors](#), 360  
[aiogram.methods.set\\_sticker\\_emoji\\_list](#), 345  
[aiogram.methods.set\\_sticker\\_keywords](#), 346  
[aiogram.methods.set\\_sticker\\_mask\\_position](#), 347  
[aiogram.methods.set\\_sticker\\_position\\_in\\_set](#), 348  
[aiogram.methods.set\\_sticker\\_set\\_thumbnail](#), 349  
[aiogram.methods.set\\_sticker\\_set\\_title](#), 350  
[aiogram.methods.set\\_webhook](#), 358  
[aiogram.methods.stop\\_message\\_live\\_location](#), 370  
[aiogram.methods.stop\\_poll](#), 371  
[aiogram.methods.unban\\_chat\\_member](#), 322  
[aiogram.methods.unban\\_chat\\_sender\\_chat](#), 323  
[aiogram.methods.unhide\\_general\\_forum\\_topic](#), 324  
[aiogram.methods.unpin\\_all\\_chat\\_messages](#), 325  
[aiogram.methods.unpin\\_all\\_forum\\_topic\\_messages](#), 326  
[aiogram.methods.unpin\\_all\\_general\\_forum\\_topic\\_messages](#), 327  
[aiogram.methods.unpin\\_chat\\_message](#), 328  
[aiogram.methods.upload\\_sticker\\_file](#), 351  
[aiogram.types.animation](#), 64  
[aiogram.types.audio](#), 64  
[aiogram.types.bot\\_command](#), 65  
[aiogram.types.bot\\_command\\_scope](#), 65  
[aiogram.types.bot\\_command\\_scope\\_all\\_chat\\_administrators](#), 66  
[aiogram.types.bot\\_command\\_scope\\_all\\_group\\_chats](#), 66  
[aiogram.types.bot\\_command\\_scope\\_all\\_private\\_chats](#), 67  
[aiogram.types.bot\\_command\\_scope\\_chat](#), 67  
[aiogram.types.bot\\_command\\_scope\\_chat\\_administrators](#), 67  
[aiogram.types.bot\\_command\\_scope\\_chat\\_member](#), 68  
[aiogram.types.bot\\_command\\_scope\\_default](#), 69  
[aiogram.types.bot\\_description](#), 69  
[aiogram.types.bot\\_name](#), 69  
[aiogram.types.bot\\_short\\_description](#), 69  
[aiogram.types.callback\\_game](#), 235  
[aiogram.types.callback\\_query](#), 70  
[aiogram.types.chat](#), 71  
[aiogram.types.chat\\_administrator\\_rights](#), 84  
[aiogram.types.chat\\_invite\\_link](#), 85  
[aiogram.types.chat\\_join\\_request](#), 86  
[aiogram.types.chat\\_location](#), 119  
[aiogram.types.chat\\_member](#), 119  
[aiogram.types.chat\\_member\\_administrator](#), 120  
[aiogram.types.chat\\_member\\_banned](#), 121  
[aiogram.types.chat\\_member\\_left](#), 122  
[aiogram.types.chat\\_member\\_member](#), 122  
[aiogram.types.chat\\_member\\_owner](#), 122



---

- aiogram.types.chat\_member\_restricted, 123
- aiogram.types.chat\_member\_updated, 124
- aiogram.types.chat\_permissions, 141
- aiogram.types.chat\_photo, 142
- aiogram.types.chat\_shared, 142
- aiogram.types.chosen\_inline\_result, 17
- aiogram.types.contact, 143
- aiogram.types.dice, 143
- aiogram.types.document, 144
- aiogram.types.encrypted\_credentials, 215
- aiogram.types.encrypted\_passport\_element, 216
- aiogram.types.error\_event, 436
- aiogram.types.file, 144
- aiogram.types.force\_reply, 145
- aiogram.types.forum\_topic, 145
- aiogram.types.forum\_topic\_closed, 146
- aiogram.types.forum\_topic\_created, 146
- aiogram.types.forum\_topic\_edited, 146
- aiogram.types.forum\_topic\_reopened, 147
- aiogram.types.game, 235
- aiogram.types.game\_high\_score, 236
- aiogram.types.general\_forum\_topic\_hidden, 147
- aiogram.types.general\_forum\_topic\_unhidden, 147
- aiogram.types.inline\_keyboard\_button, 147
- aiogram.types.inline\_keyboard\_markup, 148
- aiogram.types.inline\_query, 18
- aiogram.types.inline\_query\_result, 19
- aiogram.types.inline\_query\_result\_article, 20
- aiogram.types.inline\_query\_result\_audio, 21
- aiogram.types.inline\_query\_result\_cached\_audio, 23
- aiogram.types.inline\_query\_result\_cached\_document, 25
- aiogram.types.inline\_query\_result\_cached\_gif, 27
- aiogram.types.inline\_query\_result\_cached\_mpeg4\_gif, 29
- aiogram.types.inline\_query\_result\_cached\_photo, 32
- aiogram.types.inline\_query\_result\_cached\_sticker, 34
- aiogram.types.inline\_query\_result\_cached\_video, 35
- aiogram.types.inline\_query\_result\_cached\_voice, 37
- aiogram.types.inline\_query\_result\_contact, 39
- aiogram.types.inline\_query\_result\_document, 41
- aiogram.types.inline\_query\_result\_game, 43
- aiogram.types.inline\_query\_result\_gif, 44
- aiogram.types.inline\_query\_result\_location, 45
- aiogram.types.inline\_query\_result\_mpeg4\_gif, 48
- aiogram.types.inline\_query\_result\_photo, 50
- aiogram.types.inline\_query\_result\_venue, 51
- aiogram.types.inline\_query\_result\_video, 53
- aiogram.types.inline\_query\_result\_voice, 56
- aiogram.types.inline\_query\_results\_button, 57
- aiogram.types.input\_contact\_message\_content, 57
- aiogram.types.input\_file, 149
- aiogram.types.input\_invoice\_message\_content, 58
- aiogram.types.input\_location\_message\_content, 61
- aiogram.types.input\_media, 149
- aiogram.types.input\_media\_animation, 150
- aiogram.types.input\_media\_audio, 151
- aiogram.types.input\_media\_document, 152
- aiogram.types.input\_media\_photo, 153
- aiogram.types.input\_media\_video, 153
- aiogram.types.input\_message\_content, 62
- aiogram.types.input\_sticker, 227
- aiogram.types.input\_text\_message\_content, 62
- aiogram.types.input\_venue\_message\_content, 63
- aiogram.types.invoice, 230
- aiogram.types.keyboard\_button, 155
- aiogram.types.keyboard\_button\_poll\_type, 156
- aiogram.types.keyboard\_button\_request\_chat, 156
- aiogram.types.keyboard\_button\_request\_user, 157
- aiogram.types.labeled\_price, 231
- aiogram.types.location, 158
- aiogram.types.login\_url, 158
- aiogram.types.mask\_position, 228
- aiogram.types.menu\_button, 159
- aiogram.types.menu\_button\_commands, 159
- aiogram.types.menu\_button\_default, 160
- aiogram.types.menu\_button\_web\_app, 160
- aiogram.types.message, 160
- aiogram.types.message\_auto\_delete\_timer\_changed, 203
- aiogram.types.message\_entity, 204
- aiogram.types.message\_id, 204
- aiogram.types.order\_info, 231
- aiogram.types.passport\_data, 217
- aiogram.types.passport\_element\_error, 217
- aiogram.types.passport\_element\_error\_data\_field, 218
- aiogram.types.passport\_element\_error\_file, 219
- aiogram.types.passport\_element\_error\_files, 219

`aiogram.types.passport_element_error_front_side`,  
220  
`aiogram.types.passport_element_error_reverse_side`,  
221  
`aiogram.types.passport_element_error_selfie`,  
221  
`aiogram.types.passport_element_error_translation_file`,  
222  
`aiogram.types.passport_element_error_translation_files`,  
223  
`aiogram.types.passport_element_error_unspecified`,  
224  
`aiogram.types.passport_file`, 224  
`aiogram.types.photo_size`, 205  
`aiogram.types.poll`, 205  
`aiogram.types.poll_answer`, 206  
`aiogram.types.poll_option`, 206  
`aiogram.types.pre_checkout_query`, 231  
`aiogram.types.proximity_alert_triggered`, 207  
`aiogram.types.reply_keyboard_markup`, 207  
`aiogram.types.reply_keyboard_remove`, 208  
`aiogram.types.response_parameters`, 208  
`aiogram.types.sent_web_app_message`, 63  
`aiogram.types.shipping_address`, 232  
`aiogram.types.shipping_option`, 233  
`aiogram.types.shipping_query`, 233  
`aiogram.types.sticker`, 228  
`aiogram.types.sticker_set`, 230  
`aiogram.types.story`, 209  
`aiogram.types.successful_payment`, 234  
`aiogram.types.switch_inline_query_chosen_chat`,  
209  
`aiogram.types.update`, 225  
`aiogram.types.user`, 210  
`aiogram.types.user_profile_photos`, 211  
`aiogram.types.user_shared`, 211  
`aiogram.types.venue`, 212  
`aiogram.types.video`, 212  
`aiogram.types.video_chat_ended`, 213  
`aiogram.types.video_chat_participants_invited`,  
213  
`aiogram.types.video_chat_scheduled`, 213  
`aiogram.types.video_chat_started`, 214  
`aiogram.types.video_note`, 214  
`aiogram.types.voice`, 214  
`aiogram.types.web_app_data`, 215  
`aiogram.types.web_app_info`, 215  
`aiogram.types.webhook_info`, 226  
`aiogram.types.write_access_allowed`, 215

## Symbols

- `__call__()` (*aiogram.dispatcher.middlewares.base.BaseMiddleware* method), 434
  - `__call__()` (*aiogram.filters.base.Filter* method), 412
  - `__init__()` (*aiogram.dispatcher.dispatcher.Dispatcher* method), 398
  - `__init__()` (*aiogram.dispatcher.router.Router* method), 393
  - `__init__()` (*aiogram.filters.command.Command* method), 402
  - `__init__()` (*aiogram.fsm.storage.memory.MemoryStorage* method), 431
  - `__init__()` (*aiogram.fsm.storage.redis.RedisStorage* method), 431
  - `__init__()` (*aiogram.types.input\_file.BufferedInputFile* method), 391
  - `__init__()` (*aiogram.types.input\_file.FSInputFile* method), 391
  - `__init__()` (*aiogram.utils.callback\_answer.CallbackAnswer* method), 460
  - `__init__()` (*aiogram.utils.callback\_answer.CallbackAnswerMiddleware* method), 459
  - `__init__()` (*aiogram.utils.chat\_action.ChatActionSender* method), 453
  - `__init__()` (*aiogram.utils.formatting.Text* method), 464
  - `__init__()` (*aiogram.utils.i18n.middleware.ConstI18nMiddleware* method), 450
  - `__init__()` (*aiogram.utils.i18n.middleware.FSMI18nMiddleware* method), 450
  - `__init__()` (*aiogram.utils.i18n.middleware.I18nMiddleware* method), 451
  - `__init__()` (*aiogram.utils.i18n.middleware.SimpleI18nMiddleware* method), 450
  - `__init__()` (*aiogram.utils.keyboard.InlineKeyboardBuilder* method), 446
  - `__init__()` (*aiogram.utils.keyboard.ReplyKeyboardBuilder* method), 447
  - `__init__()` (*aiogram.webhook.aihttp\_server.BaseRequestHandler* method), 416
  - `__init__()` (*aiogram.webhook.aihttp\_server.SimpleRequestHandler* method), 416
  - `__init__()` (*aiogram.webhook.aihttp\_server.TokenBasedRequestHandler* method), 417
  - `__init__()` (*aiogram.webhook.security.IPFilter* method), 418
- ## A
- `action()` (*aiogram.methods.send\_chat\_action.SendChatAction* attribute), 285
  - `active_usernames` (*aiogram.types.chat.Chat* attribute), 72
  - `add()` (*aiogram.utils.keyboard.InlineKeyboardBuilder* method), 446
  - `add()` (*aiogram.utils.keyboard.ReplyKeyboardBuilder* method), 447
  - `added_to_attachment_menu` (*aiogram.types.user.User* attribute), 210
  - `ADDRESS` (*aiogram.enums.encrypted\_passport\_element.EncryptedPassportElement* attribute), 383
  - `address` (*aiogram.methods.sendVenue.SendVenue* attribute), 301
  - `address` (*aiogram.types.chat\_location.ChatLocation* attribute), 119
  - `address` (*aiogram.types.inline\_query\_result\_venue.InlineQueryResultVenue* attribute), 52
  - `address` (*aiogram.types.input\_venue\_message\_content.InputVenueMessageContent* attribute), 63
  - `address` (*aiogram.types.venue.Venue* attribute), 212
  - `AddStickerToSet` (class in *aiogram.methods.add\_sticker\_to\_set*), 337
  - `adjust()` (*aiogram.utils.keyboard.InlineKeyboardBuilder* method), 446
  - `adjust()` (*aiogram.utils.keyboard.ReplyKeyboardBuilder* method), 447
  - `ADMINISTRATOR` (*aiogram.enums.chat\_member\_status.ChatMemberStatus* attribute), 377
  - `AED` (*aiogram.enums.currency.Currency* attribute), 380
  - `AFN` (*aiogram.enums.currency.Currency* attribute), 380
  - `aiogram.dispatcher.flags` module, 438
  - `aiogram.enums.bot_command_scope_type` module, 376
  - `aiogram.enums.chat_action` module, 377

aiogram.enums.chat_member_status module, 377	aiogram.methods.ban_chat_member module, 238
aiogram.enums.chat_type module, 378	aiogram.methods.ban_chat_sender_chat module, 240
aiogram.enums.content_type module, 378	aiogram.methods.close module, 241
aiogram.enums.currency module, 380	aiogram.methods.close_forum_topic module, 242
aiogram.enums.dice_emoji module, 383	aiogram.methods.close_general_forum_topic module, 243
aiogram.enums.encrypted_passport_element module, 383	aiogram.methods.copy_message module, 243
aiogram.enums.inline_query_result_type module, 384	aiogram.methods.create_chat_invite_link module, 245
aiogram.enums.input_media_type module, 384	aiogram.methods.create_forum_topic module, 247
aiogram.enums.mask_position_point module, 385	aiogram.methods.create_invoice_link module, 331
aiogram.enums.menu_button_type module, 385	aiogram.methods.create_new_sticker_set module, 338
aiogram.enums.message_entity_type module, 385	aiogram.methods.decline_chat_join_request module, 248
aiogram.enums.parse_mode module, 386	aiogram.methods.delete_chat_photo module, 249
aiogram.enums.passport_element_error_type module, 386	aiogram.methods.delete_chat_sticker_set module, 250
aiogram.enums.poll_type module, 387	aiogram.methods.delete_forum_topic module, 251
aiogram.enums.sticker_format module, 387	aiogram.methods.delete_message module, 361
aiogram.enums.sticker_type module, 387	aiogram.methods.delete_my_commands module, 252
aiogram.enums.topic_icon_color module, 387	aiogram.methods.delete_sticker_from_set module, 339
aiogram.enums.update_type module, 388	aiogram.methods.delete_sticker_set module, 340
aiogram.exceptions module, 437	aiogram.methods.delete_webhook module, 356
aiogram.handlers.callback_query module, 440	aiogram.methods.edit_chat_invite_link module, 253
aiogram.methods.add_sticker_to_set module, 337	aiogram.methods.edit_forum_topic module, 254
aiogram.methods.answer_callback_query module, 236	aiogram.methods.edit_general_forum_topic module, 255
aiogram.methods.answer_inline_query module, 372	aiogram.methods.edit_message_caption module, 362
aiogram.methods.answer_pre_checkout_query module, 329	aiogram.methods.edit_message_live_location module, 364
aiogram.methods.answer_shipping_query module, 330	aiogram.methods.edit_message_media module, 366
aiogram.methods.answer_web_app_query module, 375	aiogram.methods.edit_message_reply_markup module, 367
aiogram.methods.approve_chat_join_request module, 237	aiogram.methods.edit_message_text module, 368

aiogram.methods.export_chat_invite_link module, 256	aiogram.methods.reopen_general_forum_topic module, 277
aiogram.methods.forward_message module, 257	aiogram.methods.restrict_chat_member module, 278
aiogram.methods.get_chat module, 259	aiogram.methods.revoke_chat_invite_link module, 279
aiogram.methods.get_chat_administrators module, 259	aiogram.methods.send_animation module, 280
aiogram.methods.get_chat_member module, 260	aiogram.methods.send_audio module, 282
aiogram.methods.get_chat_member_count module, 261	aiogram.methods.send_chat_action module, 285
aiogram.methods.get_chat_menu_button module, 262	aiogram.methods.send_contact module, 286
aiogram.methods.get_custom_emoji_stickers module, 341	aiogram.methods.send_dice module, 288
aiogram.methods.get_file module, 263	aiogram.methods.send_document module, 289
aiogram.methods.get_forum_topic_icon_stickers module, 264	aiogram.methods.send_game module, 353
aiogram.methods.get_game_high_scores module, 352	aiogram.methods.send_invoice module, 334
aiogram.methods.get_me module, 264	aiogram.methods.send_location module, 291
aiogram.methods.get_my_commands module, 265	aiogram.methods.send_media_group module, 293
aiogram.methods.get_my_default_administrator_rights module, 266	aiogram.methods.send_message module, 295
aiogram.methods.get_my_description module, 267	aiogram.methods.send_photo module, 297
aiogram.methods.get_my_name module, 268	aiogram.methods.send_poll module, 299
aiogram.methods.get_my_short_description module, 268	aiogram.methods.send_sticker module, 342
aiogram.methods.get_sticker_set module, 342	aiogram.methods.send_venue module, 301
aiogram.methods.get_updates module, 356	aiogram.methods.send_video module, 303
aiogram.methods.get_user_profile_photos module, 269	aiogram.methods.send_video_note module, 305
aiogram.methods.get_webhook_info module, 358	aiogram.methods.send_voice module, 307
aiogram.methods.hide_general_forum_topic module, 270	aiogram.methods.set_chat_administrator_custom_title module, 309
aiogram.methods.leave_chat module, 271	aiogram.methods.set_chat_description module, 310
aiogram.methods.log_out module, 272	aiogram.methods.set_chat_menu_button module, 311
aiogram.methods.pin_chat_message module, 273	aiogram.methods.set_chat_permissions module, 312
aiogram.methods.promote_chat_member module, 274	aiogram.methods.set_chat_photo module, 313
aiogram.methods.reopen_forum_topic module, 276	aiogram.methods.set_chat_sticker_set module, 314

aiogram.methods.set_chat_title	aiogram.types.audio
module, 315	module, 64
aiogram.methods.set_custom_emoji_sticker_set_thumbnail	aiogram.types.bot_command
module, 344	module, 65
aiogram.methods.set_game_score	aiogram.types.bot_command_scope
module, 354	module, 65
aiogram.methods.set_my_commands	aiogram.types.bot_command_scope_all_chat_administrators
module, 316	module, 66
aiogram.methods.set_my_default_administrator_rights	aiogram.types.bot_command_scope_all_group_chats
module, 317	module, 66
aiogram.methods.set_my_description	aiogram.types.bot_command_scope_all_private_chats
module, 319	module, 67
aiogram.methods.set_my_name	aiogram.types.bot_command_scope_chat
module, 320	module, 67
aiogram.methods.set_my_short_description	aiogram.types.bot_command_scope_chat_administrators
module, 321	module, 67
aiogram.methods.set_passport_data_errors	aiogram.types.bot_command_scope_chat_member
module, 360	module, 68
aiogram.methods.set_sticker_emoji_list	aiogram.types.bot_command_scope_default
module, 345	module, 69
aiogram.methods.set_sticker_keywords	aiogram.types.bot_description
module, 346	module, 69
aiogram.methods.set_sticker_mask_position	aiogram.types.bot_name
module, 347	module, 69
aiogram.methods.set_sticker_position_in_set	aiogram.types.bot_short_description
module, 348	module, 69
aiogram.methods.set_sticker_set_thumbnail	aiogram.types.callback_game
module, 349	module, 235
aiogram.methods.set_sticker_set_title	aiogram.types.callback_query
module, 350	module, 70
aiogram.methods.set_webhook	aiogram.types.chat
module, 358	module, 71
aiogram.methods.stop_message_live_location	aiogram.types.chat_administrator_rights
module, 370	module, 84
aiogram.methods.stop_poll	aiogram.types.chat_invite_link
module, 371	module, 85
aiogram.methods.unban_chat_member	aiogram.types.chat_join_request
module, 322	module, 86
aiogram.methods.unban_chat_sender_chat	aiogram.types.chat_location
module, 323	module, 119
aiogram.methods.unhide_general_forum_topic	aiogram.types.chat_member
module, 324	module, 119
aiogram.methods.unpin_all_chat_messages	aiogram.types.chat_member_administrator
module, 325	module, 120
aiogram.methods.unpin_all_forum_topic_messages	aiogram.types.chat_member_banned
module, 326	module, 121
aiogram.methods.unpin_all_general_forum_topic_messages	aiogram.types.chat_member_left
module, 327	module, 122
aiogram.methods.unpin_chat_message	aiogram.types.chat_member_member
module, 328	module, 122
aiogram.methods.upload_sticker_file	aiogram.types.chat_member_owner
module, 351	module, 122
aiogram.types.animation	aiogram.types.chat_member_restricted
module, 64	module, 123

aiogram.types.chat\_member\_updated  
module, 124

aiogram.types.chat\_permissions  
module, 141

aiogram.types.chat\_photo  
module, 142

aiogram.types.chat\_shared  
module, 142

aiogram.types.chosen\_inline\_result  
module, 17

aiogram.types.contact  
module, 143

aiogram.types.dice  
module, 143

aiogram.types.document  
module, 144

aiogram.types.encrypted\_credentials  
module, 215

aiogram.types.encrypted\_passport\_element  
module, 216

aiogram.types.error\_event  
module, 436

aiogram.types.file  
module, 144

aiogram.types.force\_reply  
module, 145

aiogram.types.forum\_topic  
module, 145

aiogram.types.forum\_topic\_closed  
module, 146

aiogram.types.forum\_topic\_created  
module, 146

aiogram.types.forum\_topic\_edited  
module, 146

aiogram.types.forum\_topic\_reopened  
module, 147

aiogram.types.game  
module, 235

aiogram.types.game\_high\_score  
module, 236

aiogram.types.general\_forum\_topic\_hidden  
module, 147

aiogram.types.general\_forum\_topic\_unhidden  
module, 147

aiogram.types.inline\_keyboard\_button  
module, 147

aiogram.types.inline\_keyboard\_markup  
module, 148

aiogram.types.inline\_query  
module, 18

aiogram.types.inline\_query\_result  
module, 19

aiogram.types.inline\_query\_result\_article  
module, 20

aiogram.types.inline\_query\_result\_audio  
module, 21

aiogram.types.inline\_query\_result\_cached\_audio  
module, 23

aiogram.types.inline\_query\_result\_cached\_document  
module, 25

aiogram.types.inline\_query\_result\_cached\_gif  
module, 27

aiogram.types.inline\_query\_result\_cached\_mpeg4\_gif  
module, 29

aiogram.types.inline\_query\_result\_cached\_photo  
module, 32

aiogram.types.inline\_query\_result\_cached\_sticker  
module, 34

aiogram.types.inline\_query\_result\_cached\_video  
module, 35

aiogram.types.inline\_query\_result\_cached\_voice  
module, 37

aiogram.types.inline\_query\_result\_contact  
module, 39

aiogram.types.inline\_query\_result\_document  
module, 41

aiogram.types.inline\_query\_result\_game  
module, 43

aiogram.types.inline\_query\_result\_gif  
module, 44

aiogram.types.inline\_query\_result\_location  
module, 45

aiogram.types.inline\_query\_result\_mpeg4\_gif  
module, 48

aiogram.types.inline\_query\_result\_photo  
module, 50

aiogram.types.inline\_query\_result\_venue  
module, 51

aiogram.types.inline\_query\_result\_video  
module, 53

aiogram.types.inline\_query\_result\_voice  
module, 56

aiogram.types.inline\_query\_results\_button  
module, 57

aiogram.types.input\_contact\_message\_content  
module, 57

aiogram.types.input\_file  
module, 149

aiogram.types.input\_invoice\_message\_content  
module, 58

aiogram.types.input\_location\_message\_content  
module, 61

aiogram.types.input\_media  
module, 149

aiogram.types.input\_media\_animation  
module, 150

aiogram.types.input\_media\_audio  
module, 151



aiogram.types.input_media_document module, 152	aiogram.types.passport_element_error_data_field module, 218
aiogram.types.input_media_photo module, 153	aiogram.types.passport_element_error_file module, 219
aiogram.types.input_media_video module, 153	aiogram.types.passport_element_error_files module, 219
aiogram.types.input_message_content module, 62	aiogram.types.passport_element_error_front_side module, 220
aiogram.types.input_sticker module, 227	aiogram.types.passport_element_error_reverse_side module, 221
aiogram.types.input_text_message_content module, 62	aiogram.types.passport_element_error_selfie module, 221
aiogram.types.input_venue_message_content module, 63	aiogram.types.passport_element_error_translation_file module, 222
aiogram.types.invoice module, 230	aiogram.types.passport_element_error_translation_files module, 223
aiogram.types.keyboard_button module, 155	aiogram.types.passport_element_error_unspecified module, 224
aiogram.types.keyboard_button_poll_type module, 156	aiogram.types.passport_file module, 224
aiogram.types.keyboard_button_request_chat module, 156	aiogram.types.photo_size module, 205
aiogram.types.keyboard_button_request_user module, 157	aiogram.types.poll module, 205
aiogram.types.labeled_price module, 231	aiogram.types.poll_answer module, 206
aiogram.types.location module, 158	aiogram.types.poll_option module, 206
aiogram.types.login_url module, 158	aiogram.types.pre_checkout_query module, 231
aiogram.types.mask_position module, 228	aiogram.types.proximity_alert_triggered module, 207
aiogram.types.menu_button module, 159	aiogram.types.reply_keyboard_markup module, 207
aiogram.types.menu_button_commands module, 159	aiogram.types.reply_keyboard_remove module, 208
aiogram.types.menu_button_default module, 160	aiogram.types.response_parameters module, 208
aiogram.types.menu_button_web_app module, 160	aiogram.types.sent_web_app_message module, 63
aiogram.types.message module, 160	aiogram.types.shipping_address module, 232
aiogram.types.message_auto_delete_timer_changed module, 203	aiogram.types.shipping_option module, 233
aiogram.types.message_entity module, 204	aiogram.types.shipping_query module, 233
aiogram.types.message_id module, 204	aiogram.types.sticker module, 228
aiogram.types.order_info module, 231	aiogram.types.sticker_set module, 230
aiogram.types.passport_data module, 217	aiogram.types.story module, 209
aiogram.types.passport_element_error module, 217	aiogram.types.successful_payment module, 234



aiogram.types.switch\_inline\_query\_chosen\_chat (attribute), 244  
aiogram.types.update module, 209  
aiogram.types.user module, 210  
aiogram.types.user\_profile\_photos module, 211  
aiogram.types.user\_shared module, 211  
aiogram.types.venue module, 212  
aiogram.types.video module, 212  
aiogram.types.video\_chat\_ended module, 213  
aiogram.types.video\_chat\_participants\_invited module, 213  
aiogram.types.video\_chat\_scheduled module, 213  
aiogram.types.video\_chat\_started module, 214  
aiogram.types.video\_note module, 214  
aiogram.types.voice module, 214  
aiogram.types.web\_app\_data module, 215  
aiogram.types.web\_app\_info module, 215  
aiogram.types.webhook\_info module, 226  
aiogram.types.write\_access\_allowed module, 215  
AiogramError, 437  
AiohttpSession (class in aiogram.client.session.aiohttp), 14  
ALL (aiogram.enums.currency.Currency attribute), 380  
ALL\_CHAT\_ADMINISTRATORS (aiogram.enums.bot\_command\_scope\_type.BotCommandScopeType attribute), 376  
ALL\_GROUP\_CHATS (aiogram.enums.bot\_command\_scope\_type.BotCommandScopeType attribute), 376  
ALL\_PRIVATE\_CHATS (aiogram.enums.bot\_command\_scope\_type.BotCommandScopeType attribute), 376  
allow\_bot\_chats (aiogram.types.switch\_inline\_query\_chosen\_chat.SwitchInlineQueryChosenChat attribute), 209  
allow\_channel\_chats (aiogram.types.switch\_inline\_query\_chosen\_chat.SwitchInlineQueryChosenChat attribute), 209  
allow\_group\_chats (aiogram.types.switch\_inline\_query\_chosen\_chat.SwitchInlineQueryChosenChat attribute), 209  
allow\_sending\_without\_reply (aiogram.methods.copy\_message.CopyMessage attribute), 209  
allow\_sending\_without\_reply (aiogram.methods.send\_animation.SendAnimation attribute), 281  
allow\_sending\_without\_reply (aiogram.methods.send\_audio.SendAudio attribute), 284  
allow\_sending\_without\_reply (aiogram.methods.send\_contact.SendContact attribute), 287  
allow\_sending\_without\_reply (aiogram.methods.send\_dice.SendDice attribute), 288  
allow\_sending\_without\_reply (aiogram.methods.send\_document.SendDocument attribute), 290  
allow\_sending\_without\_reply (aiogram.methods.send\_game.SendGame attribute), 353  
allow\_sending\_without\_reply (aiogram.methods.send\_invoice.SendInvoice attribute), 336  
allow\_sending\_without\_reply (aiogram.methods.send\_location.SendLocation attribute), 292  
allow\_sending\_without\_reply (aiogram.methods.send\_media\_group.SendMediaGroup attribute), 294  
allow\_sending\_without\_reply (aiogram.methods.send\_message.SendMessage attribute), 296  
allow\_sending\_without\_reply (aiogram.methods.send\_photo.SendPhoto attribute), 298  
allow\_sending\_without\_reply (aiogram.methods.send\_poll.SendPoll attribute), 300  
allow\_sending\_without\_reply (aiogram.methods.send\_sticker.SendSticker attribute), 343  
allow\_sending\_without\_reply (aiogram.methods.send\_venue.SendVenue attribute), 302  
allow\_sending\_without\_reply (aiogram.methods.send\_video.SendVideo attribute), 306  
allow\_sending\_without\_reply (aiogram.methods.send\_video\_note.SendVideoNote attribute), 306  
allow\_sending\_without\_reply (aiogram.methods.send\_voice.SendVoice attribute), 308  
allow\_user\_chats (aiogram.types.switch\_inline\_query\_chosen\_chat.SwitchInlineQueryChosenChat attribute), 209

allowed\_updates (*aiogram.methods.get\_updates.GetUpdates* method), 169  
     *attribute*), 357  
 allowed\_updates (*aiogram.methods.set\_webhook.SetWebhook* method), 92  
     *attribute*), 359  
 allowed\_updates (*aiogram.types.webhook\_info.WebhookInfo* method), 93  
     *attribute*), 227  
 allows\_multiple\_answers  
     (*aiogram.methods.send\_poll.SendPoll* *attribute*), 299  
 allows\_multiple\_answers (*aiogram.types.poll.Poll* *attribute*), 205  
 AMD (*aiogram.enums.currency.Currency* *attribute*), 380  
 amount (*aiogram.types.labeled\_price.LabeledPrice* *attribute*), 231  
 ANIMATED (*aiogram.enums.sticker\_format.StickerFormat* *attribute*), 387  
 ANIMATION (*aiogram.enums.content\_type.ContentType* *attribute*), 378  
 ANIMATION (*aiogram.enums.input\_media\_type.InputMediaType* *attribute*), 384  
 animation (*aiogram.methods.send\_animation.SendAnimation* *attribute*), 280  
 animation (*aiogram.types.game.Game* *attribute*), 235  
 animation (*aiogram.types.message.Message* *attribute*), 163  
 Animation (class in *aiogram.types.animation*), 64  
 answer() (*aiogram.types.callback\_query.CallbackQuery* method), 70  
 answer() (*aiogram.types.chat\_join\_request.ChatJoinRequest* method), 87  
 answer() (*aiogram.types.chat\_member\_updated.ChatMemberUpdated* method), 125  
 answer() (*aiogram.types.inline\_query.InlineQuery* method), 18  
 answer() (*aiogram.types.message.Message* method), 182  
 answer() (*aiogram.types.pre\_checkout\_query.PreCheckoutQuery* method), 232  
 answer() (*aiogram.types.shipping\_query.ShippingQuery* method), 233  
 answer\_animation() (*aiogram.types.chat\_join\_request.ChatJoinRequest* method), 88  
 answer\_animation() (*aiogram.types.chat\_member\_updated.ChatMemberUpdated* method), 126  
 answer\_animation() (*aiogram.types.message.Message* method), 167  
 answer\_animation\_pm() (*aiogram.types.chat\_join\_request.ChatJoinRequest* method), 90  
 answer\_audio() (*aiogram.types.chat\_join\_request.ChatJoinRequest* method), 91  
 answer\_audio() (*aiogram.types.chat\_member\_updated.ChatMemberUpdated* method), 127  
 answer\_audio() (*aiogram.types.message.Message* method), 169  
     *answer\_audio\_pm()* (*aiogram.types.chat\_join\_request.ChatJoinRequest* method), 92  
     *answer\_contact()* (*aiogram.types.chat\_join\_request.ChatJoinRequest* method), 93  
     *answer\_contact()* (*aiogram.types.chat\_member\_updated.ChatMemberUpdated* method), 128  
     *answer\_contact()* (*aiogram.types.message.Message* method), 171  
     *answer\_contact\_pm()* (*aiogram.types.chat\_join\_request.ChatJoinRequest* method), 94  
     *answer\_dice()* (*aiogram.types.chat\_join\_request.ChatJoinRequest* method), 108  
     *answer\_dice()* (*aiogram.types.chat\_member\_updated.ChatMemberUpdated* method), 135  
     *answer\_dice()* (*aiogram.types.message.Message* method), 187  
     *answer\_dice\_pm()* (*aiogram.types.chat\_join\_request.ChatJoinRequest* method), 109  
     *answer\_document()* (*aiogram.types.chat\_join\_request.ChatJoinRequest* method), 94  
     *answer\_document()* (*aiogram.types.chat\_member\_updated.ChatMemberUpdated* method), 129  
     *answer\_document()* (*aiogram.types.message.Message* method), 173  
     *answer\_document\_pm()* (*aiogram.types.chat\_join\_request.ChatJoinRequest* method), 96  
     *answer\_game()* (*aiogram.types.chat\_join\_request.ChatJoinRequest* method), 97  
     *answer\_game()* (*aiogram.types.chat\_member\_updated.ChatMemberUpdated* method), 130  
     *answer\_game()* (*aiogram.types.message.Message* method), 174  
     *answer\_game\_pm()* (*aiogram.types.chat\_join\_request.ChatJoinRequest* method), 97  
     *answer\_invoice()* (*aiogram.types.chat\_join\_request.ChatJoinRequest* method), 98  
     *answer\_invoice()* (*aiogram.types.chat\_member\_updated.ChatMemberUpdated* method), 130  
     *answer\_invoice()* (*aiogram.types.message.Message* method), 179  
     *answer\_invoice\_pm()* (*aiogram.types.chat\_join\_request.ChatJoinRequest* method), 99  
     *answer\_location()* (*aiogram.types.chat\_join\_request.ChatJoinRequest* method), 101  
     *answer\_location()* (*aiogram.types.chat\_member\_updated.ChatMemberUpdated* method), 132  
     *answer\_location()* (*aiogram.types.message.Message* method), 179  
     *answer\_location\_pm()* (*aiogram.types.chat\_join\_request.ChatJoinRequest* method), 101

`method`), 102  
`answer_media_group()`  
( `aiogram.types.chat_join_request.ChatJoinRequest`  
`method`), 103  
`answer_media_group()`  
( `aiogram.types.chat_member_updated.ChatMemberUpdated`  
`method`), 139  
`method`), 133  
`answer_media_group()`  
( `aiogram.types.message.Message` `method`),  
180  
`answer_media_group_pm()`  
( `aiogram.types.chat_join_request.ChatJoinRequest`  
`method`), 104  
`answer_photo()` ( `aiogram.types.chat_join_request.ChatJoinRequest`  
`method`), 114  
`method`), 104  
`answer_photo()` ( `aiogram.types.chat_member_updated.ChatMemberUpdated`  
`method`), 133  
`method`), 133  
`answer_photo()` ( `aiogram.types.message.Message`  
`method`), 183  
`answer_photo_pm()` ( `aiogram.types.chat_join_request.ChatJoinRequest`  
`method`), 105  
`answer_pm()` ( `aiogram.types.chat_join_request.ChatJoinRequest` `method`), 118  
`method`), 88  
`answer_poll()` ( `aiogram.types.chat_join_request.ChatJoinRequest` `method`), 106  
`method`), 106  
`answer_poll()` ( `aiogram.types.chat_member_updated.ChatMemberUpdated`  
`method`), 134  
`method`), 134  
`answer_poll()` ( `aiogram.types.message.Message` `method`), 185  
`method`), 185  
`answer_poll_pm()` ( `aiogram.types.chat_join_request.ChatJoinRequest`  
`method`), 107  
`method`), 107  
`answer_sticker()` ( `aiogram.types.chat_join_request.ChatJoinRequest`  
`method`), 110  
`method`), 110  
`answer_sticker()` ( `aiogram.types.chat_member_updated.ChatMemberUpdated`  
`method`), 136  
`method`), 136  
`answer_sticker()` ( `aiogram.types.message.Message`  
`method`), 188  
`method`), 188  
`answer_sticker_pm()`  
( `aiogram.types.chat_join_request.ChatJoinRequest`  
`method`), 110  
`method`), 110  
`answer_venue()` ( `aiogram.types.chat_join_request.ChatJoinRequest`  
`method`), 111  
`method`), 111  
`answer_venue()` ( `aiogram.types.chat_member_updated.ChatMemberUpdated`  
`method`), 137  
`method`), 137  
`answer_venue()` ( `aiogram.types.message.Message`  
`method`), 190  
`method`), 190  
`answer_venue_pm()` ( `aiogram.types.chat_join_request.ChatJoinRequest`  
`method`), 112  
`method`), 112  
`answer_video()` ( `aiogram.types.chat_join_request.ChatJoinRequest`  
`method`), 113  
`method`), 113  
`answer_video()` ( `aiogram.types.chat_member_updated.ChatMemberUpdated`  
`method`), 138  
`method`), 138  
`answer_video()` ( `aiogram.types.message.Message`  
`method`), 192  
`method`), 192  
`answer_video_note()`  
( `aiogram.types.chat_join_request.ChatJoinRequest`  
`method`), 115  
`method`), 115  
`answer_video_note()`  
( `aiogram.types.chat_member_updated.ChatMemberUpdated`  
`method`), 139  
`method`), 139  
`answer_video_note()`  
( `aiogram.types.message.Message` `method`),  
194  
`method`), 194  
`answer_video_note_pm()`  
( `aiogram.types.chat_join_request.ChatJoinRequest`  
`method`), 116  
`method`), 116  
`answer_video_pm()` ( `aiogram.types.chat_join_request.ChatJoinRequest`  
`method`), 114  
`method`), 114  
`answer_voice()` ( `aiogram.types.chat_join_request.ChatJoinRequest`  
`method`), 117  
`method`), 117  
`answer_voice()` ( `aiogram.types.chat_member_updated.ChatMemberUpdated`  
`method`), 140  
`method`), 140  
`answer_voice()` ( `aiogram.types.message.Message`  
`method`), 196  
`method`), 196  
`answer_voice_pm()` ( `aiogram.types.chat_join_request.ChatJoinRequest`  
`method`), 118  
`method`), 118  
`AnswerCallbackQuery` (class in  
`aiogram.methods.answer_callback_query`),  
236  
`method`), 236  
`AnswerInlineQuery` (class in  
`aiogram.methods.answer_inline_query`),  
322  
`method`), 322  
`AnswerPreCheckoutQuery` (class in  
`aiogram.methods.answer_pre_checkout_query`),  
329  
`method`), 329  
`AnswerShippingQuery` (class in  
`aiogram.methods.answer_shipping_query`),  
330  
`method`), 330  
`AnswerWebAppQuery` (class in  
`aiogram.methods.answer_web_app_query`),  
375  
`method`), 375  
`ANY` ( `aiogram.enums.content_type.ContentType` at-  
tribute), 378  
`api_url()` ( `aiogram.client.telegram.TelegramAPIServer`  
`method`), 12  
`method`), 12  
`approve()` ( `aiogram.types.chat_join_request.ChatJoinRequest`  
`method`), 86  
`method`), 86  
`ApproveChatJoinRequest` (class in  
`aiogram.methods.approve_chat_join_request`),  
237  
`method`), 237  
`args` ( `aiogram.filters.command.CommandObject` at-  
tribute), 403  
`method`), 403  
`ARS` ( `aiogram.enums.currency.Currency` attribute), 380  
`method`), 380  
`ARTICLE` ( `aiogram.enums.inline_query_result_type.InlineQueryResultType`  
attribute), 384  
`method`), 384  
`as_html()` ( `aiogram.utils.formatting.Text` `method`), 464  
`method`), 464

- as\_key\_value() (in module aiogram.utils.formatting), 462
- as\_kwargs() (aiogram.utils.formatting.Text method), 464
- as\_line() (in module aiogram.utils.formatting), 461
- as\_list() (in module aiogram.utils.formatting), 461
- as\_markdown() (aiogram.utils.formatting.Text method), 464
- as\_marked\_list() (in module aiogram.utils.formatting), 461
- as\_marked\_section() (in module aiogram.utils.formatting), 462
- as\_numbered\_list() (in module aiogram.utils.formatting), 462
- as\_numbered\_section() (in module aiogram.utils.formatting), 462
- as\_section() (in module aiogram.utils.formatting), 462
- AUD (aiogram.enums.currency.Currency attribute), 380
- AUDIO (aiogram.enums.content\_type.ContentType attribute), 378
- AUDIO (aiogram.enums.inline\_query\_result\_type.InlineQueryResultType attribute), 384
- AUDIO (aiogram.enums.input\_media\_type.InputMediaType attribute), 384
- audio (aiogram.methods.send\_audio.SendAudio attribute), 283
- audio (aiogram.types.message.Message attribute), 163
- Audio (class in aiogram.types.audio), 64
- audio\_duration (aiogram.types.inline\_query\_result\_audio.InlineQueryResultAudio attribute), 23
- audio\_file\_id (aiogram.types.inline\_query\_result\_cached\_audio.InlineQueryResultCachedAudio attribute), 25
- audio\_url (aiogram.types.inline\_query\_result\_audio.InlineQueryResultAudio attribute), 22
- auth\_date (aiogram.utils.web\_app.WebAppInitData attribute), 457
- author\_signature (aiogram.types.message.Message attribute), 163
- AZN (aiogram.enums.currency.Currency attribute), 380
- ## B
- BAM (aiogram.enums.currency.Currency attribute), 380
- ban() (aiogram.types.chat.Chat method), 82
- ban\_sender\_chat() (aiogram.types.chat.Chat method), 74
- BanChatMember (class in aiogram.methods.ban\_chat\_member), 238
- BanChatSenderChat (class in aiogram.methods.ban\_chat\_sender\_chat), 240
- BANK\_STATEMENT (aiogram.enums.encrypted\_passport\_element.EncryptedPassportElement attribute), 383
- base (aiogram.client.telegram.TelegramAPIServer attribute), 12
- BaseMiddleware (class in aiogram.dispatcher.middlewares.base), 434
- BaseRequestHandler (class in aiogram.webhook.aiohttp\_server), 416
- BaseSession (class in aiogram.client.session.base), 13
- BaseStorage (class in aiogram.fsm.storage.base), 432
- BASKETBALL (aiogram.enums.dice\_emoji.DiceEmoji attribute), 383
- BASKETBALL (aiogram.types.dice.DiceEmoji attribute), 143
- BDT (aiogram.enums.currency.Currency attribute), 380
- BGN (aiogram.enums.currency.Currency attribute), 380
- big\_file\_id (aiogram.types.chat\_photo.ChatPhoto attribute), 142
- big\_file\_unique\_id (aiogram.types.chat\_photo.ChatPhoto attribute), 142
- bio (aiogram.types.chat.Chat attribute), 72
- bio (aiogram.types.chat\_join\_request.ChatJoinRequest attribute), 86
- BLUE (aiogram.enums.topic\_icon\_color.TopicIconColor attribute), 387
- BND (aiogram.enums.currency.Currency attribute), 380
- BOB (aiogram.enums.currency.Currency attribute), 380
- BOLD (aiogram.enums.message\_entity\_type.MessageEntityType attribute), 385
- Bold (class in aiogram.utils.formatting), 466
- Bot (class in aiogram.client.bot), 389, 390
- bot\_administrator\_rights (aiogram.types.keyboard\_button\_request\_chat.KeyboardButtonRequestChat attribute), 157
- BOT\_COMMAND\_QUERY\_RESULT\_CACHED\_AUDIO (aiogram.enums.message\_entity\_type.MessageEntityType attribute), 385
- bot\_username (aiogram.types.login\_url.LoginUrl attribute), 158
- BotCommand (class in aiogram.types.bot\_command), 65
- BotCommand (class in aiogram.utils.formatting), 465
- BotCommandScope (class in aiogram.types.bot\_command\_scope), 65
- BotCommandScopeAllChatAdministrators (class in aiogram.types.bot\_command\_scope\_all\_chat\_administrators), 66
- BotCommandScopeAllGroupChats (class in aiogram.types.bot\_command\_scope\_all\_group\_chats), 66
- BotCommandScopeAllPrivateChats (class in aiogram.types.bot\_command\_scope\_all\_private\_chats), 67
- BotCommandScopeChat (class in aiogram.types.bot\_command\_scope\_chat), 67
- BotCommandScopeChatAdministrators (class in aiogram.types.bot\_command\_scope\_chat\_administrators), 67



67  
 BotCommandScopeChatMember (class in [aiogram.types.bot\\_command\\_scope\\_chat\\_member](#)), 236  
 68  
 BotCommandScopeDefault (class in [aiogram.types.bot\\_command\\_scope\\_default](#)), 69  
 69  
 BotCommandScopeType (class in [aiogram.enums.bot\\_command\\_scope\\_type](#)), 376  
 BotDescription (class in [aiogram.types.bot\\_description](#)), 69  
 BotName (class in [aiogram.types.bot\\_name](#)), 69  
 BotShortDescription (class in [aiogram.types.bot\\_short\\_description](#)), 69  
 BOWLING ([aiogram.enums.dice\\_emoji.DiceEmoji](#) attribute), 383  
 BOWLING ([aiogram.types.dice.DiceEmoji](#) attribute), 144  
 BRL ([aiogram.enums.currency.Currency](#) attribute), 380  
 BufferedInputFile (class in [aiogram.types.input\\_file](#)), 149, 391  
 build() ([aiogram.fsm.storage.redis.DefaultKeyBuilder](#) method), 432  
 build() ([aiogram.fsm.storage.redis.KeyBuilder](#) method), 432  
 button([aiogram.methods.answer\\_inline\\_query.AnswerInlineQuery](#) attribute), 373  
 button\_text([aiogram.types.web\\_app\\_data.WebAppData](#) attribute), 215  
 buttons ([aiogram.utils.keyboard.InlineKeyboardBuilder](#) property), 446  
 buttons ([aiogram.utils.keyboard.ReplyKeyboardBuilder](#) property), 447  
 BYN ([aiogram.enums.currency.Currency](#) attribute), 380  
**C**  
 cache\_time([aiogram.methods.answer\\_callback\\_query.AnswerCallbackQuery](#) attribute), 237  
 cache\_time([aiogram.methods.answer\\_inline\\_query.AnswerInlineQuery](#) attribute), 373  
 cache\_time([aiogram.utils.callback\\_answer.CallbackAnswer](#) property), 460  
 CAD ([aiogram.enums.currency.Currency](#) attribute), 380  
 callback\_data([aiogram.handlers.callback\\_query.CallbackQueryHandler](#) property), 440  
 callback\_data([aiogram.types.inline\\_keyboard\\_button.InlineKeyboardButton](#) attribute), 148  
 callback\_game([aiogram.types.inline\\_keyboard\\_button.InlineKeyboardButton](#) attribute), 148  
 CALLBACK\_QUERY ([aiogram.enums.update\\_type.UpdateType](#) attribute), 388  
 callback\_query ([aiogram.types.update.Update](#) attribute), 226  
 callback\_query\_id([aiogram.methods.answer\\_callback\\_query.AnswerCallbackQuery](#) attribute), 236  
 CallbackAnswer (class in [aiogram.utils.callback\\_answer](#)), 460  
 CallbackAnswerException, 437  
 CallbackAnswerMiddleware (class in [aiogram.utils.callback\\_answer](#)), 459  
 CallbackData (class in [aiogram.filters.callback\\_data](#)), 409  
 CallbackGame (class in [aiogram.types.callback\\_game](#)), 235  
 CallbackQuery (class in [aiogram.types.callback\\_query](#)), 70  
 CallbackQueryHandler (class in [aiogram.handlers.callback\\_query](#)), 440  
 can\_add\_web\_page\_previews ([aiogram.types.chat\\_member\\_restricted.ChatMemberRestricted](#) attribute), 124  
 can\_add\_web\_page\_previews ([aiogram.types.chat\\_permissions.ChatPermissions](#) attribute), 141  
 can\_be\_edited([aiogram.types.chat\\_member\\_administrator.ChatMemberAdministrator](#) attribute), 120  
 can\_change\_info([aiogram.methods.promote\\_chat\\_member.PromoteChatMember](#) attribute), 275  
 can\_change\_info([aiogram.types.chat\\_administrator\\_rights.ChatAdministratorRights](#) attribute), 85  
 can\_change\_info([aiogram.types.chat\\_member\\_administrator.ChatMemberAdministrator](#) attribute), 121  
 can\_change\_info([aiogram.types.chat\\_member\\_restricted.ChatMemberRestricted](#) attribute), 124  
 can\_change\_info([aiogram.types.chat\\_permissions.ChatPermissions](#) attribute), 142  
 can\_delete\_messages ([aiogram.methods.promote\\_chat\\_member.PromoteChatMember](#) attribute), 275  
 can\_delete\_messages ([aiogram.types.chat\\_administrator\\_rights.ChatAdministratorRights](#) attribute), 84  
 can\_delete\_messages ([aiogram.types.chat\\_member\\_administrator.ChatMemberAdministrator](#) attribute), 120  
 can\_edit\_messages([aiogram.methods.promote\\_chat\\_member.PromoteChatMember](#) attribute), 274  
 can\_edit\_messages([aiogram.types.chat\\_administrator\\_rights.ChatAdministratorRights](#) attribute), 85  
 can\_edit\_messages([aiogram.types.chat\\_member\\_administrator.ChatMemberAdministrator](#) attribute), 121  
 can\_invite\_users([aiogram.methods.promote\\_chat\\_member.PromoteChatMember](#) attribute), 275  
 can\_invite\_users([aiogram.types.chat\\_administrator\\_rights.ChatAdministratorRights](#) attribute), 85  
 can\_invite\_users([aiogram.types.chat\\_member\\_administrator.ChatMemberAdministrator](#) attribute), 121

[can\\_invite\\_users\(aiogram.types.chat\\_member\\_restricted.ChatMemberRestricted, attribute\), 124](#)  
[can\\_invite\\_users\(aiogram.types.chat\\_permissions.ChatPermissions, attribute\), 142](#)  
[can\\_join\\_groups\(aiogram.types.user.User attribute\), 210](#)  
[can\\_manage\\_chat\(aiogram.methods.promote\\_chat\\_member.PromoteChatMember, attribute\), 274](#)  
[can\\_manage\\_chat\(aiogram.types.chat\\_administrator\\_rights.ChatAdministratorRights, attribute\), 84](#)  
[can\\_manage\\_chat\(aiogram.types.chat\\_member\\_administrator.ChatMemberAdministrator, attribute\), 120](#)  
[can\\_manage\\_topics\(aiogram.methods.promote\\_chat\\_member.PromoteChatMember, attribute\), 275](#)  
[can\\_manage\\_topics\(aiogram.types.chat\\_administrator\\_rights.ChatAdministratorRights, attribute\), 85](#)  
[can\\_manage\\_topics\(aiogram.types.chat\\_member\\_administrator.ChatMemberAdministrator, attribute\), 121](#)  
[can\\_manage\\_topics\(aiogram.types.chat\\_member\\_restricted.ChatMemberRestricted, attribute\), 124](#)  
[can\\_manage\\_topics\(aiogram.types.chat\\_permissions.ChatPermissions, attribute\), 141](#)  
[can\\_manage\\_topics\(aiogram.types.chat\\_permissions.ChatPermissions, attribute\), 142](#)  
[can\\_manage\\_video\\_chats\(aiogram.methods.promote\\_chat\\_member.PromoteChatMember, attribute\), 275](#)  
[can\\_manage\\_video\\_chats\(aiogram.types.chat\\_administrator\\_rights.ChatAdministratorRights, attribute\), 85](#)  
[can\\_manage\\_video\\_chats\(aiogram.types.chat\\_member\\_administrator.ChatMemberAdministrator, attribute\), 121](#)  
[can\\_pin\\_messages\(aiogram.methods.promote\\_chat\\_member.PromoteChatMember, attribute\), 275](#)  
[can\\_pin\\_messages\(aiogram.types.chat\\_administrator\\_rights.ChatAdministratorRights, attribute\), 85](#)  
[can\\_pin\\_messages\(aiogram.types.chat\\_member\\_administrator.ChatMemberAdministrator, attribute\), 121](#)  
[can\\_pin\\_messages\(aiogram.types.chat\\_member\\_restricted.ChatMemberRestricted, attribute\), 124](#)  
[can\\_pin\\_messages\(aiogram.types.chat\\_permissions.ChatPermissions, attribute\), 141](#)  
[can\\_pin\\_messages\(aiogram.types.chat\\_permissions.ChatPermissions, attribute\), 142](#)  
[can\\_post\\_messages\(aiogram.methods.promote\\_chat\\_member.PromoteChatMember, attribute\), 274](#)  
[can\\_post\\_messages\(aiogram.types.chat\\_administrator\\_rights.ChatAdministratorRights, attribute\), 85](#)  
[can\\_post\\_messages\(aiogram.types.chat\\_member\\_administrator.ChatMemberAdministrator, attribute\), 121](#)  
[can\\_promote\\_members\(aiogram.methods.promote\\_chat\\_member.PromoteChatMember, attribute\), 275](#)  
[can\\_promote\\_members\(aiogram.types.chat\\_administrator\\_rights.ChatAdministratorRights, attribute\), 85](#)  
[can\\_promote\\_members\(aiogram.types.chat\\_member\\_restricted.ChatMemberRestricted, attribute\), 124](#)  
[can\\_promote\\_members\(aiogram.types.chat\\_permissions.ChatPermissions, attribute\), 141](#)  
[can\\_restrict\\_members\(aiogram.methods.promote\\_chat\\_member.PromoteChatMember, attribute\), 275](#)  
[can\\_restrict\\_members\(aiogram.types.chat\\_administrator\\_rights.ChatAdministratorRights, attribute\), 85](#)  
[can\\_restrict\\_members\(aiogram.types.chat\\_member\\_administrator.ChatMemberAdministrator, attribute\), 121](#)  
[can\\_restrict\\_members\(aiogram.types.chat\\_member\\_restricted.ChatMemberRestricted, attribute\), 124](#)  
[can\\_restrict\\_members\(aiogram.types.chat\\_permissions.ChatPermissions, attribute\), 141](#)  
[can\\_restrict\\_members\(aiogram.types.chat\\_permissions.ChatPermissions, attribute\), 142](#)  
[can\\_send\\_audios\(aiogram.types.chat\\_member\\_restricted.ChatMemberRestricted, attribute\), 123](#)  
[can\\_send\\_audios\(aiogram.types.chat\\_permissions.ChatPermissions, attribute\), 141](#)  
[can\\_send\\_documents\(aiogram.types.chat\\_member\\_restricted.ChatMemberRestricted, attribute\), 123](#)  
[can\\_send\\_documents\(aiogram.types.chat\\_permissions.ChatPermissions, attribute\), 141](#)  
[can\\_send\\_messages\(aiogram.types.chat\\_member\\_restricted.ChatMemberRestricted, attribute\), 123](#)  
[can\\_send\\_messages\(aiogram.types.chat\\_permissions.ChatPermissions, attribute\), 141](#)  
[can\\_send\\_messages\(aiogram.types.chat\\_permissions.ChatPermissions, attribute\), 142](#)  
[can\\_send\\_other\\_messages\(aiogram.types.chat\\_member\\_restricted.ChatMemberRestricted, attribute\), 124](#)  
[can\\_send\\_other\\_messages\(aiogram.types.chat\\_permissions.ChatPermissions, attribute\), 141](#)  
[can\\_send\\_photos\(aiogram.types.chat\\_member\\_restricted.ChatMemberRestricted, attribute\), 123](#)  
[can\\_send\\_photos\(aiogram.types.chat\\_permissions.ChatPermissions, attribute\), 141](#)  
[can\\_send\\_polls\(aiogram.types.chat\\_member\\_restricted.ChatMemberRestricted, attribute\), 124](#)  
[can\\_send\\_polls\(aiogram.types.chat\\_permissions.ChatPermissions, attribute\), 141](#)  
[can\\_send\\_video\\_notes\(aiogram.types.chat\\_member\\_restricted.ChatMemberRestricted, attribute\), 124](#)  
[can\\_send\\_video\\_notes\(aiogram.types.chat\\_permissions.ChatPermissions, attribute\), 141](#)  
[can\\_send\\_voice\\_notes\(aiogram.types.chat\\_member\\_restricted.ChatMemberRestricted, attribute\), 124](#)  
[can\\_send\\_voice\\_notes\(aiogram.types.chat\\_permissions.ChatPermissions, attribute\), 141](#)

attribute), 141  
 can\_set\_sticker\_set (aiogram.types.chat.Chat attribute), 73  
 caption (aiogram.methods.copy\_message.CopyMessage attribute), 244  
 caption (aiogram.methods.edit\_message\_caption.EditMessageCaption attribute), 363  
 caption (aiogram.methods.send\_animation.SendAnimation attribute), 281  
 caption (aiogram.methods.send\_audio.SendAudio attribute), 283  
 caption (aiogram.methods.send\_document.SendDocument attribute), 290  
 caption (aiogram.methods.send\_photo.SendPhoto attribute), 297  
 caption (aiogram.methods.send\_video.SendVideo attribute), 304  
 caption (aiogram.methods.send\_voice.SendVoice attribute), 308  
 caption (aiogram.types.inline\_query\_result\_audio.InlineQueryResultAudio attribute), 22  
 caption (aiogram.types.inline\_query\_result\_cached\_audio.InlineQueryResultCachedAudio attribute), 25  
 caption (aiogram.types.inline\_query\_result\_cached\_document.InlineQueryResultCachedDocument attribute), 27  
 caption (aiogram.types.inline\_query\_result\_cached\_gif.InlineQueryResultCachedGif attribute), 29  
 caption (aiogram.types.inline\_query\_result\_cached\_mpeg4\_gif.InlineQueryResultCachedMpeg4Gif attribute), 31  
 caption (aiogram.types.inline\_query\_result\_cached\_photo.InlineQueryResultCachedPhoto attribute), 33  
 caption (aiogram.types.inline\_query\_result\_cached\_video.InlineQueryResultCachedVideo attribute), 37  
 caption (aiogram.types.inline\_query\_result\_cached\_voice.InlineQueryResultCachedVoice attribute), 39  
 caption (aiogram.types.inline\_query\_result\_document.InlineQueryResultDocument attribute), 43  
 caption (aiogram.types.inline\_query\_result\_gif.InlineQueryResultGif attribute), 45  
 caption (aiogram.types.inline\_query\_result\_mpeg4\_gif.InlineQueryResultMpeg4Gif attribute), 49  
 caption (aiogram.types.inline\_query\_result\_photo.InlineQueryResultPhoto attribute), 51  
 caption (aiogram.types.inline\_query\_result\_video.InlineQueryResultVideo attribute), 55  
 caption (aiogram.types.inline\_query\_result\_voice.InlineQueryResultVoice attribute), 56  
 caption (aiogram.types.input\_media\_animation.InputMediaAnimation attribute), 150  
 caption (aiogram.types.input\_media\_audio.InputMediaAudio attribute), 151  
 caption (aiogram.types.input\_media\_document.InputMediaDocument attribute), 152  
 caption (aiogram.types.input\_media\_photo.InputMediaPhoto attribute), 152  
 caption (aiogram.types.input\_media\_video.InputMediaVideo attribute), 154  
 caption (aiogram.types.message.Message attribute), 163  
 caption\_entities (aiogram.methods.copy\_message.CopyMessage attribute), 244  
 caption\_entities (aiogram.methods.edit\_message\_caption.EditMessageCaption attribute), 363  
 caption\_entities (aiogram.methods.send\_animation.SendAnimation attribute), 281  
 caption\_entities (aiogram.methods.send\_audio.SendAudio attribute), 283  
 caption\_entities (aiogram.methods.send\_document.SendDocument attribute), 290  
 caption\_entities (aiogram.methods.send\_photo.SendPhoto attribute), 297  
 caption\_entities (aiogram.methods.send\_video.SendVideo attribute), 304  
 caption\_entities (aiogram.methods.send\_voice.SendVoice attribute), 308  
 caption\_entities (aiogram.types.inline\_query\_result\_audio.InlineQueryResultAudio attribute), 22  
 caption\_entities (aiogram.types.inline\_query\_result\_cached\_audio.InlineQueryResultCachedAudio attribute), 25  
 caption\_entities (aiogram.types.inline\_query\_result\_cached\_document.InlineQueryResultCachedDocument attribute), 27  
 caption\_entities (aiogram.types.inline\_query\_result\_cached\_gif.InlineQueryResultCachedGif attribute), 29  
 caption\_entities (aiogram.types.inline\_query\_result\_cached\_mpeg4\_gif.InlineQueryResultCachedMpeg4Gif attribute), 31  
 caption\_entities (aiogram.types.inline\_query\_result\_cached\_photo.InlineQueryResultCachedPhoto attribute), 33  
 caption\_entities (aiogram.types.inline\_query\_result\_cached\_video.InlineQueryResultCachedVideo attribute), 37  
 caption\_entities (aiogram.types.inline\_query\_result\_cached\_voice.InlineQueryResultCachedVoice attribute), 39  
 caption\_entities (aiogram.types.inline\_query\_result\_document.InlineQueryResultDocument attribute), 43  
 caption\_entities (aiogram.types.inline\_query\_result\_gif.InlineQueryResultGif attribute), 45  
 caption\_entities (aiogram.types.inline\_query\_result\_mpeg4\_gif.InlineQueryResultMpeg4Gif attribute), 49  
 caption\_entities (aiogram.types.inline\_query\_result\_photo.InlineQueryResultPhoto attribute), 51  
 caption\_entities (aiogram.types.inline\_query\_result\_video.InlineQueryResultVideo attribute), 55  
 caption\_entities (aiogram.types.inline\_query\_result\_voice.InlineQueryResultVoice attribute), 56  
 caption\_entities (aiogram.types.input\_media\_animation.InputMediaAnimation attribute), 150  
 caption\_entities (aiogram.types.input\_media\_audio.InputMediaAudio attribute), 151  
 caption\_entities (aiogram.types.input\_media\_document.InputMediaDocument attribute), 152  
 caption\_entities (aiogram.types.input\_media\_photo.InputMediaPhoto attribute), 152

caption\_entities(aiogram.types.input\_media\_photo.InputMediaPhoto attribute), 153

caption\_entities(aiogram.types.input\_media\_video.InputMediaVideo attribute), 154

caption\_entities(aiogram.types.message.Message attribute), 163

CASHTAG(aiogram.enums.message\_entity\_type.MessageEntityType attribute), 385

CashTag(class in aiogram.utils.formatting), 465

certificate(aiogram.methods.set\_webhook.SetWebhook attribute), 359

CHANNEL(aiogram.enums.chat\_type.ChatType attribute), 378

CHANNEL\_CHAT\_CREATED(aiogram.enums.content\_type.ContentType attribute), 379

channel\_chat\_created(aiogram.types.message.Message attribute), 164

CHANNEL\_POST(aiogram.enums.update\_type.UpdateType attribute), 388

channel\_post(aiogram.types.update.Update attribute), 225

CHAT(aiogram.enums.bot\_command\_scope\_type.BotCommandScopeType attribute), 376

chat(aiogram.types.chat\_join\_request.ChatJoinRequest attribute), 86

chat(aiogram.types.chat\_member\_updated.ChatMemberUpdated attribute), 124

chat(aiogram.types.message.Message attribute), 162

Chat(class in aiogram.types.chat), 71

CHAT\_ADMINISTRATORS(aiogram.enums.bot\_command\_scope\_type.BotCommandScopeType attribute), 376

chat\_has\_username(aiogram.types.keyboard\_button\_request.KeyboardButtonRequest attribute), 157

chat\_id(aiogram.methods.approve\_chat\_join\_request.ApproveChatJoinRequest attribute), 237

chat\_id(aiogram.methods.ban\_chat\_member.BanChatMember attribute), 238

chat\_id(aiogram.methods.ban\_chat\_sender\_chat.BanChatSenderChat attribute), 240

chat\_id(aiogram.methods.close\_forum\_topic.CloseForumTopic attribute), 242

chat\_id(aiogram.methods.close\_general\_forum\_topic.CloseGeneralForumTopic attribute), 243

chat\_id(aiogram.methods.copy\_message.CopyMessage attribute), 244

chat\_id(aiogram.methods.create\_chat\_invite\_link.CreateChatInviteLink attribute), 246

chat\_id(aiogram.methods.create\_forum\_topic.CreateForumTopic attribute), 247

chat\_id(aiogram.methods.decline\_chat\_join\_request.DeclineChatJoinRequest attribute), 248

chat\_id(aiogram.methods.delete\_chat\_photo.DeleteChatPhoto attribute), 249

chat\_id(aiogram.methods.delete\_chat\_sticker\_set.DeleteChatStickerSet attribute), 250

chat\_id(aiogram.methods.delete\_forum\_topic.DeleteForumTopic attribute), 251

chat\_id(aiogram.methods.delete\_message.DeleteMessage attribute), 361

chat\_id(aiogram.methods.edit\_chat\_invite\_link.EditChatInviteLink attribute), 253

chat\_id(aiogram.methods.edit\_forum\_topic.EditForumTopic attribute), 254

chat\_id(aiogram.methods.edit\_general\_forum\_topic.EditGeneralForumTopic attribute), 255

chat\_id(aiogram.methods.edit\_message\_caption.EditMessageCaption attribute), 363

chat\_id(aiogram.methods.edit\_message\_live\_location.EditMessageLiveLocation attribute), 364

chat\_id(aiogram.methods.edit\_message\_media.EditMessageMedia attribute), 366

chat\_id(aiogram.methods.edit\_message\_reply\_markup.EditMessageReplyMarkup attribute), 367

chat\_id(aiogram.methods.edit\_message\_text.EditMessageText attribute), 369

chat\_id(aiogram.methods.export\_chat\_invite\_link.ExportChatInviteLink attribute), 256

chat\_id(aiogram.methods.forward\_message.ForwardMessage attribute), 257

chat\_id(aiogram.methods.get\_chat.GetChat attribute), 259

chat\_id(aiogram.methods.get\_chat\_administrators.GetChatAdministrators attribute), 259

chat\_id(aiogram.methods.get\_chat\_member.GetChatMember attribute), 260

chat\_id(aiogram.methods.get\_chat\_member\_count.GetChatMemberCount attribute), 261

chat\_id(aiogram.methods.get\_chat\_menu\_button.GetChatMenuButton attribute), 262

chat\_id(aiogram.methods.get\_game\_high\_scores.GetGameHighScores attribute), 352

chat\_id(aiogram.methods.hide\_general\_forum\_topic.HideGeneralForumTopic attribute), 270

chat\_id(aiogram.methods.leave\_chat.LeaveChat attribute), 271

chat\_id(aiogram.methods.pin\_chat\_message.PinChatMessage attribute), 273

chat\_id(aiogram.methods.promote\_chat\_member.PromoteChatMember attribute), 274

chat\_id(aiogram.methods.reopen\_forum\_topic.ReopenForumTopic attribute), 276

chat\_id(aiogram.methods.reopen\_general\_forum\_topic.ReopenGeneralForumTopic attribute), 277

chat\_id(aiogram.methods.restrict\_chat\_member.RestrictChatMember attribute), 278



[chat\\_id \(aiogram.methods.revoke\\_chat\\_invite\\_link.RevokeChatInviteLink attribute\), 279](#)  
[chat\\_id \(aiogram.methods.send\\_animation.SendAnimation chat\\_id \(aiogram.methods.stop\\_poll.StopPoll attribute\), 371](#)  
[chat\\_id \(aiogram.methods.send\\_audio.SendAudio attribute\), 283](#)  
[chat\\_id \(aiogram.methods.send\\_chat\\_action.SendChatAction chat\\_id \(aiogram.methods.unban\\_chat\\_member.UnbanChatMember attribute\), 322](#)  
[chat\\_id \(aiogram.methods.send\\_contact.SendContact chat\\_id \(aiogram.methods.unban\\_chat\\_sender\\_chat.UnbanChatSenderChat attribute\), 323](#)  
[chat\\_id \(aiogram.methods.send\\_dice.SendDice attribute\), 288](#)  
[chat\\_id \(aiogram.methods.send\\_document.SendDocument chat\\_id \(aiogram.methods.unhide\\_general\\_forum\\_topic.UnhideGeneralForumTopic attribute\), 324](#)  
[chat\\_id \(aiogram.methods.send\\_game.SendGame chat\\_id \(aiogram.methods.unpin\\_all\\_chat\\_messages.UnpinAllChatMessages attribute\), 325](#)  
[chat\\_id \(aiogram.methods.send\\_invoice.SendInvoice attribute\), 334](#)  
[chat\\_id \(aiogram.methods.send\\_location.SendLocation chat\\_id \(aiogram.methods.unpin\\_all\\_forum\\_topic\\_messages.UnpinAllForumTopicMessages attribute\), 326](#)  
[chat\\_id \(aiogram.methods.send\\_media\\_group.SendMediaGroup chat\\_id \(aiogram.methods.unpin\\_all\\_general\\_forum\\_topic\\_messages.UnpinAllGeneralForumTopicMessages attribute\), 327](#)  
[chat\\_id \(aiogram.methods.send\\_message.SendMessage chat\\_id \(aiogram.methods.unpin\\_chat\\_message.UnpinChatMessage attribute\), 328](#)  
[chat\\_id \(aiogram.methods.send\\_photo.SendPhoto attribute\), 297](#)  
[chat\\_id \(aiogram.methods.send\\_poll.SendPoll attribute\), 299](#)  
[chat\\_id \(aiogram.methods.send\\_sticker.SendSticker attribute\), 343](#)  
[chat\\_id \(aiogram.methods.send\\_venue.SendVenue attribute\), 301](#)  
[chat\\_id \(aiogram.methods.send\\_video.SendVideo chat\\_id \(aiogram.types.bot\\_command\\_scope\\_chat.BotCommandScopeChat attribute\), 67](#)  
[chat\\_id \(aiogram.methods.send\\_video\\_note.SendVideoNote chat\\_id \(aiogram.types.bot\\_command\\_scope\\_chat\\_administrators.BotCommandScopeChatAdministrators attribute\), 68](#)  
[chat\\_id \(aiogram.methods.send\\_voice.SendVoice chat\\_id \(aiogram.types.bot\\_command\\_scope\\_chat\\_member.BotCommandScopeChatMember attribute\), 68](#)  
[chat\\_id \(aiogram.methods.set\\_chat\\_administrator\\_custom\\_title.SetChatAdministratorCustomTitle chat\\_id \(aiogram.types.chat\\_shared.ChatShared attribute\), 142](#)  
[chat\\_id \(aiogram.methods.set\\_chat\\_description.SetChatDescription chat\\_instance \(aiogram.types.callback\\_query.CallbackQuery attribute\), 70](#)  
[chat\\_id \(aiogram.methods.set\\_chat\\_menu\\_button.SetChatMenuButton chat\\_is\\_channel \(aiogram.types.keyboard\\_button\\_request\\_chat.KeyboardButtonRequestChat attribute\), 156](#)  
[chat\\_id \(aiogram.methods.set\\_chat\\_permissions.SetChatPermissions chat\\_is\\_created \(aiogram.types.keyboard\\_button\\_request\\_chat.KeyboardButtonRequestChat attribute\), 157](#)  
[chat\\_id \(aiogram.methods.set\\_chat\\_photo.SetChatPhoto chat\\_is\\_forum \(aiogram.types.keyboard\\_button\\_request\\_chat.KeyboardButtonRequestChat attribute\), 156](#)  
[chat\\_id \(aiogram.methods.set\\_chat\\_sticker\\_set.SetChatStickerSet CHAT\\_JOIN\\_REQUEST \(aiogram.enums.update\\_type.UpdateType attribute\), 388](#)  
[chat\\_id \(aiogram.methods.set\\_chat\\_title.SetChatTitle chat\\_join\\_request \(aiogram.types.update.Update attribute\), 226](#)  
[chat\\_id \(aiogram.methods.set\\_game\\_score.SetGameScore CHAT\\_MEMBER \(aiogram.enums.update\\_type.UpdateType attribute\), 376](#)  
[chat\\_id \(aiogram.methods.set\\_game\\_score.SetGameScore CHAT\\_MEMBER \(aiogram.enums.update\\_type.UpdateType attribute\), 388](#)  
[chat\\_id \(aiogram.methods.set\\_game\\_score.SetGameScore ChatMember \(aiogram.types.update.Update attribute\), 226](#)  
[chat\\_id \(aiogram.methods.set\\_game\\_score.SetGameScore CHAT\\_SHARED \(aiogram.enums.content\\_type.ContentType attribute\), 379](#)  
[chat\\_id \(aiogram.methods.set\\_game\\_score.SetGameScore chat\\_shared \(aiogram.types.message.Message attribute\), 165](#)  
[chat\\_id \(aiogram.methods.set\\_game\\_score.SetGameScore chat\\_type \(aiogram.types.inline\\_query.InlineQuery attribute\), 18](#)  
[chat\\_id \(aiogram.methods.set\\_game\\_score.SetGameScore ChatAction \(class in aiogram.enums.chat\\_action\), 377](#)  
[chat\\_id \(aiogram.methods.set\\_game\\_score.SetGameScore ChatActionMiddleware \(class in aiogram.utils.chat\\_action\), 454](#)  
[chat\\_id \(aiogram.methods.set\\_game\\_score.SetGameScore ChatActionSender \(class in aiogram.utils.chat\\_action\),](#)

- 453
- `ChatAdministratorRights` (class in `aiogram.types.chat_administrator_rights`), 84
- `ChatInviteLink` (class in `aiogram.types.chat_invite_link`), 85
- `ChatJoinRequest` (class in `aiogram.types.chat_join_request`), 86
- `ChatLocation` (class in `aiogram.types.chat_location`), 119
- `ChatMember` (class in `aiogram.types.chat_member`), 119
- `ChatMemberAdministrator` (class in `aiogram.types.chat_member_administrator`), 120
- `ChatMemberBanned` (class in `aiogram.types.chat_member_banned`), 121
- `ChatMemberLeft` (class in `aiogram.types.chat_member_left`), 122
- `ChatMemberMember` (class in `aiogram.types.chat_member_member`), 122
- `ChatMemberOwner` (class in `aiogram.types.chat_member_owner`), 122
- `ChatMemberRestricted` (class in `aiogram.types.chat_member_restricted`), 123
- `ChatMemberStatus` (class in `aiogram.enums.chat_member_status`), 377
- `ChatMemberUpdated` (class in `aiogram.types.chat_member_updated`), 124
- `ChatMemberUpdatedFilter` (class in `aiogram.filters.chat_member_updated`), 404
- `ChatPermissions` (class in `aiogram.types.chat_permissions`), 141
- `ChatPhoto` (class in `aiogram.types.chat_photo`), 142
- `ChatShared` (class in `aiogram.types.chat_shared`), 142
- `ChatType` (class in `aiogram.enums.chat_type`), 378
- `check_flags()` (in module `aiogram.dispatcher.flags`), 438
- `check_response()` (`aiogram.client.session.base.BaseSession` method), 13
- `check_webapp_signature()` (in module `aiogram.utils.web_app`), 455
- CHF (`aiogram.enums.currency.Currency` attribute), 380
- CHIN (`aiogram.enums.mask_position_point.MaskPositionPoint` attribute), 385
- CHOOSE\_STICKER (`aiogram.enums.chat_action.ChatAction` attribute), 377
- `choose_sticker()` (`aiogram.utils.chat_action.ChatAction` class method), 453
- CHOSEN\_INLINE\_RESULT (`aiogram.enums.update_type.UpdateType` attribute), 388
- `chosen_inline_result` (`aiogram.types.update.Update` attribute), 225
- `ChosenInlineResult` (class in `aiogram.types.chosen_inline_result`), 17
- city (`aiogram.types.shipping_address.ShippingAddress` attribute), 233
- `ClientDecodeError`, 437
- `Close` (class in `aiogram.methods.close`), 241
- `close()` (`aiogram.client.session.base.BaseSession` method), 13
- `close()` (`aiogram.fsm.storage.base.BaseStorage` method), 433
- `close()` (`aiogram.webhook.aihttp_server.SimpleRequestHandler` method), 416
- `close_date` (`aiogram.methods.send_poll.SendPoll` attribute), 300
- `close_date` (`aiogram.types.poll.Poll` attribute), 206
- `CloseForumTopic` (class in `aiogram.methods.close_forum_topic`), 242
- `CloseGeneralForumTopic` (class in `aiogram.methods.close_general_forum_topic`), 243
- CLP (`aiogram.enums.currency.Currency` attribute), 380
- CNY (`aiogram.enums.currency.Currency` attribute), 380
- CODE (`aiogram.enums.message_entity_type.MessageEntityType` attribute), 385
- `Code` (class in `aiogram.utils.formatting`), 466
- `command` (`aiogram.filters.command.CommandObject` attribute), 403
- `command` (`aiogram.types.bot_command.BotCommand` attribute), 65
- `Command` (class in `aiogram.filters.command`), 402
- `CommandObject` (class in `aiogram.filters.command`), 402
- COMMANDS (`aiogram.enums.menu_button_type.MenuButtonType` attribute), 385
- `commands` (`aiogram.methods.set_my_commands.SetMyCommands` attribute), 316
- CONNECTED\_WEBSITE (`aiogram.enums.content_type.ContentType` attribute), 379
- `connected_website` (`aiogram.types.message.Message` attribute), 165
- `ConstI18nMiddleware` (class in `aiogram.utils.i18n.middleware`), 450
- CONTACT (`aiogram.enums.content_type.ContentType` attribute), 378
- CONTACT (`aiogram.enums.inline_query_result_type.InlineQueryResultType` attribute), 384
- `contact` (`aiogram.types.message.Message` attribute), 163
- `Contact` (class in `aiogram.types.contact`), 143
- `Content` (class in `aiogram.types.message.Message` property), 166
- `ContentType` (class in `aiogram.enums.content_type`), 378
- COP (`aiogram.enums.currency.Currency` attribute), 380
- `copy()` (`aiogram.utils.keyboard.InlineKeyboardBuilder` method), 446

`copy()` (*aiogram.utils.keyboard.ReplyKeyboardBuilder* method), 448  
`copy_to()` (*aiogram.types.message.Message* method), 197  
`CopyMessage` (class in *aiogram.methods.copy\_message*), 243  
`correct_option_id` (*aiogram.methods.send\_poll.SendPoll* attribute), 299  
`correct_option_id` (*aiogram.types.poll.Poll* attribute), 206  
`country_code` (*aiogram.types.shipping\_address.ShippingAddress* attribute), 232  
`CRC` (*aiogram.enums.currency.Currency* attribute), 380  
`create_invite_link()` (*aiogram.types.chat.Chat* method), 76  
`CreateChatInviteLink` (class in *aiogram.methods.create\_chat\_invite\_link*), 245  
`CreateForumTopic` (class in *aiogram.methods.create\_forum\_topic*), 247  
`CreateInvoiceLink` (class in *aiogram.methods.create\_invoice\_link*), 331  
`CreateNewStickerSet` (class in *aiogram.methods.create\_new\_sticker\_set*), 338  
`creates_join_request` (*aiogram.methods.create\_chat\_invite\_link.CreateChatInviteLink* attribute), 246  
`creates_join_request` (*aiogram.methods.edit\_chat\_invite\_link.EditChatInviteLink* attribute), 253  
`creates_join_request` (*aiogram.types.chat\_invite\_link.ChatInviteLink* attribute), 85  
`CREATOR` (*aiogram.enums.chat\_member\_status.ChatMemberStatus* attribute), 377  
`creator` (*aiogram.types.chat\_invite\_link.ChatInviteLink* attribute), 85  
`credentials` (*aiogram.types.passport\_data.PassportData* attribute), 217  
`currency` (*aiogram.methods.create\_invoice\_link.CreateInvoiceLink* attribute), 332  
`currency` (*aiogram.methods.send\_invoice.SendInvoice* attribute), 334  
`currency` (*aiogram.types.input\_invoice\_message\_content.InputInvoiceMessageContent* attribute), 60  
`currency` (*aiogram.types.invoice.Invoice* attribute), 230  
`currency` (*aiogram.types.pre\_checkout\_query.PreCheckoutQuery* attribute), 231  
`currency` (*aiogram.types.successful\_payment.SuccessfulPayment* attribute), 234  
`Currency` (class in *aiogram.enums.currency*), 380  
`CUSTOM_EMOJI` (*aiogram.enums.message\_entity\_type.MessageEntityType* attribute), 386  
`CUSTOM_EMOJI` (*aiogram.enums.sticker\_type.StickerType* attribute), 387  
`custom_emoji_id` (*aiogram.methods.set\_custom\_emoji\_sticker\_set\_thumbnail.SetCustomEmojiStickerSetThumbnail* attribute), 344  
`custom_emoji_id` (*aiogram.types.message\_entity.MessageEntity* attribute), 204  
`custom_emoji_id` (*aiogram.types.sticker.Sticker* attribute), 229  
`custom_emoji_ids` (*aiogram.methods.get\_custom\_emoji\_stickers.GetCustomEmojiStickers* attribute), 341  
`custom_title` (*aiogram.methods.set\_chat\_administrator\_custom\_title.SetChatAdministratorCustomTitle* attribute), 310  
`custom_title` (*aiogram.types.chat\_member\_administrator.ChatMemberAdministrator* attribute), 121  
`custom_title` (*aiogram.types.chat\_member\_owner.ChatMemberOwner* attribute), 123  
`CustomEmoji` (class in *aiogram.utils.formatting*), 467  
`CZK` (*aiogram.enums.currency.Currency* attribute), 380  

## D

`DART` (*aiogram.enums.dice\_emoji.DiceEmoji* attribute), 383  
`DART` (*aiogram.types.dice.DiceEmoji* attribute), 143  
`DATA` (*aiogram.enums.passport\_element\_error\_type.PassportElementErrorType* attribute), 386  
`data` (*aiogram.types.callback\_query.CallbackQuery* attribute), 70  
`data` (*aiogram.types.encrypted\_credentials.EncryptedCredentials* attribute), 215  
`data` (*aiogram.types.encrypted\_passport\_element.EncryptedPassportElement* attribute), 216  
`data` (*aiogram.types.passport\_data.PassportData* attribute), 217  
`data` (*aiogram.types.web\_app\_data.WebAppData* attribute), 215  
`data_hash` (*aiogram.types.passport\_element\_error\_data\_field.PassportElementErrorDataField* attribute), 218  
`date` (*aiogram.types.chat\_join\_request.ChatJoinRequest* attribute), 86  
`date` (*aiogram.types.chat\_member\_updated.ChatMemberUpdated* attribute), 125  
`date` (*aiogram.types.message.Message* attribute), 162  
`decline()` (*aiogram.types.chat\_join\_request.ChatJoinRequest* method), 87  
`DeclineChatJoinRequest` (class in *aiogram.methods.decline\_chat\_join\_request*), 248  
`DEFAULT` (*aiogram.enums.bot\_command\_scope\_type.BotCommandScopeType* attribute), 376  
`DEFAULT` (*aiogram.enums.menu\_button\_type.MenuButtonType* attribute), 385  
`DefaultKeyBuilder` (class in *aiogram.fsm.storage.redis*), 432

`delete()` (*aiogram.types.message.Message* method), 202  
`DELETE_CHAT_PHOTO` (*aiogram.enums.content\_type.ContentType* attribute), 379  
`delete_chat_photo` (*aiogram.types.message.Message* attribute), 164  
`delete_from_set()` (*aiogram.types.sticker.Sticker* method), 229  
`delete_message()` (*aiogram.types.chat.Chat* method), 74  
`delete_photo()` (*aiogram.types.chat.Chat* method), 83  
`delete_reply_markup()` (*aiogram.types.message.Message* method), 200  
`delete_sticker_set()` (*aiogram.types.chat.Chat* method), 77  
`DeleteChatPhoto` (class in *aiogram.methods.delete\_chat\_photo*), 249  
`DeleteChatStickerSet` (class in *aiogram.methods.delete\_chat\_sticker\_set*), 250  
`DeleteForumTopic` (class in *aiogram.methods.delete\_forum\_topic*), 251  
`DeleteMessage` (class in *aiogram.methods.delete\_message*), 361  
`DeleteMyCommands` (class in *aiogram.methods.delete\_my\_commands*), 252  
`DeleteStickerFromSet` (class in *aiogram.methods.delete\_sticker\_from\_set*), 339  
`DeleteStickerSet` (class in *aiogram.methods.delete\_sticker\_set*), 340  
`DeleteWebhook` (class in *aiogram.methods.delete\_webhook*), 356  
`description` (*aiogram.methods.create\_invoice\_link.CreateInvoiceLink* attribute), 332  
`description` (*aiogram.methods.send\_invoice.SendInvoice* attribute), 334  
`description` (*aiogram.methods.set\_chat\_description.SetChatDescription* attribute), 311  
`description` (*aiogram.methods.set\_my\_description.SetMyDescription* attribute), 319  
`description` (*aiogram.types.bot\_command.BotCommand* attribute), 65  
`description` (*aiogram.types.bot\_description.BotDescription* attribute), 69  
`description` (*aiogram.types.chat.Chat* attribute), 72  
`description` (*aiogram.types.game.Game* attribute), 235  
`description` (*aiogram.types.inline\_query\_result\_article.InlineQueryResultArticle* attribute), 21  
`description` (*aiogram.types.inline\_query\_result\_cached\_document.InlineQueryResultCachedDocument* attribute), 27  
`description` (*aiogram.types.inline\_query\_result\_cached\_photo.InlineQueryResultCachedPhoto* attribute), 33  
`description` (*aiogram.types.inline\_query\_result\_cached\_video.InlineQueryResultCachedVideo* attribute), 37  
`description` (*aiogram.types.inline\_query\_result\_document.InlineQueryResultDocument* attribute), 43  
`description` (*aiogram.types.inline\_query\_result\_photo.InlineQueryResultPhoto* attribute), 51  
`description` (*aiogram.types.inline\_query\_result\_video.InlineQueryResultVideo* attribute), 55  
`description` (*aiogram.types.input\_invoice\_message\_content.InputInvoiceMessageContent* attribute), 59  
`description` (*aiogram.types.invoice.Invoice* attribute), 230  
`DetailedAiogramError`, 437  
`DICE` (*aiogram.enums.content\_type.ContentType* attribute), 378  
`DICE` (*aiogram.enums.dice\_emoji.DiceEmoji* attribute), 383  
`dice` (*aiogram.types.dice.DiceEmoji* attribute), 143  
`Dice` (class in *aiogram.types.dice*), 143  
`DiceEmoji` (class in *aiogram.enums.dice\_emoji*), 383  
`DiceEmoji` (class in *aiogram.types.dice*), 143  
`disable()` (*aiogram.utils.callback\_answer.CallbackAnswer* method), 460  
`disable_content_type_detection` (*aiogram.methods.send\_document.SendDocument* attribute), 290  
`disable_content_type_detection` (*aiogram.types.input\_media\_document.InputMediaDocument* attribute), 152  
`disable_edit_message` (*aiogram.methods.set\_game\_score.SetGameScore* attribute), 355  
`disable_notification` (*aiogram.methods.copy\_message.CopyMessage* attribute), 244  
`disable_notification` (*aiogram.methods.forward\_message.ForwardMessage* attribute), 258  
`disable_notification` (*aiogram.methods.pin\_chat\_message.PinChatMessage* attribute), 273  
`disable_notification` (*aiogram.methods.send\_animation.SendAnimation* attribute), 281  
`disable_notification` (*aiogram.methods.send\_audio.SendAudio* attribute), 283  
`disable_notification` (*aiogram.methods.send\_contact.SendContact* attribute), 284  
`disable_notification` (*aiogram.methods.send\_dice.SendDice* attribute), 285



- tribute), 288
- disable\_notification
  - (aiogram.methods.send\_document.SendDocumentdo() attribute), 290
- disable\_notification
  - (aiogram.methods.send\_game.SendGame attribute), 353
- disable\_notification
  - (aiogram.methods.send\_invoice.SendInvoice attribute), 336
- disable\_notification
  - (aiogram.methods.send\_location.SendLocation attribute), 292
- disable\_notification
  - (aiogram.methods.send\_media\_group.SendMediaGroup attribute), 294
- disable\_notification
  - (aiogram.methods.send\_message.SendMessage attribute), 296
- disable\_notification
  - (aiogram.methods.send\_photo.SendPhoto attribute), 297
- disable\_notification
  - (aiogram.methods.send\_poll.SendPoll attribute), 300
- disable\_notification
  - (aiogram.methods.send\_sticker.SendSticker attribute), 343
- disable\_notification
  - (aiogram.methods.send\_venue.SendVenue attribute), 302
- disable\_notification
  - (aiogram.methods.send\_video.SendVideo attribute), 304
- disable\_notification
  - (aiogram.methods.send\_video\_note.SendVideoNote attribute), 306
- disable\_notification
  - (aiogram.methods.send\_voice.SendVoice attribute), 308
- disable\_web\_page\_preview
  - (aiogram.methods.edit\_message\_text.EditMessageText attribute), 369
- disable\_web\_page\_preview
  - (aiogram.methods.send\_message.SendMessage attribute), 295
- disable\_web\_page\_preview
  - (aiogram.types.input\_text\_message\_content.InputTextMessageContent attribute), 62
- disabled (aiogram.utils.callback\_answer.CallbackAnswer property), 460
- Dispatcher (class in aiogram.dispatcher.dispatcher), 397
- distance (aiogram.types.proximity\_alert\_triggered.ProximityAlertTriggered attribute), 207
- DKK (aiogram.enums.currency.Currency attribute), 380
- do() (aiogram.types.chat.Chat method), 77
- DOCUMENT (aiogram.enums.content\_type.ContentType attribute), 378
- DOCUMENT (aiogram.enums.inline\_query\_result\_type.InlineQueryResultType attribute), 384
- DOCUMENT (aiogram.enums.input\_media\_type.InputMediaType attribute), 384
- document (aiogram.methods.send\_document.SendDocument attribute), 290
- document (aiogram.types.message.Message attribute), 163
- Document (class in aiogram.types.document), 144
- document\_file\_id (aiogram.types.inline\_query\_result\_cached\_document.InlineQueryResultCachedDocument attribute), 27
- document\_url (aiogram.types.inline\_query\_result\_document.InlineQueryResultDocument attribute), 43
- DOP (aiogram.enums.currency.Currency attribute), 380
- download() (aiogram.client.bot.Bot method), 390
- download\_file() (aiogram.client.bot.Bot method), 389
- DRIVER\_LICENSE (aiogram.enums.encrypted\_passport\_element.EncryptedPassportElement attribute), 383
- drop\_pending\_updates
  - (aiogram.methods.delete\_webhook.DeleteWebhook attribute), 356
- drop\_pending\_updates
  - (aiogram.methods.set\_webhook.SetWebhook attribute), 359
- duration (aiogram.methods.send\_animation.SendAnimation attribute), 281
- duration (aiogram.methods.send\_audio.SendAudio attribute), 283
- duration (aiogram.methods.send\_video.SendVideo attribute), 304
- duration (aiogram.methods.send\_video\_note.SendVideoNote attribute), 306
- duration (aiogram.methods.send\_voice.SendVoice attribute), 308
- duration (aiogram.types.animation.Animation attribute), 64
- duration (aiogram.types.audio.Audio attribute), 65
- duration (aiogram.types.input\_media\_animation.InputMediaAnimation attribute), 150
- duration (aiogram.types.input\_media\_audio.InputMediaAudio attribute), 151
- duration (aiogram.types.input\_media\_video.InputMediaVideo attribute), 154
- duration (aiogram.types.video.Video attribute), 212
- duration (aiogram.types.video\_chat\_ended.VideoChatEnded attribute), 213
- duration (aiogram.types.video\_note.VideoNote attribute), 214
- duration (aiogram.types.voice.Voice attribute), 214

DZD (*aiogram.enums.currency.Currency* attribute), 380

## E

`edit_caption()` (*aiogram.types.message.Message* method), 201

`edit_date` (*aiogram.types.message.Message* attribute), 162

`edit_invite_link()` (*aiogram.types.chat.Chat* method), 75

`edit_live_location()` (*aiogram.types.message.Message* method), 200

`edit_media()` (*aiogram.types.message.Message* method), 199

`edit_reply_markup()` (*aiogram.types.message.Message* method), 199

`edit_text()` (*aiogram.types.message.Message* method), 198

`EditChatInviteLink` (class in *aiogram.methods.edit\_chat\_invite\_link*), 253

`EDITED_CHANNEL_POST` (*aiogram.enums.update\_type.UpdateType* attribute), 388

`edited_channel_post` (*aiogram.types.update.Update* attribute), 225

`EDITED_MESSAGE` (*aiogram.enums.update\_type.UpdateType* attribute), 388

`edited_message` (*aiogram.types.update.Update* attribute), 225

`EditForumTopic` (class in *aiogram.methods.edit\_forum\_topic*), 254

`EditGeneralForumTopic` (class in *aiogram.methods.edit\_general\_forum\_topic*), 255

`EditMessageCaption` (class in *aiogram.methods.edit\_message\_caption*), 362

`EditMessageLiveLocation` (class in *aiogram.methods.edit\_message\_live\_location*), 364

`EditMessageMedia` (class in *aiogram.methods.edit\_message\_media*), 366

`EditMessageReplyMarkup` (class in *aiogram.methods.edit\_message\_reply\_markup*), 367

`EditMessageText` (class in *aiogram.methods.edit\_message\_text*), 368

`EGP` (*aiogram.enums.currency.Currency* attribute), 381

`element_hash` (*aiogram.types.passport\_element\_error\_unspecified.PassportElementErrorUnspecified* attribute), 224

`EMAIL` (*aiogram.enums.encrypted\_passport\_element.EncryptedPassportElement* attribute), 383

`EMAIL` (*aiogram.enums.message\_entity\_type.MessageEntityType* attribute), 385

`email` (*aiogram.types.encrypted\_passport\_element.EncryptedPassportElement* attribute), 216

`email` (*aiogram.types.order\_info.OrderInfo* attribute), 231

`Email` (class in *aiogram.utils.formatting*), 465

`emoji` (*aiogram.methods.send\_dice.SendDice* attribute), 288

`emoji` (*aiogram.methods.send\_sticker.SendSticker* attribute), 343

`emoji` (*aiogram.types.dice.Dice* attribute), 143

`emoji` (*aiogram.types.sticker.Sticker* attribute), 229

`emoji_list` (*aiogram.methods.set\_sticker\_emoji\_list.SetStickerEmojiList* attribute), 345

`emoji_list` (*aiogram.types.input\_sticker.InputSticker* attribute), 227

`emoji_status_custom_emoji_id` (*aiogram.types.chat.Chat* attribute), 72

`emoji_status_expiration_date` (*aiogram.types.chat.Chat* attribute), 72

`EncryptedCredentials` (class in *aiogram.types.encrypted\_credentials*), 215

`EncryptedPassportElement` (class in *aiogram.enums.encrypted\_passport\_element*), 383

`EncryptedPassportElement` (class in *aiogram.types.encrypted\_passport\_element*), 216

`entities` (*aiogram.methods.edit\_message\_text.EditMessageText* attribute), 369

`entities` (*aiogram.methods.send\_message.SendMessage* attribute), 295

`entities` (*aiogram.types.input\_text\_message\_content.InputTextMessageContent* attribute), 62

`entities` (*aiogram.types.message.Message* attribute), 163

`error_message` (*aiogram.methods.answer\_pre\_checkout\_query.AnswerPreCheckoutQuery* attribute), 329

`error_message` (*aiogram.methods.answer\_shipping\_query.AnswerShippingQuery* attribute), 331

`ErrorEvent` (class in *aiogram.types.error\_event*), 436

`errors` (*aiogram.methods.set\_passport\_data\_errors.SetPassportDataErrors* attribute), 360

`ETB` (*aiogram.enums.currency.Currency* attribute), 381

`EUR` (*aiogram.enums.currency.Currency* attribute), 381

`event` (*aiogram.types.update.Update* property), 226

`event_type` (*aiogram.types.update.Update* property), 226

`exception` (*aiogram.types.error\_event.ErrorEvent* attribute), 436

`ExceptionMessageFilter` (class in *aiogram.filters.exception*), 411

`exceptions` (*aiogram.filters.exception.ExceptionTypeFilter* attribute), 411

[attribute](#)), 411  
[ExceptionTypeFilter](#) (class in [aiogram.filters.exception](#)), 411  
[expire\\_date](#) ([aiogram.methods.create\\_chat\\_invite\\_link.CreateChatInviteLink](#) attribute), 246  
[expire\\_date](#) ([aiogram.methods.edit\\_chat\\_invite\\_link.EditChatInviteLink](#) attribute), 253  
[expire\\_date](#) ([aiogram.types.chat\\_invite\\_link.ChatInviteLink](#) attribute), 86  
[explanation](#) ([aiogram.methods.send\\_poll.SendPoll](#) attribute), 299  
[explanation](#) ([aiogram.types.poll.Poll](#) attribute), 206  
[explanation\\_entities](#) ([aiogram.methods.send\\_poll.SendPoll](#) attribute), 299  
[explanation\\_entities](#) ([aiogram.types.poll.Poll](#) attribute), 206  
[explanation\\_parse\\_mode](#) ([aiogram.methods.send\\_poll.SendPoll](#) attribute), 299  
[export\(\)](#) ([aiogram.utils.keyboard.InlineKeyboardBuilder](#) method), 446  
[export\(\)](#) ([aiogram.utils.keyboard.ReplyKeyboardBuilder](#) method), 448  
[export\\_invite\\_link\(\)](#) ([aiogram.types.chat.Chat](#) method), 76  
[ExportChatInviteLink](#) (class in [aiogram.methods.export\\_chat\\_invite\\_link](#)), 256  
[extract\\_flags\(\)](#) (in module [aiogram.dispatcher.flags](#)), 438  
[extract\\_from\(\)](#) ([aiogram.types.message\\_entity.MessageEntity](#) method), 204  
[EYES](#) ([aiogram.enums.mask\\_position\\_point.MaskPositionPoint](#) attribute), 385

## F

[feed\\_raw\\_update\(\)](#) ([aiogram.dispatcher.dispatcher.Dispatcher](#) method), 398  
[feed\\_update\(\)](#) ([aiogram.dispatcher.dispatcher.Dispatcher](#) method), 398  
[field\\_name](#) ([aiogram.types.passport\\_element\\_error\\_data\\_field.PassportElementErrorDataField](#) attribute), 218  
[file](#) ([aiogram.client.telegram.TelegramAPIServer](#) attribute), 12  
[FILE](#) ([aiogram.enums.passport\\_element\\_error\\_type.PassportElementErrorType](#) attribute), 386  
[File](#) (class in [aiogram.types.file](#)), 144  
[file\\_date](#) ([aiogram.types.passport\\_file.PassportFile](#) attribute), 225  
[file\\_hash](#) ([aiogram.types.passport\\_element\\_error\\_file.PassportElementErrorFile](#) attribute), 219  
[file\\_hash](#) ([aiogram.types.passport\\_element\\_error\\_front\\_side.PassportElementErrorFrontSide](#) attribute), 220  
[file\\_hash](#) ([aiogram.types.passport\\_element\\_error\\_reverse\\_side.PassportElementErrorReverseSide](#) attribute), 221  
[file\\_hash](#) ([aiogram.types.passport\\_element\\_error\\_selfie.PassportElementErrorSelfie](#) attribute), 222  
[file\\_hash](#) ([aiogram.types.passport\\_element\\_error\\_translation\\_file.PassportElementErrorTranslationFile](#) attribute), 223  
[file\\_hashes](#) ([aiogram.types.passport\\_element\\_error\\_files.PassportElementErrorFiles](#) attribute), 220  
[file\\_hashes](#) ([aiogram.types.passport\\_element\\_error\\_translation\\_files.PassportElementErrorTranslationFiles](#) attribute), 223  
[file\\_id](#) ([aiogram.methods.get\\_file.GetFile](#) attribute), 263  
[file\\_id](#) ([aiogram.types.animation.Animation](#) attribute), 64  
[file\\_id](#) ([aiogram.types.audio.Audio](#) attribute), 64  
[file\\_id](#) ([aiogram.types.document.Document](#) attribute), 144  
[file\\_id](#) ([aiogram.types.file.File](#) attribute), 144  
[file\\_id](#) ([aiogram.types.passport\\_file.PassportFile](#) attribute), 224  
[file\\_id](#) ([aiogram.types.photo\\_size.PhotoSize](#) attribute), 205  
[file\\_id](#) ([aiogram.types.sticker.Sticker](#) attribute), 228  
[file\\_id](#) ([aiogram.types.video.Video](#) attribute), 212  
[file\\_id](#) ([aiogram.types.video\\_note.VideoNote](#) attribute), 214  
[file\\_id](#) ([aiogram.types.voice.Voice](#) attribute), 214  
[file\\_name](#) ([aiogram.types.animation.Animation](#) attribute), 64  
[file\\_name](#) ([aiogram.types.audio.Audio](#) attribute), 65  
[file\\_name](#) ([aiogram.types.document.Document](#) attribute), 144  
[file\\_name](#) ([aiogram.types.video.Video](#) attribute), 213  
[file\\_path](#) ([aiogram.types.file.File](#) attribute), 145  
[file\\_size](#) ([aiogram.types.animation.Animation](#) attribute), 64  
[file\\_size](#) ([aiogram.types.audio.Audio](#) attribute), 65  
[file\\_size](#) ([aiogram.types.document.Document](#) attribute), 144  
[file\\_size](#) ([aiogram.types.file.File](#) attribute), 144  
[file\\_size](#) ([aiogram.types.passport\\_file.PassportFile](#) attribute), 225  
[file\\_size](#) ([aiogram.types.photo\\_size.PhotoSize](#) attribute), 205  
[file\\_size](#) ([aiogram.types.sticker.Sticker](#) attribute), 229  
[file\\_size](#) ([aiogram.types.video.Video](#) attribute), 213  
[file\\_size](#) ([aiogram.types.video\\_note.VideoNote](#) attribute), 214  
[file\\_size](#) ([aiogram.types.voice.Voice](#) attribute), 214  
[file\\_unique\\_id](#) ([aiogram.types.animation.Animation](#) attribute), 64  
[file\\_unique\\_id](#) ([aiogram.types.audio.Audio](#) attribute), 64  
[file\\_unique\\_id](#) ([aiogram.types.document.Document](#) attribute), 144

attribute), 144  
 file\_unique\_id (aiogram.types.file.File attribute), 144  
 file\_unique\_id (aiogram.types.passport\_file.PassportFile attribute), 225  
 file\_unique\_id (aiogram.types.photo\_size.PhotoSize attribute), 205  
 file\_unique\_id (aiogram.types.sticker.Sticker attribute), 228  
 file\_unique\_id (aiogram.types.video.Video attribute), 212  
 file\_unique\_id (aiogram.types.video\_note.VideoNote attribute), 214  
 file\_unique\_id (aiogram.types.voice.Voice attribute), 214  
 file\_url() (aiogram.client.telegram.TelegramAPIServer method), 12  
 FILES (aiogram.enums.passport\_element\_error\_type.PassportElementErrorType attribute), 386  
 files (aiogram.types.encrypted\_passport\_element.EncryptedPassportElement attribute), 216  
 Filter (class in aiogram.filters.base), 412  
 filter() (aiogram.filters.callback\_data.CallbackData class method), 409  
 FIND\_LOCATION (aiogram.enums.chat\_action.ChatAction attribute), 377  
 find\_location() (aiogram.utils.chat\_action.ChatActionSender class method), 453  
 first\_name (aiogram.methods.send\_contact.SendContact attribute), 286  
 first\_name (aiogram.types.chat.Chat attribute), 72  
 first\_name (aiogram.types.contact.Contact attribute), 143  
 first\_name (aiogram.types.inline\_query\_result\_contact.InlineQueryResultContact attribute), 40  
 first\_name (aiogram.types.input\_contact\_message\_content.InputContactMessageContent attribute), 58  
 first\_name (aiogram.types.user.User attribute), 210  
 first\_name (aiogram.utils.web\_app.WebAppUser attribute), 457  
 FOOTBALL (aiogram.enums.dice\_emoji.DiceEmoji attribute), 383  
 FOOTBALL (aiogram.types.dice.DiceEmoji attribute), 143  
 for\_channels (aiogram.methods.get\_my\_default\_administrators.GetMyDefaultAdministrators attribute), 266  
 for\_channels (aiogram.methods.set\_my\_default\_administrators.SetMyDefaultAdministrators attribute), 318  
 force (aiogram.methods.set\_game\_score.SetGameScore attribute), 355  
 force\_reply (aiogram.types.force\_reply.ForceReply attribute), 145  
 ForceReply (class in aiogram.types.force\_reply), 145  
 FOREHEAD (aiogram.enums.mask\_position\_point.MaskPositionPoint attribute), 385  
 FORUM\_TOPIC\_CLOSED (aiogram.enums.content\_type.ContentType attribute), 379  
 forum\_topic\_closed (aiogram.types.message.Message attribute), 165  
 FORUM\_TOPIC\_CREATED (aiogram.enums.content\_type.ContentType attribute), 379  
 forum\_topic\_created (aiogram.types.message.Message attribute), 165  
 FORUM\_TOPIC\_EDITED (aiogram.enums.content\_type.ContentType attribute), 379  
 forum\_topic\_edited (aiogram.types.message.Message attribute), 165  
 FORUM\_TOPIC\_REOPENED (aiogram.enums.content\_type.ContentType attribute), 379  
 forum\_topic\_reopened (aiogram.types.message.Message attribute), 165  
 ForumTopic (class in aiogram.types.forum\_topic), 145  
 ForumTopicClosed (class in aiogram.types.forum\_topic\_closed), 146  
 ForumTopicCreated (class in aiogram.types.forum\_topic\_created), 146  
 ForumTopicEdited (class in aiogram.types.forum\_topic\_edited), 146  
 ForumTopicReopened (class in aiogram.types.forum\_topic\_reopened), 147  
 forward() (aiogram.types.message.Message method), 198  
 forward\_date (aiogram.types.message.Message attribute), 162  
 forward\_from (aiogram.types.message.Message attribute), 162  
 forward\_from\_chat (aiogram.types.message.Message attribute), 162  
 forward\_from\_message\_id (aiogram.types.message.Message attribute), 162  
 forward\_sender\_name (aiogram.types.message.Message attribute), 162  
 forward\_signature (aiogram.types.message.Message attribute), 162  
 forward\_timestamp (aiogram.types.message.Message attribute), 162  
 ForwardMessage (class in aiogram.methods.forward\_message), 257  
 foursquare\_id (aiogram.methods.send\_venue.SendVenue attribute), 302  
 foursquare\_id (aiogram.types.inline\_query\_result\_venue.InlineQueryResultVenue attribute), 53  
 foursquare\_id (aiogram.types.input\_venue\_message\_content.InputVenueMessageContent attribute), 63



foursquare\_id (aiogram.types.venue.Venue attribute), 212  
 foursquare\_type (aiogram.methods.send\_venue.SendVenue attribute), 302  
 foursquare\_type (aiogram.types.inline\_query\_result\_venue.InlineQueryResultVenue attribute), 53  
 foursquare\_type (aiogram.types.input\_venue\_message\_content.InputVenueMessageContent attribute), 63  
 foursquare\_type (aiogram.types.venue.Venue attribute), 212  
 from\_base() (aiogram.client.telegram.TelegramAPIServer class method), 12  
 from\_chat\_id (aiogram.methods.copy\_message.CopyMessage attribute), 244  
 from\_chat\_id (aiogram.methods.forward\_message.ForwardMessage attribute), 257  
 from\_file() (aiogram.types.input\_file.BufferedInputFile class method), 149  
 from\_markup() (aiogram.utils.keyboard.InlineKeyboardBuilder class method), 446  
 from\_markup() (aiogram.utils.keyboard.ReplyKeyboardBuilder class method), 448  
 from\_url() (aiogram.fsm.storage.redis.RedisStorage class method), 431  
 from\_user (aiogram.handlers.callback\_query.CallbackQueryHandler property), 440  
 from\_user (aiogram.types.callback\_query.CallbackQuery attribute), 70  
 from\_user (aiogram.types.chat\_join\_request.ChatJoinRequest attribute), 86  
 from\_user (aiogram.types.chat\_member\_updated.ChatMemberUpdated attribute), 124  
 from\_user (aiogram.types.chosen\_inline\_result.ChosenInlineResult attribute), 17  
 from\_user (aiogram.types.inline\_query.InlineQuery attribute), 18  
 from\_user (aiogram.types.message.Message attribute), 162  
 from\_user (aiogram.types.pre\_checkout\_query.PreCheckoutQuery attribute), 231  
 from\_user (aiogram.types.shipping\_query.ShippingQuery attribute), 233  
 FRONT\_SIDE (aiogram.enums.passport\_element\_error\_type.PassportElementErrorType attribute), 386  
 front\_side (aiogram.types.encrypted\_passport\_element.EncryptedPassportElement attribute), 217  
 FSInputFile (class in aiogram.types.input\_file), 149, 391  
 FSMI18nMiddleware (class in aiogram.utils.i18n.middleware), 450  
 full\_name (aiogram.types.chat.Chat property), 73  
 full\_name (aiogram.types.user.User property), 210

**G**  
 GAME (aiogram.enums.content\_type.ContentType attribute), 378  
 GAME (aiogram.enums.inline\_query\_result\_type.InlineQueryResultType attribute), 384  
 game (aiogram.types.message.Message attribute), 164  
 game (class in aiogram.types.game), 235  
 game\_short\_name (aiogram.methods.send\_game.SendGame attribute), 353  
 game\_short\_name (aiogram.types.callback\_query.CallbackQuery attribute), 70  
 game\_short\_name (aiogram.types.inline\_query\_result\_game.InlineQueryResultGame attribute), 44  
 GameHighScore (class in aiogram.types.game\_high\_score), 236  
 GBP (aiogram.enums.currency.Currency attribute), 381  
 GEL (aiogram.enums.currency.Currency attribute), 381  
 GENERAL\_FORUM\_TOPIC\_HIDDEN (aiogram.enums.content\_type.ContentType attribute), 379  
 general\_forum\_topic\_hidden (aiogram.types.message.Message attribute), 165  
 GENERAL\_FORUM\_TOPIC\_UNHIDDEN (aiogram.enums.content\_type.ContentType attribute), 379  
 general\_forum\_topic\_unhidden (aiogram.types.message.Message attribute), 165  
 GeneralForumTopicHidden (class in aiogram.types.general\_forum\_topic\_hidden), 147  
 GeneralForumTopicUnhidden (class in aiogram.types.general\_forum\_topic\_unhidden), 147  
 get\_administrators() (aiogram.types.chat.Chat method), 74  
 get\_data() (aiogram.fsm.storage.base.BaseStorage method), 433  
 get\_flag() (in module aiogram.dispatcher.flags), 439  
 get\_locale() (aiogram.utils.i18n.middleware.I18nMiddleware method), 451  
 get\_member\_count() (aiogram.types.chat.Chat method), 78  
 get\_member\_count() (aiogram.types.chat.Chat method), 78  
 get\_profile\_photos() (aiogram.types.user.User method), 210  
 get\_state() (aiogram.fsm.storage.base.BaseStorage method), 432  
 get\_url() (aiogram.types.message.Message method), 203  
 GetChat (class in aiogram.methods.get\_chat), 259  
 GetChatAdministrators (class in aiogram.methods.get\_chat\_administrators),

259

GetChatMember (class in aiogram.methods.get\_chat\_member), 260

GetChatMemberCount (class in aiogram.methods.get\_chat\_member\_count), 261

GetChatMenuButton (class in aiogram.methods.get\_chat\_menu\_button), 262

GetCustomEmojiStickers (class in aiogram.methods.get\_custom\_emoji\_stickers), 341

GetFile (class in aiogram.methods.get\_file), 263

GetForumTopicIconStickers (class in aiogram.methods.get\_forum\_topic\_icon\_stickers), 264

GetGameHighScores (class in aiogram.methods.get\_game\_high\_scores), 352

GetMe (class in aiogram.methods.get\_me), 264

GetMyCommands (class in aiogram.methods.get\_my\_commands), 265

GetMyDefaultAdministratorRights (class in aiogram.methods.get\_my\_default\_administrator\_rights), 266

GetMyDescription (class in aiogram.methods.get\_my\_description), 267

GetMyName (class in aiogram.methods.get\_my\_name), 268

GetMyShortDescription (class in aiogram.methods.get\_my\_short\_description), 268

GetStickerSet (class in aiogram.methods.get\_sticker\_set), 342

GetUpdates (class in aiogram.methods.get\_updates), 356

GetUserProfilePhotos (class in aiogram.methods.get\_user\_profile\_photos), 269

GetWebhookInfo (class in aiogram.methods.get\_webhook\_info), 358

GIF (aiogram.enums.inline\_query\_result\_type.InlineQueryResultType attribute), 384

gif\_duration (aiogram.types.inline\_query\_result\_gif.InlineQueryResultGif attribute), 45

gif\_file\_id (aiogram.types.inline\_query\_result\_cached\_gif.InlineQueryResultCachedGif attribute), 28

gif\_height (aiogram.types.inline\_query\_result\_gif.InlineQueryResultGif attribute), 45

gif\_url (aiogram.types.inline\_query\_result\_gif.InlineQueryResultGif attribute), 44

gif\_width (aiogram.types.inline\_query\_result\_gif.InlineQueryResultGif attribute), 45

google\_place\_id (aiogram.methods.send\_venue.SendVenue attribute), 151

google\_place\_id (aiogram.types.inline\_query\_result\_venue.InlineQueryResultVenue attribute), 302

google\_place\_id (aiogram.types.input\_venue\_message\_content.InputVenueMessageContent attribute), 63

google\_place\_id (aiogram.types.venue.Venue attribute), 212

google\_place\_type (aiogram.methods.send\_venue.SendVenue attribute), 302

google\_place\_type (aiogram.types.inline\_query\_result\_venue.InlineQueryResultVenue attribute), 53

google\_place\_type (aiogram.types.input\_venue\_message\_content.InputVenueMessageContent attribute), 63

google\_place\_type (aiogram.types.venue.Venue attribute), 212

GREEN (aiogram.enums.topic\_icon\_color.TopicIconColor attribute), 387

GROUP (aiogram.enums.chat\_type.ChatType attribute), 378

GROUP\_CHAT\_CREATED (aiogram.enums.content\_type.ContentType attribute), 379

group\_chat\_created (aiogram.types.message.Message attribute), 164

GROUP (aiogram.enums.currency.Currency attribute), 381

## H

has\_aggressive\_anti\_spam\_enabled (aiogram.types.chat.Chat attribute), 73

has\_custom\_certificate (aiogram.types.webhook\_info.WebhookInfo attribute), 227

has\_hidden\_members (aiogram.types.chat.Chat attribute), 73

HAS\_MEDIA\_SPOILER (aiogram.enums.content\_type.ContentType attribute), 378

has\_media\_spoiler (aiogram.types.message.Message attribute), 163

has\_private\_forwards (aiogram.types.chat.Chat attribute), 72

has\_protected\_content (aiogram.types.chat.Chat attribute), 73

has\_protected\_content (aiogram.types.message.Message attribute), 163

has\_restricted\_voice\_and\_video\_messages (aiogram.types.chat.Chat attribute), 72

has\_spoiler (aiogram.methods.send\_animation.SendAnimation attribute), 281

has\_spoiler (aiogram.methods.send\_photo.SendPhoto attribute), 297

has\_spoiler (aiogram.methods.send\_video.SendVideo attribute), 304

has\_spoiler (aiogram.types.input\_media\_animation.InputMediaAnimation attribute), 151

has_spoiler (aiogram.types.input_media_photo.InputMediaPhoto attribute), 153	has_spoiler (aiogram.types.input_location_message_content.InputLocationMessageContent attribute), 61
has_spoiler (aiogram.types.input_media_video.InputMediaVideo attribute), 154	horizontal_accuracy (aiogram.types.location.Location attribute), 158
hash (aiogram.types.encrypted_credentials.EncryptedCredentials attribute), 216	HRK (aiogram.enums.currency.Currency attribute), 381
hash (aiogram.types.encrypted_passport_element.EncryptedPassportElement attribute), 216	HTML (aiogram.enums.parse_mode.ParseMode attribute), 386
hash (aiogram.utils.web_app.WebAppInitData attribute), 457	html_text (aiogram.types.message.Message property), 166
HASHTAG (aiogram.enums.message_entity_type.MessageEntityType attribute), 385	HUF (aiogram.enums.currency.Currency attribute), 381
HashTag (class in aiogram.utils.formatting), 465	HTTPMiddleware (class in aiogram.utils.middleware), 451
heading (aiogram.methods.edit_message_live_location.EditMessageLiveLocation attribute), 365	icon_color (aiogram.methods.create_forum_topic.CreateForumTopic attribute), 247
heading (aiogram.methods.send_location.SendLocation attribute), 292	icon_color (aiogram.types.forum_topic.ForumTopic attribute), 146
heading (aiogram.types.inline_query_result_location.InlineQueryResultLocation attribute), 47	icon_color (aiogram.types.forum_topic_created.ForumTopicCreated attribute), 146
heading (aiogram.types.input_location_message_content.InputLocationMessageContent attribute), 61	icon_custom_emoji_id (aiogram.methods.create_forum_topic.CreateForumTopic attribute), 247
heading (aiogram.types.location.Location attribute), 158	icon_custom_emoji_id (aiogram.methods.edit_forum_topic.EditForumTopic attribute), 255
height (aiogram.methods.send_animation.SendAnimation attribute), 281	icon_custom_emoji_id (aiogram.types.forum_topic.ForumTopic attribute), 146
height (aiogram.methods.send_video.SendVideo attribute), 304	icon_custom_emoji_id (aiogram.types.forum_topic_created.ForumTopicCreated attribute), 146
height (aiogram.types.animation.Animation attribute), 64	icon_custom_emoji_id (aiogram.types.forum_topic_edited.ForumTopicEdited attribute), 146
height (aiogram.types.input_media_animation.InputMediaAnimation attribute), 150	id (aiogram.types.callback_query.CallbackQuery attribute), 70
height (aiogram.types.input_media_video.InputMediaVideo attribute), 154	id (aiogram.types.chat.Chat attribute), 71
height (aiogram.types.photo_size.PhotoSize attribute), 205	id (aiogram.types.inline_query.InlineQuery attribute), 18
height (aiogram.types.sticker.Sticker attribute), 228	id (aiogram.types.inline_query_result_article.InlineQueryResultArticle attribute), 20
height (aiogram.types.video.Video attribute), 212	id (aiogram.types.inline_query_result_audio.InlineQueryResultAudio attribute), 22
hide_url (aiogram.types.inline_query_result_article.InlineQueryResultArticle attribute), 21	id (aiogram.types.inline_query_result_cached_audio.InlineQueryResultCachedAudio attribute), 24
HideGeneralForumTopic (class in aiogram.methods.hide_general_forum_topic), 270	id (aiogram.types.inline_query_result_cached_document.InlineQueryResultCachedDocument attribute), 27
HKD (aiogram.enums.currency.Currency attribute), 381	id (aiogram.types.inline_query_result_cached_gif.InlineQueryResultCachedGif attribute), 28
HNL (aiogram.enums.currency.Currency attribute), 381	id (aiogram.types.inline_query_result_cached_mpeg4_gif.InlineQueryResultCachedMpeg4Gif attribute), 31
horizontal_accuracy (aiogram.methods.edit_message_live_location.EditMessageLiveLocation attribute), 364	id (aiogram.types.inline_query_result_cached_photo.InlineQueryResultCachedPhoto attribute), 33
horizontal_accuracy (aiogram.methods.send_location.SendLocation attribute), 292	
horizontal_accuracy (aiogram.types.inline_query_result_location.InlineQueryResultLocation attribute), 47	
horizontal_accuracy (aiogram.types.inline_query_result_location.InlineQueryResultLocation attribute), 47	

[id \(aiogram.types.inline\\_query\\_result\\_cached\\_sticker.InlineQueryResultCachedSticker attribute\), 34](#)  
[id \(aiogram.types.inline\\_query\\_result\\_cached\\_video.InlineQueryResultCachedVideo attribute\), 37](#)  
[id \(aiogram.types.inline\\_query\\_result\\_cached\\_voice.InlineQueryResultCachedVoice attribute\), 38](#)  
[id \(aiogram.types.inline\\_query\\_result\\_contact.InlineQueryResultContact attribute\), 370](#)  
[id \(aiogram.types.inline\\_query\\_result\\_document.InlineQueryResultDocument attribute\), 42](#)  
[id \(aiogram.types.inline\\_query\\_result\\_game.InlineQueryResultGame attribute\), 17](#)  
[id \(aiogram.types.inline\\_query\\_result\\_gif.InlineQueryResultGif attribute\), 63](#)  
[id \(aiogram.types.inline\\_query\\_result\\_location.InlineQueryResultLocation attribute\), 388](#)  
[id \(aiogram.types.inline\\_query\\_result\\_mpeg4\\_gif.InlineQueryResultMpeg4Gif attribute\), 49](#)  
[id \(aiogram.types.inline\\_query\\_result\\_photo.InlineQueryResultPhoto attribute\), 373](#)  
[id \(aiogram.types.inline\\_query\\_result\\_venue.InlineQueryResultVenue attribute\), 52](#)  
[id \(aiogram.types.inline\\_query\\_result\\_video.InlineQueryResultVideo attribute\), 54](#)  
[id \(aiogram.types.inline\\_query\\_result\\_voice.InlineQueryResultVoice attribute\), 56](#)  
[id \(aiogram.types.poll.Poll attribute\), 205](#)  
[id \(aiogram.types.pre\\_checkout\\_query.PreCheckoutQuery attribute\), 231](#)  
[id \(aiogram.types.shipping\\_option.ShippingOption attribute\), 233](#)  
[id \(aiogram.types.shipping\\_query.ShippingQuery attribute\), 233](#)  
[id \(aiogram.types.user.User attribute\), 210](#)  
[id \(aiogram.utils.web\\_app.WebAppUser attribute\), 457](#)  
[IDENTITY\\_CARD \(aiogram.enums.encrypted\\_passport\\_element\\_type attribute\), 383](#)  
[IDR \(aiogram.enums.currency.Currency attribute\), 381](#)  
[ILS \(aiogram.enums.currency.Currency attribute\), 381](#)  
[include\\_router\(\) \(aiogram.dispatcher.router.Router method\), 393](#)  
[include\\_routers\(\) \(aiogram.dispatcher.router.Router method\), 393](#)  
[inline\\_keyboard \(aiogram.types.inline\\_keyboard\\_markup.InlineKeyboardMarkup attribute\), 148](#)  
[inline\\_message\\_id \(aiogram.methods.edit\\_message\\_caption.EditMessageCaption attribute\), 363](#)  
[inline\\_message\\_id \(aiogram.methods.edit\\_message\\_live\\_location.EditMessageLiveLocation attribute\), 364](#)  
[inline\\_message\\_id \(aiogram.methods.edit\\_message\\_media.EditMessageMedia attribute\), 366](#)  
[inline\\_message\\_id \(aiogram.methods.edit\\_message\\_reply\\_markup.EditMessageReplyMarkup attribute\), 367](#)  
[inline\\_message\\_id \(aiogram.methods.edit\\_message\\_text.EditMessageText attribute\), 368](#)  
[inline\\_message\\_id \(aiogram.methods.get\\_game\\_high\\_scores.GetGameHighScores attribute\), 370](#)  
[inline\\_message\\_id \(aiogram.methods.set\\_game\\_score.SetGameScore attribute\), 372](#)  
[inline\\_message\\_id \(aiogram.methods.stop\\_message\\_live\\_location.StopMessageLiveLocation attribute\), 375](#)  
[inline\\_message\\_id \(aiogram.types.callback\\_query.CallbackQuery attribute\), 70](#)  
[inline\\_message\\_id \(aiogram.types.chosen\\_inline\\_result.ChosenInlineResult attribute\), 17](#)  
[inline\\_message\\_id \(aiogram.types.sent\\_web\\_app\\_message.SentWebAppMessage attribute\), 172](#)  
[inline\\_query \(aiogram.enums.update\\_type.UpdateType attribute\), 388](#)  
[inline\\_query\\_id \(aiogram.methods.answer\\_inline\\_query.AnswerInlineQuery attribute\), 373](#)  
[InlineKeyboardBuilder \(class in aiogram.types.inline\\_keyboard\\_markup\), 446](#)  
[InlineKeyboardButton \(class in aiogram.types.inline\\_keyboard\\_markup\), 147](#)  
[InlineKeyboardMarkup \(class in aiogram.types.inline\\_keyboard\\_markup\), 148](#)  
[InlineQuery \(class in aiogram.types.inline\\_query\), 18](#)  
[InlineQueryResult \(class in aiogram.types.inline\\_query\\_result\), 19](#)  
[InlineQueryResultArticle \(class in aiogram.types.inline\\_query\\_result\\_article\), 20](#)  
[InlineQueryResultAudio \(class in aiogram.types.inline\\_query\\_result\\_audio\), 21](#)  
[InlineQueryResultCachedAudio \(class in aiogram.types.inline\\_query\\_result\\_cached\\_audio\), 23](#)  
[InlineQueryResultCachedDocument \(class in aiogram.types.inline\\_query\\_result\\_cached\\_document\), 25](#)  
[InlineQueryResultCachedGif \(class in aiogram.types.inline\\_query\\_result\\_cached\\_gif\), 29](#)  
[InlineQueryResultCachedMpeg4Gif \(class in aiogram.types.inline\\_query\\_result\\_cached\\_mpeg4\\_gif\), 29](#)  
[InlineQueryResultCachedPhoto \(class in aiogram.types.inline\\_query\\_result\\_cached\\_photo\), 33](#)  
[InlineQueryResultCachedSticker \(class in aiogram.types.inline\\_query\\_result\\_cached\\_sticker\), 34](#)  
[InlineQueryResultCachedVideo \(class in aiogram.types.inline\\_query\\_result\\_cached\\_video\), 37](#)



[aiogram.types.inline\\_query\\_result\\_cached\\_video](#))[input\\_message\\_content](#)  
35 ([aiogram.types.inline\\_query\\_result\\_cached\\_document.InlineQueryResultCachedDocument](#) attribute), 27  
[InlineQueryResultCachedVoice](#) (class in [aiogram.types.inline\\_query\\_result\\_cached\\_voice](#))[input\\_message\\_content](#)  
37 ([aiogram.types.inline\\_query\\_result\\_cached\\_gif.InlineQueryResultCachedGif](#) attribute), 29  
[InlineQueryResultContact](#) (class in [aiogram.types.inline\\_query\\_result\\_contact](#)), [input\\_message\\_content](#)  
39 ([aiogram.types.inline\\_query\\_result\\_cached\\_mpeg4\\_gif.InlineQueryResultCachedMpeg4Gif](#) attribute), 31  
[InlineQueryResultDocument](#) (class in [aiogram.types.inline\\_query\\_result\\_document](#)), [input\\_message\\_content](#)  
41 ([aiogram.types.inline\\_query\\_result\\_cached\\_photo.InlineQueryResultCachedPhoto](#) attribute), 33  
[InlineQueryResultGame](#) (class in [aiogram.types.inline\\_query\\_result\\_game](#)), [input\\_message\\_content](#)  
43 ([aiogram.types.inline\\_query\\_result\\_cached\\_sticker.InlineQueryResultCachedSticker](#) attribute), 35  
[InlineQueryResultGif](#) (class in [aiogram.types.inline\\_query\\_result\\_gif](#)), [input\\_message\\_content](#)  
44 ([aiogram.types.inline\\_query\\_result\\_cached\\_video.InlineQueryResultCachedVideo](#) attribute), 37  
[InlineQueryResultLocation](#) (class in [aiogram.types.inline\\_query\\_result\\_location](#)), [input\\_message\\_content](#)  
45 ([aiogram.types.inline\\_query\\_result\\_cached\\_voice.InlineQueryResultCachedVoice](#) attribute), 39  
[InlineQueryResultMpeg4Gif](#) (class in [aiogram.types.inline\\_query\\_result\\_mpeg4\\_gif](#)), [input\\_message\\_content](#)  
48 ([aiogram.types.inline\\_query\\_result\\_contact.InlineQueryResultContact](#) attribute), 41  
[InlineQueryResultPhoto](#) (class in [aiogram.types.inline\\_query\\_result\\_photo](#)), [input\\_message\\_content](#)  
50 ([aiogram.types.inline\\_query\\_result\\_document.InlineQueryResultDocument](#) attribute), 43  
[InlineQueryResultsButton](#) (class in [aiogram.types.inline\\_query\\_results\\_button](#)), 57  
[InlineQueryResultType](#) (class in [aiogram.enums.inline\\_query\\_result\\_type](#)), [input\\_message\\_content](#)  
384 ([aiogram.types.inline\\_query\\_result\\_gif.InlineQueryResultGif](#) attribute), 45  
[InlineQueryResultVenue](#) (class in [aiogram.types.inline\\_query\\_result\\_venue](#)), [input\\_message\\_content](#)  
51 ([aiogram.types.inline\\_query\\_result\\_location.InlineQueryResultLocation](#) attribute), 47  
[InlineQueryResultVideo](#) (class in [aiogram.types.inline\\_query\\_result\\_video](#)), [input\\_message\\_content](#)  
53 ([aiogram.types.inline\\_query\\_result\\_mpeg4\\_gif.InlineQueryResultMpeg4Gif](#) attribute), 49  
[InlineQueryResultVoice](#) (class in [aiogram.types.inline\\_query\\_result\\_voice](#)), [input\\_message\\_content](#)  
56 ([aiogram.types.inline\\_query\\_result\\_photo.InlineQueryResultPhoto](#) attribute), 51  
[input\\_field\\_placeholder](#) [input\\_message\\_content](#)  
([aiogram.types.force\\_reply.ForceReply](#) attribute), 145 ([aiogram.types.inline\\_query\\_result\\_venue.InlineQueryResultVenue](#) attribute), 53  
[input\\_field\\_placeholder](#) [input\\_message\\_content](#)  
([aiogram.types.reply\\_keyboard\\_markup.ReplyKeyboardMarkup](#) attribute), 207 ([aiogram.types.inline\\_query\\_result\\_video.InlineQueryResultVideo](#) attribute), 55  
[input\\_message\\_content](#) [input\\_message\\_content](#)  
([aiogram.types.inline\\_query\\_result\\_article.InlineQueryResultArticle](#) attribute), 20 ([aiogram.types.inline\\_query\\_result\\_voice.InlineQueryResultVoice](#) attribute), 57  
[input\\_message\\_content](#) [InputContactMessageContent](#) (class in [aiogram.types.input\\_contact\\_message\\_content](#)),  
attribute), 23 57  
[input\\_message\\_content](#) [InputFile](#) (class in [aiogram.types.input\\_file](#)), 149  
([aiogram.types.inline\\_query\\_result\\_cached\\_audio.InlineQueryResultCachedAudio](#) attribute), 25 ([aiogram.types.input\\_invoice\\_message\\_content](#)),

- 58
- `InputLocationMessageContent` (class in `aiogram.types.input_location_message_content`), 61
- `InputMedia` (class in `aiogram.types.input_media`), 149
- `InputMediaAnimation` (class in `aiogram.types.input_media_animation`), 150
- `InputMediaAudio` (class in `aiogram.types.input_media_audio`), 151
- `InputMediaDocument` (class in `aiogram.types.input_media_document`), 152
- `InputMediaPhoto` (class in `aiogram.types.input_media_photo`), 153
- `InputMediaType` (class in `aiogram.enums.input_media_type`), 384
- `InputMediaVideo` (class in `aiogram.types.input_media_video`), 153
- `InputMessageContent` (class in `aiogram.types.input_message_content`), 62
- `InputSticker` (class in `aiogram.types.input_sticker`), 227
- `InputTextMessageContent` (class in `aiogram.types.input_text_message_content`), 62
- `InputVenueMessageContent` (class in `aiogram.types.input_venue_message_content`), 63
- `INR` (`aiogram.enums.currency.Currency` attribute), 381
- `INTERNAL_PASSPORT` (`aiogram.enums.encrypted_passport_encrypted_passport_element_type.EncryptedPassportElement` attribute), 383
- `invite_link` (`aiogram.methods.edit_chat_invite_link.EditChatInviteLink` attribute), 253
- `invite_link` (`aiogram.methods.revoke_chat_invite_link.RevokeChatInviteLink` attribute), 279
- `invite_link` (`aiogram.types.chat.Chat` attribute), 72
- `invite_link` (`aiogram.types.chat_invite_link.ChatInviteLink` attribute), 85
- `invite_link` (`aiogram.types.chat_join_request.ChatJoinRequest` attribute), 86
- `invite_link` (`aiogram.types.chat_member_updated.ChatMemberUpdated` attribute), 125
- `INVOICE` (`aiogram.enums.content_type.ContentType` attribute), 379
- `invoice` (`aiogram.types.message.Message` attribute), 165
- `Invoice` (class in `aiogram.types.invoice`), 230
- `invoice_payload` (`aiogram.types.pre_checkout_query.PreCheckoutQuery` attribute), 232
- `invoice_payload` (`aiogram.types.shipping_query.ShippingQuery` attribute), 233
- `invoice_payload` (`aiogram.types.successful_payment.SuccessfulPayment` attribute), 234
- `ip_address` (`aiogram.methods.set_webhook.SetWebhook` attribute), 359
- `ip_address` (`aiogram.types.webhook_info.WebhookInfo` attribute), 227
- `ip_filter_middleware()` (in module `aiogram.webhook.aiohttp_server`), 418
- `IPFilter` (class in `aiogram.webhook.security`), 418
- `is_animated` (`aiogram.types.sticker.Sticker` attribute), 228
- `is_animated` (`aiogram.types.sticker_set.StickerSet` attribute), 230
- `is_anonymous` (`aiogram.methods.promote_chat_member.PromoteChatMember` attribute), 274
- `is_anonymous` (`aiogram.methods.send_poll.SendPoll` attribute), 299
- `is_anonymous` (`aiogram.types.chat_administrator_rights.ChatAdministratorRights` attribute), 84
- `is_anonymous` (`aiogram.types.chat_member_administrator.ChatMemberAdministrator` attribute), 120
- `is_anonymous` (`aiogram.types.chat_member_owner.ChatMemberOwner` attribute), 122
- `is_anonymous` (`aiogram.types.poll.Poll` attribute), 205
- `is_automatic_forward` (`aiogram.types.message.Message` attribute), 162
- `is_bot` (`aiogram.types.user.User` attribute), 210
- `is_bot` (`aiogram.utils.web_app.WebAppUser` attribute), 457
- `is_closed` (`aiogram.methods.send_poll.SendPoll` attribute), 300
- `is_closed` (`aiogram.types.poll.Poll` attribute), 205
- `is_flexible` (`aiogram.methods.send_invoice.SendInvoice` attribute), 333
- `is_flexible` (`aiogram.types.input_invoice_message_content.InputInvoiceMessageContent` attribute), 61
- `is_forum` (`aiogram.types.chat.Chat` attribute), 72
- `is_local` (`aiogram.client.telegram.TelegramAPIServer` attribute), 12
- `is_member` (`aiogram.types.chat_member_restricted.ChatMemberRestricted` attribute), 123
- `is_persistent` (`aiogram.types.reply_keyboard_markup.ReplyKeyboardMarkup` attribute), 207
- `is_personal` (`aiogram.methods.answer_inline_query.AnswerInlineQuery` attribute), 373
- `is_premium` (`aiogram.types.user.User` attribute), 210
- `is_primary` (`aiogram.types.chat_invite_link.ChatInviteLink` attribute), 85
- `is_revoked` (`aiogram.types.chat_invite_link.ChatInviteLink` attribute), 230
- `is_topic_message` (`aiogram.types.message.Message` attribute), 162
- `is_video` (`aiogram.types.sticker.Sticker` attribute), 228
- `is_video` (`aiogram.types.sticker_set.StickerSet` attribute), 230
- `ISK` (`aiogram.enums.currency.Currency` attribute), 381

- ITALIC (*aiogram.enums.message\_entity\_type.MessageEntityType attribute*), 385
- Italic (*class in aiogram.utils.formatting*), 466
- ## J
- JMD (*aiogram.enums.currency.Currency attribute*), 381
- join\_by\_request (*aiogram.types.chat.Chat attribute*), 72
- join\_to\_send\_messages (*aiogram.types.chat.Chat attribute*), 72
- JPY (*aiogram.enums.currency.Currency attribute*), 381
- ## K
- KES (*aiogram.enums.currency.Currency attribute*), 381
- keyboard (*aiogram.types.reply\_keyboard\_markup.ReplyKeyboardMarkup attribute*), 207
- KeyboardButton (*class in aiogram.types.keyboard\_button*), 155
- KeyboardButtonPollType (*class in aiogram.types.keyboard\_button\_poll\_type*), 156
- KeyboardButtonRequestChat (*class in aiogram.types.keyboard\_button\_request\_chat*), 156
- KeyboardButtonRequestUser (*class in aiogram.types.keyboard\_button\_request\_user*), 157
- KeyBuilder (*class in aiogram.fsm.storage.redis*), 432
- keywords (*aiogram.methods.set\_sticker\_keywords.SetStickerKeywords attribute*), 346
- keywords (*aiogram.types.input\_sticker.InputSticker attribute*), 227
- KGS (*aiogram.enums.currency.Currency attribute*), 381
- KICKED (*aiogram.enums.chat\_member\_status.ChatMemberStatus attribute*), 378
- KRW (*aiogram.enums.currency.Currency attribute*), 381
- KZT (*aiogram.enums.currency.Currency attribute*), 381
- ## L
- label (*aiogram.types.labeled\_price.LabeledPrice attribute*), 231
- LabeledPrice (*class in aiogram.types.labeled\_price*), 231
- language (*aiogram.types.message\_entity.MessageEntityType attribute*), 204
- language\_code (*aiogram.methods.delete\_my\_commands.DeleteMyCommands attribute*), 252
- language\_code (*aiogram.methods.get\_my\_commands.GetMyCommands attribute*), 265
- language\_code (*aiogram.methods.get\_my\_description.GetMyDescription attribute*), 267
- language\_code (*aiogram.methods.get\_my\_name.GetMyName attribute*), 268
- language\_code (*aiogram.methods.get\_my\_short\_description.GetMyShortDescription attribute*), 268
- language\_code (*aiogram.methods.set\_my\_commands.SetMyCommands attribute*), 317
- language\_code (*aiogram.methods.set\_my\_description.SetMyDescription attribute*), 319
- language\_code (*aiogram.methods.set\_my\_name.SetMyName attribute*), 320
- language\_code (*aiogram.methods.set\_my\_short\_description.SetMyShortDescription attribute*), 321
- language\_code (*aiogram.types.user.User attribute*), 210
- language\_code (*aiogram.utils.web\_app.WebAppUser attribute*), 457
- last\_error\_date (*aiogram.types.webhook\_info.WebhookInfo attribute*), 227
- last\_error\_message (*aiogram.types.webhook\_info.WebhookInfo attribute*), 227
- last\_name (*aiogram.methods.send\_contact.SendContact attribute*), 286
- last\_name (*aiogram.types.chat.Chat attribute*), 72
- last\_name (*aiogram.types.contact.Contact attribute*), 143
- last\_name (*aiogram.types.inline\_query\_result\_contact.InlineQueryResultContact attribute*), 40
- last\_name (*aiogram.types.input\_contact\_message\_content.InputContactMessageContent attribute*), 58
- last\_name (*aiogram.types.user.User attribute*), 210
- last\_name (*aiogram.utils.web\_app.WebAppUser attribute*), 457
- last\_synchronization\_error\_date (*aiogram.types.webhook\_info.WebhookInfo attribute*), 227
- latitude (*aiogram.methods.edit\_message\_live\_location.EditMessageLiveLocation attribute*), 364
- latitude (*aiogram.methods.send\_location.SendLocation attribute*), 292
- latitude (*aiogram.methods.send\_venue.SendVenue attribute*), 301
- latitude (*aiogram.types.inline\_query\_result\_location.InlineQueryResultLocation attribute*), 46
- latitude (*aiogram.types.inline\_query\_result\_venue.InlineQueryResultVenue attribute*), 52
- latitude (*aiogram.types.input\_location\_message\_content.InputLocationMessageContent attribute*), 61
- latitude (*aiogram.types.input\_venue\_message\_content.InputVenueMessageContent attribute*), 63
- latitude (*aiogram.types.location.Location attribute*), 158
- LBP (*aiogram.enums.currency.Currency attribute*), 381
- leave() (*aiogram.types.chat.Chat method*), 78
- LeaveChat (*class in aiogram.methods.leave\_chat*), 271
- LEFT (*aiogram.enums.chat\_member\_status.ChatMemberStatus attribute*), 377
- LEFT\_CHAT\_MEMBER (*aiogram.enums.content\_type.ContentType*

attribute), 379  
 left\_chat\_member (aiogram.types.message.Message attribute), 164  
 length (aiogram.methods.send\_video\_note.SendVideoNote attribute), 306  
 length (aiogram.types.message\_entity.MessageEntity attribute), 204  
 length (aiogram.types.video\_note.VideoNote attribute), 214  
 limit (aiogram.methods.get\_updates.GetUpdates attribute), 357  
 limit (aiogram.methods.get\_user\_profile\_photos.GetUserProfilePhotos attribute), 269  
 linked\_chat\_id (aiogram.types.chat.Chat attribute), 73  
 live\_period (aiogram.methods.send\_location.SendLocation attribute), 292  
 live\_period (aiogram.types.inline\_query\_result\_location.InlineQueryResultLocation attribute), 47  
 live\_period (aiogram.types.input\_location\_message\_content.InputLocationMessageContent attribute), 61  
 live\_period (aiogram.types.location.Location attribute), 158  
 LKR (aiogram.enums.currency.Currency attribute), 381  
 LOCATION (aiogram.enums.content\_type.ContentType attribute), 379  
 LOCATION (aiogram.enums.inline\_query\_result\_type.InlineQueryResultType attribute), 384  
 location (aiogram.types.chat.Chat attribute), 73  
 location (aiogram.types.chat\_location.ChatLocation attribute), 119  
 location (aiogram.types.chosen\_inline\_result.ChosenInlineResult attribute), 17  
 location (aiogram.types.inline\_query.InlineQuery attribute), 18  
 location (aiogram.types.message.Message attribute), 164  
 location (aiogram.types.venue.Venue attribute), 212  
 Location (class in aiogram.types.location), 158  
 login\_url (aiogram.types.inline\_keyboard\_button.InlineKeyboardButton attribute), 148  
 LoginUrl (class in aiogram.types.login\_url), 158  
 Logout (class in aiogram.methods.log\_out), 272  
 longitude (aiogram.methods.edit\_message\_live\_location.EditMessageLiveLocation attribute), 364  
 longitude (aiogram.methods.send\_location.SendLocation attribute), 292  
 longitude (aiogram.methods.send\_venue.SendVenue attribute), 301  
 longitude (aiogram.types.inline\_query\_result\_location.InlineQueryResultLocation attribute), 46  
 longitude (aiogram.types.inline\_query\_result\_venue.InlineQueryResultVenue attribute), 52  
 longitude (aiogram.types.input\_location\_message\_content.InputLocationMessageContent attribute), 61  
 longitude (aiogram.types.input\_venue\_message\_content.InputVenueMessageContent attribute), 63  
 longitude (aiogram.types.location.Location attribute), 158

## M

MAD (aiogram.enums.currency.Currency attribute), 381  
 magic\_data (aiogram.filters.magic\_data.MagicData attribute), 408  
 magic\_result (aiogram.filters.command.CommandObject attribute), 403  
 MagicData (class in aiogram.filters.magic\_data), 408  
 make\_request() (aiogram.client.session.base.BaseSession method), 13  
 MARKDOWN (aiogram.enums.parse\_mode.ParseMode attribute), 386  
 MARKDOWN\_V2 (aiogram.enums.parse\_mode.ParseMode attribute), 386  
 MaskPosition (class in aiogram.types.mask\_position), 228  
 mask\_position (aiogram.methods.set\_sticker\_mask\_position.SetStickerMaskPosition attribute), 347  
 mask\_position (aiogram.types.input\_sticker.InputSticker attribute), 227  
 mask\_position (aiogram.types.sticker.Sticker attribute), 227  
 MaskPositionPoint (class in aiogram.enums.mask\_position\_point), 385  
 max\_connections (aiogram.methods.set\_webhook.SetWebhook attribute), 359  
 max\_connections (aiogram.types.webhook\_info.WebhookInfo attribute), 227  
 max\_tip\_amount (aiogram.methods.create\_invoice\_link.CreateInvoiceLink attribute), 332  
 max\_tip\_amount (aiogram.methods.send\_invoice.SendInvoice attribute), 335  
 max\_tip\_amount (aiogram.types.input\_invoice\_message\_content.InputInvoiceMessageContent attribute), 60  
 md\_text (aiogram.types.message.Message property), 166  
 MDL (aiogram.enums.currency.Currency attribute), 381  
 media (aiogram.methods.edit\_message\_media.EditMessageMedia attribute), 366  
 media (aiogram.methods.send\_media\_group.SendMediaGroup attribute), 294  
 media (aiogram.types.input\_media\_animation.InputMediaAnimation attribute), 151  
 media (aiogram.types.input\_media\_audio.InputMediaAudio attribute), 151  
 media (aiogram.types.input\_media\_document.InputMediaDocument attribute), 151



media (aiogram.types.input\_media\_photo.InputMediaPhoto attribute), 220  
 media (aiogram.types.input\_media\_video.InputMediaVideo attribute), 153  
 media\_group\_id (aiogram.types.message.Message attribute), 163  
 MEMBER (aiogram.enums.chat\_member\_status.ChatMemberStatus attribute), 377  
 member\_limit (aiogram.methods.create\_chat\_invite\_link.CreateChatInviteLink attribute), 246  
 member\_limit (aiogram.methods.edit\_chat\_invite\_link.EditChatInviteLink attribute), 253  
 member\_limit (aiogram.types.chat\_invite\_link.ChatInviteLink attribute), 86  
 member\_status\_changed (aiogram.filters.chat\_member\_updated.ChatMemberUpdated attribute), 404  
 MemoryStorage (class in aiogram.fsm.storage.memory), 431  
 MENTION (aiogram.enums.message\_entity\_type.MessageEntityType attribute), 385  
 mention (aiogram.filters.command.CommandObject attribute), 403  
 mention\_html() (aiogram.types.user.User method), 210  
 mention\_markdown() (aiogram.types.user.User method), 210  
 mentioned (aiogram.filters.command.CommandObject property), 403  
 menu\_button (aiogram.methods.set\_chat\_menu\_button.SetChatMenuButton attribute), 312  
 MenuButton (class in aiogram.types.menu\_button), 159  
 MenuButtonCommands (class in aiogram.types.menu\_button\_commands), 159  
 MenuButtonDefault (class in aiogram.types.menu\_button\_default), 160  
 MenuButtonType (class in aiogram.enums.menu\_button\_type), 385  
 MenuButtonWebApp (class in aiogram.types.menu\_button\_web\_app), 160  
 MESSAGE (aiogram.enums.update\_type.UpdateType attribute), 388  
 message (aiogram.handlers.callback\_query.CallbackQueryHandler attribute), 440  
 message (aiogram.types.callback\_query.CallbackQuery attribute), 70  
 message (aiogram.types.passport\_element\_error\_data\_field.PassportElementErrorDataField attribute), 218  
 message (aiogram.types.passport\_element\_error\_file.PassportElementErrorFile attribute), 219  
 message (aiogram.types.passport\_element\_error\_files.PassportElementErrorFiles attribute), 220  
 message (aiogram.types.passport\_element\_error\_front\_side.PassportElementErrorFrontSide attribute), 220  
 message (aiogram.types.passport\_element\_error\_reverse\_side.PassportElementErrorReverseSide attribute), 221  
 message (aiogram.types.passport\_element\_error\_selfie.PassportElementErrorSelfie attribute), 222  
 message (aiogram.types.passport\_element\_error\_translation\_file.PassportElementErrorTranslationFile attribute), 223  
 message (aiogram.types.passport\_element\_error\_translation\_files.PassportElementErrorTranslationFiles attribute), 223  
 message (aiogram.types.passport\_element\_error\_unspecified.PassportElementErrorUnspecified attribute), 224  
 message (aiogram.types.update.Update attribute), 225  
 Message (class in aiogram.types.message), 160  
 message\_auto\_delete\_time (aiogram.types.chat.Chat attribute), 73  
 message\_auto\_delete\_time (aiogram.types.message\_auto\_delete\_timer\_changed.MessageAutoDeleteTimerChanged attribute), 203  
 MESSAGE\_AUTO\_DELETE\_TIMER\_CHANGED (aiogram.enums.content\_type.ContentType attribute), 379  
 message\_auto\_delete\_timer\_changed (aiogram.types.message.Message attribute), 164  
 message\_id (aiogram.methods.copy\_message.CopyMessage attribute), 244  
 message\_id (aiogram.methods.delete\_message.DeleteMessage attribute), 362  
 message\_id (aiogram.methods.edit\_message\_caption.EditMessageCaption attribute), 363  
 message\_id (aiogram.methods.edit\_message\_live\_location.EditMessageLiveLocation attribute), 364  
 message\_id (aiogram.methods.edit\_message\_media.EditMessageMedia attribute), 366  
 message\_id (aiogram.methods.edit\_message\_reply\_markup.EditMessageReplyMarkup attribute), 367  
 message\_id (aiogram.methods.edit\_message\_text.EditMessageText attribute), 369  
 message\_id (aiogram.methods.forward\_message.ForwardMessage attribute), 257  
 message\_id (aiogram.methods.get\_game\_high\_scores.GetGameHighScores attribute), 352  
 message\_id (aiogram.methods.pin\_chat\_message.PinChatMessage attribute), 273  
 message\_id (aiogram.methods.set\_game\_score.SetGameScore attribute), 355  
 message\_id (aiogram.methods.stop\_message\_live\_location.StopMessageLiveLocation attribute), 371  
 message\_id (aiogram.methods.stop\_poll.StopPoll attribute), 371  
 message\_id (aiogram.methods.unpin\_chat\_message.UnpinChatMessage attribute), 371  
 message\_id (aiogram.types.message.Message attribute), 160

[message\\_id \(aiogram.types.message\\_id.MessageId attribute\), 204](#)  
[message\\_text \(aiogram.types.input\\_text\\_message\\_content.InputTextMessageContent attribute\), 62](#)  
[message\\_thread\\_id \(aiogram.methods.close\\_forum\\_topic.CloseForumTopic attribute\), 242](#)  
[message\\_thread\\_id \(aiogram.methods.copy\\_message.CopyMessage attribute\), 244](#)  
[message\\_thread\\_id \(aiogram.methods.delete\\_forum\\_topic.DeleteForumTopic attribute\), 251](#)  
[message\\_thread\\_id \(aiogram.methods.edit\\_forum\\_topic.EditForumTopic attribute\), 254](#)  
[message\\_thread\\_id \(aiogram.methods.forward\\_message.ForwardMessage attribute\), 258](#)  
[message\\_thread\\_id \(aiogram.methods.reopen\\_forum\\_topic.ReopenForumTopic attribute\), 276](#)  
[message\\_thread\\_id \(aiogram.methods.send\\_animation.SendAnimation attribute\), 281](#)  
[message\\_thread\\_id \(aiogram.methods.send\\_audio.SendAudio attribute\), 283](#)  
[message\\_thread\\_id \(aiogram.methods.send\\_chat\\_action.SendChatAction attribute\), 285](#)  
[message\\_thread\\_id \(aiogram.methods.send\\_contact.SendContact attribute\), 286](#)  
[message\\_thread\\_id \(aiogram.methods.send\\_dice.SendDice attribute\), 288](#)  
[message\\_thread\\_id \(aiogram.methods.send\\_document.SendDocument attribute\), 290](#)  
[message\\_thread\\_id \(aiogram.methods.send\\_game.SendGame attribute\), 353](#)  
[message\\_thread\\_id \(aiogram.methods.send\\_invoice.SendInvoice attribute\), 335](#)  
[message\\_thread\\_id \(aiogram.methods.send\\_location.SendLocation attribute\), 292](#)  
[message\\_thread\\_id \(aiogram.methods.send\\_media\\_group.SendMediaGroup attribute\), 294](#)  
[message\\_thread\\_id \(aiogram.methods.send\\_message.SendMessage attribute\), 295](#)  
[message\\_thread\\_id \(aiogram.methods.send\\_photo.SendPhoto attribute\), 297](#)  
[message\\_thread\\_id \(aiogram.methods.send\\_poll.SendPoll attribute\), 299](#)  
[message\\_thread\\_id \(aiogram.methods.send\\_sticker.SendSticker attribute\), 343](#)  
[message\\_thread\\_id \(aiogram.methods.send\\_venue.SendVenue attribute\), 302](#)  
[message\\_thread\\_id \(aiogram.methods.send\\_video.SendVideo attribute\), 303](#)  
[message\\_thread\\_id \(aiogram.methods.send\\_video\\_note.SendVideoNote attribute\), 306](#)  
[message\\_thread\\_id \(aiogram.methods.send\\_voice.SendVoice attribute\), 308](#)  
[message\\_thread\\_id \(aiogram.methods.unpin\\_all\\_forum\\_topic.UnpinAllForumTopic attribute\), 326](#)  
[message\\_thread\\_id \(aiogram.types.forum\\_topic.ForumTopic attribute\), 145](#)  
[message\\_thread\\_id \(aiogram.types.message.Message attribute\), 162](#)  
[MessageAutoDeleteTimerChanged \(class in aiogram.types.message\\_auto\\_delete\\_timer\\_changed\), 203](#)  
[MessageEntity \(class in aiogram.types.message\\_entity\), 204](#)  
[MessageEntityType \(class in aiogram.enums.message\\_entity\\_type\), 385](#)  
[MessageId \(class in aiogram.types.message\\_id\), 204](#)  
[MIGRATE\\_FROM\\_CHAT\\_ID \(aiogram.enums.content\\_type.ContentType attribute\), 379](#)  
[migrate\\_from\\_chat\\_id \(aiogram.types.message.Message attribute\), 164](#)  
[MIGRATE\\_TO\\_CHAT\\_ID \(aiogram.enums.content\\_type.ContentType attribute\), 379](#)  
[single\\_chat\\_action\\_chat\\_id \(aiogram.types.message.Message attribute\), 164](#)  
[migrate\\_to\\_chat\\_id \(aiogram.types.response\\_parameters.ResponseParameters attribute\), 208](#)  
[mime\\_type \(aiogram.types.animation.Animation attribute\), 64](#)  
[mime\\_type \(aiogram.types.audio.Audio attribute\), 65](#)  
[mime\\_type \(aiogram.types.document.Document attribute\), 144](#)  
[mime\\_type \(aiogram.types.inline\\_query\\_result\\_document.InlineQueryResultDocument attribute\), 43](#)  
[mime\\_type \(aiogram.types.inline\\_query\\_result\\_video.InlineQueryResultVideo attribute\), 54](#)  
[mime\\_type \(aiogram.types.video.Video attribute\), 213](#)  
[mime\\_type \(aiogram.types.voice.Voice attribute\), 214](#)  
[MNT \(aiogram.enums.currency.Currency attribute\), 381](#)  
[model\\_config \(aiogram.utils.web\\_app.WebAppInitData attribute\), 456](#)  
[model\\_config \(aiogram.utils.web\\_app.WebAppUser attribute\), 457](#)  
[model\\_fields \(aiogram.utils.web\\_app.WebAppInitData attribute\), 456](#)  
[model\\_fields \(aiogram.utils.web\\_app.WebAppUser attribute\), 457](#)  
[module \(aiogram.dispatcher.flags\), 438](#)  
[aiogram.enums.bot\\_command\\_scope\\_type, 376](#)  
[aiogram.enums.chat\\_action, 377](#)  
[aiogram.enums.chat\\_member\\_status, 377](#)  
[aiogram.enums.chat\\_type, 378](#)  
[aiogram.enums.content\\_type, 378](#)  
[aiogram.enums.currency, 380](#)  
[aiogram.enums.all\\_forum\\_topic\\_messages, 383](#)  
[aiogram.enums.encrypted\\_passport\\_element, 383](#)

- 383
- aiogram.enums.inline\_query\_result\_type, 384
- aiogram.enums.input\_media\_type, 384
- aiogram.enums.mask\_position\_point, 385
- aiogram.enums.menu\_button\_type, 385
- aiogram.enums.message\_entity\_type, 385
- aiogram.enums.parse\_mode, 386
- aiogram.enums.passport\_element\_error\_type, 386
- aiogram.enums.poll\_type, 387
- aiogram.enums.sticker\_format, 387
- aiogram.enums.sticker\_type, 387
- aiogram.enums.topic\_icon\_color, 387
- aiogram.enums.update\_type, 388
- aiogram.exceptions, 437
- aiogram.handlers.callback\_query, 440
- aiogram.methods.add\_sticker\_to\_set, 337
- aiogram.methods.answer\_callback\_query, 236
- aiogram.methods.answer\_inline\_query, 372
- aiogram.methods.answer\_pre\_checkout\_query, 329
- aiogram.methods.answer\_shipping\_query, 330
- aiogram.methods.answer\_web\_app\_query, 375
- aiogram.methods.approve\_chat\_join\_request, 237
- aiogram.methods.ban\_chat\_member, 238
- aiogram.methods.ban\_chat\_sender\_chat, 240
- aiogram.methods.close, 241
- aiogram.methods.close\_forum\_topic, 242
- aiogram.methods.close\_general\_forum\_topic, 243
- aiogram.methods.copy\_message, 243
- aiogram.methods.create\_chat\_invite\_link, 245
- aiogram.methods.create\_forum\_topic, 247
- aiogram.methods.create\_invoice\_link, 331
- aiogram.methods.create\_new\_sticker\_set, 338
- aiogram.methods.decline\_chat\_join\_request, 248
- aiogram.methods.delete\_chat\_photo, 249
- aiogram.methods.delete\_chat\_sticker\_set, 250
- aiogram.methods.delete\_forum\_topic, 251
- aiogram.methods.delete\_message, 361
- aiogram.methods.delete\_my\_commands, 252
- aiogram.methods.delete\_sticker\_from\_set, 339
- aiogram.methods.delete\_sticker\_set, 340
- aiogram.methods.delete\_webhook, 356
- aiogram.methods.edit\_chat\_invite\_link, 253
- aiogram.methods.edit\_forum\_topic, 254
- aiogram.methods.edit\_general\_forum\_topic, 255
- aiogram.methods.edit\_message\_caption, 362
- aiogram.methods.edit\_message\_live\_location, 364
- aiogram.methods.edit\_message\_media, 366
- aiogram.methods.edit\_message\_reply\_markup, 367
- aiogram.methods.edit\_message\_text, 368
- aiogram.methods.export\_chat\_invite\_link, 256
- aiogram.methods.forward\_message, 257
- aiogram.methods.get\_chat, 259
- aiogram.methods.get\_chat\_administrators, 259
- aiogram.methods.get\_chat\_member, 260
- aiogram.methods.get\_chat\_member\_count, 261
- aiogram.methods.get\_chat\_menu\_button, 262
- aiogram.methods.get\_custom\_emoji\_stickers, 341
- aiogram.methods.get\_file, 263
- aiogram.methods.get\_forum\_topic\_icon\_stickers, 264
- aiogram.methods.get\_game\_high\_scores, 352
- aiogram.methods.get\_me, 264
- aiogram.methods.get\_my\_commands, 265
- aiogram.methods.get\_my\_default\_administrator\_rights, 266
- aiogram.methods.get\_my\_description, 267
- aiogram.methods.get\_my\_name, 268
- aiogram.methods.get\_my\_short\_description, 268
- aiogram.methods.get\_sticker\_set, 342
- aiogram.methods.get\_updates, 356
- aiogram.methods.get\_user\_profile\_photos, 269
- aiogram.methods.get\_webhook\_info, 358
- aiogram.methods.hide\_general\_forum\_topic, 270
- aiogram.methods.leave\_chat, 271
- aiogram.methods.log\_out, 272
- aiogram.methods.pin\_chat\_message, 273
- aiogram.methods.promote\_chat\_member, 274
- aiogram.methods.reopen\_forum\_topic, 276
- aiogram.methods.reopen\_general\_forum\_topic, 277
- aiogram.methods.restrict\_chat\_member, 278
- aiogram.methods.revoke\_chat\_invite\_link, 279
- aiogram.methods.send\_animation, 280

- aiogram.methods.send\_audio, 282
- aiogram.methods.send\_chat\_action, 285
- aiogram.methods.send\_contact, 286
- aiogram.methods.send\_dice, 288
- aiogram.methods.send\_document, 289
- aiogram.methods.send\_game, 353
- aiogram.methods.send\_invoice, 334
- aiogram.methods.send\_location, 291
- aiogram.methods.send\_media\_group, 293
- aiogram.methods.send\_message, 295
- aiogram.methods.send\_photo, 297
- aiogram.methods.send\_poll, 299
- aiogram.methods.send\_sticker, 342
- aiogram.methods.send\_venue, 301
- aiogram.methods.send\_video, 303
- aiogram.methods.send\_video\_note, 305
- aiogram.methods.send\_voice, 307
- aiogram.methods.set\_chat\_administrator\_custom\_title, 309
- aiogram.methods.set\_chat\_description, 310
- aiogram.methods.set\_chat\_menu\_button, 311
- aiogram.methods.set\_chat\_permissions, 312
- aiogram.methods.set\_chat\_photo, 313
- aiogram.methods.set\_chat\_sticker\_set, 314
- aiogram.methods.set\_chat\_title, 315
- aiogram.methods.set\_custom\_emoji\_sticker\_set\_thumbnail, 344
- aiogram.methods.set\_game\_score, 354
- aiogram.methods.set\_my\_commands, 316
- aiogram.methods.set\_my\_default\_administrator\_rights, 317
- aiogram.methods.set\_my\_description, 319
- aiogram.methods.set\_my\_name, 320
- aiogram.methods.set\_my\_short\_description, 321
- aiogram.methods.set\_passport\_data\_errors, 360
- aiogram.methods.set\_sticker\_emoji\_list, 345
- aiogram.methods.set\_sticker\_keywords, 346
- aiogram.methods.set\_sticker\_mask\_position, 347
- aiogram.methods.set\_sticker\_position\_in\_set, 348
- aiogram.methods.set\_sticker\_set\_thumbnail, 349
- aiogram.methods.set\_sticker\_set\_title, 350
- aiogram.methods.set\_webhook, 358
- aiogram.methods.stop\_message\_live\_location, 370
- aiogram.methods.stop\_poll, 371
- aiogram.methods.unban\_chat\_member, 322
- aiogram.methods.unban\_chat\_sender\_chat, 323
- aiogram.methods.unhide\_general\_forum\_topic, 324
- aiogram.methods.unpin\_all\_chat\_messages, 325
- aiogram.methods.unpin\_all\_forum\_topic\_messages, 326
- aiogram.methods.unpin\_all\_general\_forum\_topic\_messages, 327
- aiogram.methods.unpin\_chat\_message, 328
- aiogram.methods.upload\_sticker\_file, 351
- aiogram.types.animation, 64
- aiogram.types.audio, 64
- aiogram.types.bot\_command, 65
- aiogram.types.bot\_command\_scope, 65
- aiogram.types.bot\_command\_scope\_all\_chat\_administrators, 66
- aiogram.types.bot\_command\_scope\_all\_group\_chats, 66
- aiogram.types.bot\_command\_scope\_all\_private\_chats, 67
- aiogram.types.bot\_command\_scope\_chat, 67
- aiogram.types.bot\_command\_scope\_chat\_administrators, 67
- aiogram.types.bot\_command\_scope\_chat\_member, 68
- aiogram.types.bot\_command\_scope\_default, 69
- aiogram.types.bot\_description, 69
- aiogram.types.bot\_name, 69
- aiogram.types.bot\_short\_description, 69
- aiogram.types.callback\_game, 235
- aiogram.types.callback\_query, 70
- aiogram.types.chat, 71
- aiogram.types.chat\_administrator\_rights, 84
- aiogram.types.chat\_invite\_link, 85
- aiogram.types.chat\_join\_request, 86
- aiogram.types.chat\_location, 119
- aiogram.types.chat\_member, 119
- aiogram.types.chat\_member\_administrator, 120
- aiogram.types.chat\_member\_banned, 121
- aiogram.types.chat\_member\_left, 122
- aiogram.types.chat\_member\_member, 122
- aiogram.types.chat\_member\_owner, 122
- aiogram.types.chat\_member\_restricted, 123
- aiogram.types.chat\_member\_updated, 124
- aiogram.types.chat\_permissions, 141
- aiogram.types.chat\_photo, 142
- aiogram.types.chat\_shared, 142
- aiogram.types.chosen\_inline\_result, 17
- aiogram.types.contact, 143

aiogram.types.dice, 143  
 aiogram.types.document, 144  
 aiogram.types.encrypted\_credentials, 215  
 aiogram.types.encrypted\_passport\_element, 216  
 aiogram.types.error\_event, 436  
 aiogram.types.file, 144  
 aiogram.types.force\_reply, 145  
 aiogram.types.forum\_topic, 145  
 aiogram.types.forum\_topic\_closed, 146  
 aiogram.types.forum\_topic\_created, 146  
 aiogram.types.forum\_topic\_edited, 146  
 aiogram.types.forum\_topic\_reopened, 147  
 aiogram.types.game, 235  
 aiogram.types.game\_high\_score, 236  
 aiogram.types.general\_forum\_topic\_hidden, 147  
 aiogram.types.general\_forum\_topic\_unhidden, 147  
 aiogram.types.inline\_keyboard\_button, 147  
 aiogram.types.inline\_keyboard\_markup, 148  
 aiogram.types.inline\_query, 18  
 aiogram.types.inline\_query\_result, 19  
 aiogram.types.inline\_query\_result\_article, 20  
 aiogram.types.inline\_query\_result\_audio, 21  
 aiogram.types.inline\_query\_result\_cached\_audio, 23  
 aiogram.types.inline\_query\_result\_cached\_document, 25  
 aiogram.types.inline\_query\_result\_cached\_gif, 27  
 aiogram.types.inline\_query\_result\_cached\_mpeg4\_gif, 29  
 aiogram.types.inline\_query\_result\_cached\_photo, 32  
 aiogram.types.inline\_query\_result\_cached\_sticker, 34  
 aiogram.types.inline\_query\_result\_cached\_video, 35  
 aiogram.types.inline\_query\_result\_cached\_voice, 37  
 aiogram.types.inline\_query\_result\_contact, 39  
 aiogram.types.inline\_query\_result\_document, 41  
 aiogram.types.inline\_query\_result\_game, 43  
 aiogram.types.inline\_query\_result\_gif, 44  
 aiogram.types.inline\_query\_result\_location, 45  
 aiogram.types.inline\_query\_result\_mpeg4\_gif, 48  
 aiogram.types.inline\_query\_result\_photo, 50  
 aiogram.types.inline\_query\_result\_venue, 51  
 aiogram.types.inline\_query\_result\_video, 53  
 aiogram.types.inline\_query\_result\_voice, 56  
 aiogram.types.inline\_query\_results\_button, 57  
 aiogram.types.input\_contact\_message\_content, 57  
 aiogram.types.input\_file, 149  
 aiogram.types.input\_invoice\_message\_content, 58  
 aiogram.types.input\_location\_message\_content, 61  
 aiogram.types.input\_media, 149  
 aiogram.types.input\_media\_animation, 150  
 aiogram.types.input\_media\_audio, 151  
 aiogram.types.input\_media\_document, 152  
 aiogram.types.input\_media\_photo, 153  
 aiogram.types.input\_media\_video, 153  
 aiogram.types.input\_message\_content, 62  
 aiogram.types.input\_sticker, 227  
 aiogram.types.input\_text\_message\_content, 62  
 aiogram.types.input\_venue\_message\_content, 63  
 aiogram.types.invoice, 230  
 aiogram.types.keyboard\_button, 155  
 aiogram.types.keyboard\_button\_poll\_type, 156  
 aiogram.types.keyboard\_button\_request\_chat, 156  
 aiogram.types.keyboard\_button\_request\_user, 157  
 aiogram.types.labeled\_price, 231  
 aiogram.types.location, 158  
 aiogram.types.login\_url, 158  
 aiogram.types.mask\_position, 228  
 aiogram.types.menu\_button, 159  
 aiogram.types.menu\_button\_commands, 159  
 aiogram.types.menu\_button\_default, 160  
 aiogram.types.menu\_button\_web\_app, 160  
 aiogram.types.message, 160  
 aiogram.types.message\_auto\_delete\_timer\_changed, 203  
 aiogram.types.message\_entity, 204  
 aiogram.types.message\_id, 204  
 aiogram.types.order\_info, 231  
 aiogram.types.passport\_data, 217  
 aiogram.types.passport\_element\_error, 217



[aiogram.types.passport\\_element\\_error\\_data\\_field\\_id](#), 218  
[aiogram.types.passport\\_element\\_error\\_file](#), 219  
[aiogram.types.passport\\_element\\_error\\_files](#), 219  
[aiogram.types.passport\\_element\\_error\\_front\\_side](#), 220  
[aiogram.types.passport\\_element\\_error\\_reverse\\_side](#), 221  
[aiogram.types.passport\\_element\\_error\\_selfie](#), 221  
[aiogram.types.passport\\_element\\_error\\_transformation\\_file](#), 222  
[aiogram.types.passport\\_element\\_error\\_transformation\\_files](#), 223  
[aiogram.types.passport\\_element\\_error\\_unspecified\\_file](#), 224  
[aiogram.types.passport\\_file](#), 224  
[aiogram.types.photo\\_size](#), 205  
[aiogram.types.poll](#), 205  
[aiogram.types.poll\\_answer](#), 206  
[aiogram.types.poll\\_option](#), 206  
[aiogram.types.pre\\_checkout\\_query](#), 231  
[aiogram.types.proximity\\_alert\\_triggered](#), 207  
[aiogram.types.reply\\_keyboard\\_markup](#), 207  
[aiogram.types.reply\\_keyboard\\_remove](#), 208  
[aiogram.types.response\\_parameters](#), 208  
[aiogram.types.sent\\_web\\_app\\_message](#), 63  
[aiogram.types.shipping\\_address](#), 232  
[aiogram.types.shipping\\_option](#), 233  
[aiogram.types.shipping\\_query](#), 233  
[aiogram.types.sticker](#), 228  
[aiogram.types.sticker\\_set](#), 230  
[aiogram.types.story](#), 209  
[aiogram.types.successful\\_payment](#), 234  
[aiogram.types.switch\\_inline\\_query\\_chosen\\_chat](#), 209  
[aiogram.types.update](#), 225  
[aiogram.types.user](#), 210  
[aiogram.types.user\\_profile\\_photos](#), 211  
[aiogram.types.user\\_shared](#), 211  
[aiogram.types.venue](#), 212  
[aiogram.types.video](#), 212  
[aiogram.types.video\\_chat\\_ended](#), 213  
[aiogram.types.video\\_chat\\_participants\\_invited](#), 213  
[aiogram.types.video\\_chat\\_scheduled](#), 213  
[aiogram.types.video\\_chat\\_started](#), 214  
[aiogram.types.video\\_note](#), 214  
[aiogram.types.voice](#), 214  
[aiogram.types.web\\_app\\_data](#), 215  
[aiogram.types.web\\_app\\_info](#), 215  
[aiogram.types.webhook\\_info](#), 226  
[aiogram.types.write\\_access\\_allowed](#), 215  
[aiogram.enums.mask\\_position\\_point.MaskPositionPoint](#) attribute), 385  
[aiogram.types.inline\\_query\\_result\\_mpeg4\\_gif.InlineQueryResultMpeg4Gif](#) attribute), 49  
[aiogram.types.inline\\_query\\_result\\_cached\\_mpeg4\\_gif.InlineQueryResultCachedMpeg4Gif](#) attribute), 31  
[aiogram.enums.inline\\_query\\_result\\_type.InlineQueryResultType](#) attribute), 384  
[aiogram.types.inline\\_query\\_result\\_mpeg4\\_gif.InlineQueryResultMpeg4Gif](#) attribute), 49  
[aiogram.types.inline\\_query\\_result\\_mpeg4\\_gif.InlineQueryResultMpeg4Gif](#) attribute), 49  
[aiogram.types.inline\\_query\\_result\\_mpeg4\\_gif.InlineQueryResultMpeg4Gif](#) attribute), 49  
[aiogram.enums.currency.Currency](#) attribute), 381  
[aiogram.enums.currency.Currency](#) attribute), 381  
[aiogram.enums.currency.Currency](#) attribute), 381  
[aiogram.enums.update\\_type.UpdateType](#) attribute), 388  
[aiogram.types.update.Update](#) attribute), 226  
[aiogram.enums.currency.Currency](#) attribute), 381  
[aiogram.enums.currency.Currency](#) attribute), 381

## N

[aiogram.methods.add\\_sticker\\_to\\_set.AddStickerToSet](#) attribute), 337  
[aiogram.methods.create\\_chat\\_invite\\_link.CreateChatInviteLink](#) attribute), 246  
[aiogram.methods.create\\_forum\\_topic.CreateForumTopic](#) attribute), 247  
[aiogram.methods.create\\_new\\_sticker\\_set.CreateNewStickerSet](#) attribute), 338  
[aiogram.methods.delete\\_sticker\\_set.DeleteStickerSet](#) attribute), 340  
[aiogram.methods.edit\\_chat\\_invite\\_link.EditChatInviteLink](#) attribute), 253  
[aiogram.methods.edit\\_forum\\_topic.EditForumTopic](#) attribute), 254  
[aiogram.methods.edit\\_general\\_forum\\_topic.EditGeneralForumTopic](#) attribute), 255  
[aiogram.methods.get\\_sticker\\_set.GetStickerSet](#) attribute), 342  
[aiogram.methods.set\\_custom\\_emoji\\_sticker\\_set\\_thumbnail.SetCustomEmojiStickerSetThumbnail](#) attribute), 344  
[aiogram.methods.set\\_my\\_name.SetMyName](#) attribute), 320  
[aiogram.methods.set\\_sticker\\_set\\_thumbnail.SetStickerSetThumbnail](#) attribute), 349  
[aiogram.methods.set\\_sticker\\_set\\_title.SetStickerSetTitle](#) attribute), 350  
[aiogram.types.bot\\_name.BotName](#) attribute), 69

[name \(aiogram.types.chat\\_invite\\_link.ChatInviteLink attribute\), 86](#)  
[name \(aiogram.types.forum\\_topic.ForumTopic attribute\), 146](#)  
[name \(aiogram.types.forum\\_topic\\_created.ForumTopicCreated attribute\), 146](#)  
[name \(aiogram.types.forum\\_topic\\_edited.ForumTopicEdited attribute\), 146](#)  
[name \(aiogram.types.order\\_info.OrderInfo attribute\), 231](#)  
[name \(aiogram.types.sticker\\_set.StickerSet attribute\), 230](#)  
[need\\_email \(aiogram.methods.create\\_invoice\\_link.CreateInvoiceLink attribute\), 333](#)  
[need\\_email \(aiogram.methods.send\\_invoice.SendInvoice attribute\), 335](#)  
[need\\_email \(aiogram.types.input\\_invoice\\_message\\_content.InputInvoiceMessageContent attribute\), 60](#)  
[need\\_name \(aiogram.methods.create\\_invoice\\_link.CreateInvoiceLink attribute\), 333](#)  
[need\\_name \(aiogram.methods.send\\_invoice.SendInvoice attribute\), 335](#)  
[need\\_name \(aiogram.types.input\\_invoice\\_message\\_content.InputInvoiceMessageContent attribute\), 60](#)  
[need\\_phone\\_number \(aiogram.methods.create\\_invoice\\_link.CreateInvoiceLink attribute\), 333](#)  
[need\\_phone\\_number \(aiogram.methods.send\\_invoice.SendInvoice attribute\), 335](#)  
[need\\_phone\\_number \(aiogram.types.input\\_invoice\\_message\\_content.InputInvoiceMessageContent attribute\), 60](#)  
[need\\_shipping\\_address \(aiogram.methods.create\\_invoice\\_link.CreateInvoiceLink attribute\), 333](#)  
[need\\_shipping\\_address \(aiogram.methods.send\\_invoice.SendInvoice attribute\), 335](#)  
[need\\_shipping\\_address \(aiogram.types.input\\_invoice\\_message\\_content.InputInvoiceMessageContent attribute\), 60](#)  
[needs\\_repainting \(aiogram.methods.create\\_new\\_sticker\\_set.CreateNewStickerSet attribute\), 338](#)  
[needs\\_repainting \(aiogram.types.sticker.Sticker attribute\), 229](#)  
[new\\_chat\\_member \(aiogram.types.chat\\_member\\_updated.ChatMemberUpdated attribute\), 125](#)  
[NEW\\_CHAT\\_MEMBERS \(aiogram.enums.content\\_type.ContentType attribute\), 379](#)  
[new\\_chat\\_members \(aiogram.types.message.Message attribute\), 164](#)  
[NEW\\_CHAT\\_PHOTO \(aiogram.enums.content\\_type.ContentType attribute\), 379](#)  
[new\\_chat\\_photo \(aiogram.types.message.Message attribute\), 164](#)  
[NEW\\_CHAT\\_TITLE \(aiogram.enums.content\\_type.ContentType attribute\), 379](#)  
[new\\_chat\\_title \(aiogram.types.message.Message attribute\), 164](#)  
[next\\_offset \(aiogram.methods.answer\\_inline\\_query.AnswerInlineQuery attribute\), 373](#)  
[NGN \(aiogram.enums.currency.Currency attribute\), 381](#)  
[NIO \(aiogram.enums.currency.Currency attribute\), 382](#)  
[NOK \(aiogram.enums.currency.Currency attribute\), 382](#)  
[NPR \(aiogram.enums.currency.Currency attribute\), 382](#)  
[NZD \(aiogram.enums.currency.Currency attribute\), 382](#)  

## O

[offset \(aiogram.methods.get\\_updates.GetUpdates attribute\), 357](#)  
[offset \(aiogram.methods.get\\_user\\_profile\\_photos.GetUserProfilePhotos attribute\), 269](#)  
[offset \(aiogram.types.inline\\_query.InlineQuery attribute\), 18](#)  
[offset \(aiogram.types.message\\_entity.MessageEntity attribute\), 204](#)  
[ok \(aiogram.methods.answer\\_pre\\_checkout\\_query.AnswerPreCheckoutQuery attribute\), 329](#)  
[ok \(aiogram.methods.answer\\_shipping\\_query.AnswerShippingQuery attribute\), 330](#)  
[old\\_chat\\_member \(aiogram.types.chat\\_member\\_updated.ChatMemberUpdated attribute\), 125](#)  
[one\\_time\\_keyboard \(aiogram.types.reply\\_keyboard\\_markup.ReplyKeyboardMarkup attribute\), 207](#)  
[only\\_if\\_banned \(aiogram.methods.unban\\_chat\\_member.UnbanChatMember attribute\), 322](#)  
[open\\_period \(aiogram.methods.send\\_poll.SendPoll attribute\), 300](#)  
[open\\_period \(aiogram.types.poll.Poll attribute\), 206](#)  
[option\\_ids \(aiogram.types.poll\\_answer.PollAnswer attribute\), 206](#)  
[options \(aiogram.methods.send\\_poll.SendPoll attribute\), 299](#)  
[options \(aiogram.types.poll.Poll attribute\), 205](#)  
[order\\_info \(aiogram.types.pre\\_checkout\\_query.PreCheckoutQuery attribute\), 235](#)  
[order\\_info \(aiogram.types.successful\\_payment.SuccessfulPayment attribute\), 235](#)  
[OrderInfo \(class in aiogram.types.order\\_info\), 231](#)  

## P

[PAB \(aiogram.enums.currency.Currency attribute\), 382](#)  
[pack\(\) \(aiogram.filters.callback\\_data.CallbackData method\), 409](#)  
[parse\\_mode \(aiogram.methods.copy\\_message.CopyMessage attribute\), 244](#)  
[parse\\_mode \(aiogram.methods.edit\\_message\\_caption.EditMessageCaption attribute\), 363](#)  
[parse\\_mode \(aiogram.methods.edit\\_message\\_text.EditMessageText attribute\), 369](#)  
[parse\\_mode \(aiogram.methods.send\\_animation.SendAnimation attribute\), 281](#)

`parse_mode` (`aiogram.methods.send_audio.SendAudio` attribute), 283  
`parse_mode` (`aiogram.methods.send_document.SendDocument` attribute), 290  
`parse_mode` (`aiogram.methods.send_message.SendMessage` attribute), 295  
`parse_mode` (`aiogram.methods.send_photo.SendPhoto` attribute), 297  
`parse_mode` (`aiogram.methods.send_video.SendVideo` attribute), 304  
`parse_mode` (`aiogram.methods.send_voice.SendVoice` attribute), 308  
`parse_mode` (`aiogram.types.inline_query_result_audio.InlineQueryResultCachedAudio` attribute), 22  
`parse_mode` (`aiogram.types.inline_query_result_cached_audio.InlineQueryResultCachedAudio` attribute), 25  
`parse_mode` (`aiogram.types.inline_query_result_cached_document.InlineQueryResultCachedDocument` attribute), 27  
`parse_mode` (`aiogram.types.inline_query_result_cached_gif.InlineQueryResultCachedGif` attribute), 29  
`parse_mode` (`aiogram.types.inline_query_result_cached_mpeg4_gif.InlineQueryResultCachedMpeg4Gif` attribute), 31  
`parse_mode` (`aiogram.types.inline_query_result_cached_photo.InlineQueryResultCachedPhoto` attribute), 33  
`parse_mode` (`aiogram.types.inline_query_result_cached_video.InlineQueryResultCachedVideo` attribute), 37  
`parse_mode` (`aiogram.types.inline_query_result_cached_voice.InlineQueryResultCachedVoice` attribute), 39  
`parse_mode` (`aiogram.types.inline_query_result_document.InlineQueryResultDocument` attribute), 43  
`parse_mode` (`aiogram.types.inline_query_result_gif.InlineQueryResultGif` attribute), 45  
`parse_mode` (`aiogram.types.inline_query_result_mpeg4_gif.InlineQueryResultMpeg4Gif` attribute), 49  
`parse_mode` (`aiogram.types.inline_query_result_photo.InlineQueryResultPhoto` attribute), 51  
`parse_mode` (`aiogram.types.inline_query_result_video.InlineQueryResultVideo` attribute), 55  
`parse_mode` (`aiogram.types.inline_query_result_voice.InlineQueryResultVoice` attribute), 56  
`parse_mode` (`aiogram.types.input_media_animation.InputMediaAnimation` attribute), 150  
`parse_mode` (`aiogram.types.input_media_audio.InputMediaAudio` attribute), 151  
`parse_mode` (`aiogram.types.input_media_document.InputMediaDocument` attribute), 152  
`parse_mode` (`aiogram.types.input_media_photo.InputMediaPhoto` attribute), 153  
`parse_mode` (`aiogram.types.input_media_video.InputMediaVideo` attribute), 154  
`parse_mode` (`aiogram.types.input_text_message_content.InputTextMessageContent` attribute), 62  
`parse_webapp_init_data()` (in module `aiogram.utils.web_app`), 455  
`ParseMode` (class in `aiogram.enums.parse_mode`), 386  
`PASSPORT` (`aiogram.enums.encrypted_passport_element.EncryptedPassportElement` attribute), 383  
`PASSPORT_DATA` (`aiogram.enums.content_type.ContentType` attribute), 379  
`passport_data` (`aiogram.types.message.Message` attribute), 165  
`PASSPORT_REGISTRATION` (`aiogram.enums.encrypted_passport_element.EncryptedPassportElement` attribute), 383  
`PassportData` (class in `aiogram.types.passport_data`), 217  
`PassportElementError` (class in `aiogram.types.passport_element_error`), 217  
`PassportElementErrorDataField` (class in `aiogram.types.passport_element_error_data_field`), 218  
`PassportElementErrorFile` (class in `aiogram.types.passport_element_error_file`), 219  
`PassportElementErrorFiles` (class in `aiogram.types.passport_element_error_files`), 219  
`PassportElementErrorFrontSide` (class in `aiogram.types.passport_element_error_front_side`), 220  
`PassportElementErrorReverseSide` (class in `aiogram.types.passport_element_error_reverse_side`), 221  
`PassportElementErrorSelfie` (class in `aiogram.types.passport_element_error_selfie`), 221  
`PassportElementErrorTranslationFile` (class in `aiogram.types.passport_element_error_translation_file`), 222  
`PassportElementErrorTranslationFiles` (class in `aiogram.types.passport_element_error_translation_files`), 222  
`PassportFileType` (class in `aiogram.enums.passport_element_error_type`), 386  
`PassportElementErrorUnspecified` (class in `aiogram.types.passport_element_error_unspecified`), 224  
`PassportFile` (class in `aiogram.types.passport_file`), 224  
`pattern` (`aiogram.filters.exception.ExceptionMessageFilter` attribute), 411  
`pay` (`aiogram.types.inline_keyboard_button.InlineKeyboardButton` attribute), 448  
`payload` (`aiogram.methods.create_invoice_link.CreateInvoiceLink` attribute), 332  
`payload` (`aiogram.methods.send_invoice.SendInvoice` attribute), 332



tribute), 334

payload (aiogram.types.input\_invoice\_message\_content.InputInvoiceMessageContent attribute), 60

PEN (aiogram.enums.currency.Currency attribute), 382

pending\_join\_request\_count (aiogram.types.chat\_invite\_link.ChatInviteLink attribute), 86

pending\_update\_count (aiogram.types.webhook\_info.WebhookInfo attribute), 227

performer (aiogram.methods.send\_audio.SendAudio attribute), 283

performer (aiogram.types.audio.Audio attribute), 65

performer (aiogram.types.inline\_query\_result\_audio.InlineQueryResultAudio attribute), 23

performer (aiogram.types.input\_media\_audio.InputMediaAudio attribute), 151

permissions (aiogram.methods.restrict\_chat\_member.RestrictChatMember attribute), 278

permissions (aiogram.methods.set\_chat\_permissions.SetChatPermissions attribute), 312

permissions (aiogram.types.chat.Chat attribute), 73

PERSONAL\_DETAILS (aiogram.enums.encrypted\_passport\_element\_type.EncryptedPassportElement attribute), 383

PHONE\_NUMBER (aiogram.enums.encrypted\_passport\_element\_type.EncryptedPassportElement attribute), 383

PHONE\_NUMBER (aiogram.enums.message\_entity\_type.MessageEntityType attribute), 385

phone\_number (aiogram.methods.send\_contact.SendContact attribute), 286

phone\_number (aiogram.types.contact.Contact attribute), 143

phone\_number (aiogram.types.encrypted\_passport\_element\_type.EncryptedPassportElement attribute), 216

phone\_number (aiogram.types.inline\_query\_result\_contact.InlineQueryResultContact attribute), 40

phone\_number (aiogram.types.input\_contact\_message\_content.InputContactMessageContent attribute), 57

phone\_number (aiogram.types.order\_info.OrderInfo attribute), 231

PhoneNumber (class in aiogram.utils.formatting), 465

PHOTO (aiogram.enums.content\_type.ContentType attribute), 378

PHOTO (aiogram.enums.inline\_query\_result\_type.InlineQueryResultType attribute), 384

PHOTO (aiogram.enums.input\_media\_type.InputMediaType attribute), 384

photo (aiogram.methods.send\_photo.SendPhoto attribute), 297

photo (aiogram.methods.set\_chat\_photo.SetChatPhoto attribute), 314

photo (aiogram.types.chat.Chat attribute), 72

photo (aiogram.types.game.Game attribute), 235

photo (aiogram.types.message.Message attribute), 163

photo\_file\_id (aiogram.types.inline\_query\_result\_cached\_photo.InlineQueryResultCachedPhoto attribute), 333

photo\_height (aiogram.methods.create\_invoice\_link.CreateInvoiceLink attribute), 333

photo\_height (aiogram.methods.send\_invoice.SendInvoice attribute), 335

photo\_height (aiogram.types.inline\_query\_result\_photo.InlineQueryResultPhoto attribute), 50

photo\_height (aiogram.types.input\_invoice\_message\_content.InputInvoiceMessageContent attribute), 60

photo\_size (aiogram.methods.create\_invoice\_link.CreateInvoiceLink attribute), 333

photo\_size (aiogram.methods.send\_invoice.SendInvoice attribute), 335

photo\_size (aiogram.types.input\_invoice\_message\_content.InputInvoiceMessageContent attribute), 60

photo\_url (aiogram.methods.create\_invoice\_link.CreateInvoiceLink attribute), 333

photo\_url (aiogram.methods.send\_invoice.SendInvoice attribute), 335

photo\_url (aiogram.types.inline\_query\_result\_photo.InlineQueryResultPhoto attribute), 50

photo\_url (aiogram.types.input\_invoice\_message\_content.InputInvoiceMessageContent attribute), 60

photo\_url (aiogram.types.input\_media\_audio.InputMediaAudio attribute), 151

photo\_url (aiogram.types.web\_app.WebAppUser attribute), 457

photo\_width (aiogram.methods.create\_invoice\_link.CreateInvoiceLink attribute), 333

photo\_width (aiogram.methods.send\_invoice.SendInvoice attribute), 335

photo\_width (aiogram.types.inline\_query\_result\_photo.InlineQueryResultPhoto attribute), 50

photo\_width (aiogram.types.input\_invoice\_message\_content.InputInvoiceMessageContent attribute), 60

PhotoSize (class in aiogram.types.message.Message), 205

PHP (aiogram.enums.currency.Currency attribute), 382

pin() (aiogram.types.message.Message method), 202

pin\_message() (aiogram.types.chat.Chat method), 79

PinChatMessage (class in aiogram.methods.pin\_chat\_message), 273

PINNED\_MESSAGE (aiogram.enums.content\_type.ContentType attribute), 379

pinned\_message (aiogram.types.chat.Chat attribute), 72

pinned\_message (aiogram.types.message.Message attribute), 165

PKR (aiogram.enums.currency.Currency attribute), 382

PLN (aiogram.enums.currency.Currency attribute), 382

point (aiogram.types.mask\_position.MaskPosition attribute), 228

POLL (aiogram.enums.content\_type.ContentType attribute), 378

POLL (*aiogram.enums.update\_type.UpdateType* attribute), 388

poll (*aiogram.types.message.Message* attribute), 164

poll (*aiogram.types.update.Update* attribute), 226

Poll (class in *aiogram.types.poll*), 205

POLL\_ANSWER (*aiogram.enums.update\_type.UpdateType* attribute), 388

poll\_answer (*aiogram.types.update.Update* attribute), 226

poll\_id (*aiogram.types.poll\_answer.PollAnswer* attribute), 206

PollAnswer (class in *aiogram.types.poll\_answer*), 206

PollOption (class in *aiogram.types.poll\_option*), 206

PollType (class in *aiogram.enums.poll\_type*), 387

position (*aiogram.methods.set\_sticker\_position\_in\_set.SetStickerPositionInSet* attribute), 348

position (*aiogram.types.game\_high\_score.GameHighScore* attribute), 236

post\_code (*aiogram.types.shipping\_address.ShippingAddress* attribute), 233

PRE (*aiogram.enums.message\_entity\_type.MessageEntityType* attribute), 386

Pre (class in *aiogram.utils.formatting*), 466

PRE\_CHECKOUT\_QUERY (*aiogram.enums.update\_type.UpdateType* attribute), 388

pre\_checkout\_query (*aiogram.types.update.Update* attribute), 226

pre\_checkout\_query\_id (*aiogram.methods.answer\_pre\_checkout\_query.AnswerPreCheckoutQuery* attribute), 329

PreCheckoutQuery (class in *aiogram.types.pre\_checkout\_query*), 231

prefix (*aiogram.filters.command.CommandObject* attribute), 403

premium\_animation (*aiogram.types.sticker.Sticker* attribute), 229

prepare\_value() (*aiogram.client.session.base.BaseSession* method), 13

prices (*aiogram.methods.create\_invoice\_link.CreateInvoiceLink* attribute), 332

prices (*aiogram.methods.send\_invoice.SendInvoice* attribute), 335

prices (*aiogram.types.input\_invoice\_message\_content.InputInvoiceMessageContent* attribute), 60

prices (*aiogram.types.shipping\_option.ShippingOption* attribute), 233

PRIVATE (*aiogram.enums.chat\_type.ChatType* attribute), 378

promote() (*aiogram.types.chat.Chat* method), 80

PromoteChatMember (class in *aiogram.methods.promote\_chat\_member*), 274

protect\_content (*aiogram.methods.copy\_message.CopyMessage* attribute), 244

protect\_content (*aiogram.methods.forward\_message.ForwardMessage* attribute), 258

protect\_content (*aiogram.methods.send\_animation.SendAnimation* attribute), 281

protect\_content (*aiogram.methods.send\_audio.SendAudio* attribute), 284

protect\_content (*aiogram.methods.send\_contact.SendContact* attribute), 286

protect\_content (*aiogram.methods.send\_dice.SendDice* attribute), 288

protect\_content (*aiogram.methods.send\_document.SendDocument* attribute), 290

protect\_content (*aiogram.methods.send\_game.SendGame* attribute), 353

protect\_content (*aiogram.methods.send\_invoice.SendInvoice* attribute), 336

protect\_content (*aiogram.methods.send\_location.SendLocation* attribute), 292

protect\_content (*aiogram.methods.send\_media\_group.SendMediaGroup* attribute), 294

protect\_content (*aiogram.methods.send\_message.SendMessage* attribute), 296

protect\_content (*aiogram.methods.send\_photo.SendPhoto* attribute), 298

protect\_content (*aiogram.methods.send\_poll.SendPoll* attribute), 300

protect\_content (*aiogram.methods.send\_sticker.SendSticker* attribute), 343

protect\_content (*aiogram.methods.send\_venue.SendVenue* attribute), 302

protect\_content (*aiogram.methods.send\_video.SendVideo* attribute), 304

protect\_content (*aiogram.methods.send\_video\_note.SendVideoNote* attribute), 306

protect\_content (*aiogram.methods.send\_voice.SendVoice* attribute), 308

provider\_data (*aiogram.methods.create\_invoice\_link.CreateInvoiceLink* attribute), 332

provider\_data (*aiogram.methods.send\_invoice.SendInvoice* attribute), 335

provider\_data (*aiogram.types.input\_invoice\_message\_content.InputInvoiceMessageContent* attribute), 60

provider\_data (*aiogram.methods.create\_invoice\_link.CreateInvoiceLink* attribute), 332

provider\_data (*aiogram.methods.send\_invoice.SendInvoice* attribute), 334

provider\_data (*aiogram.types.input\_invoice\_message\_content.InputInvoiceMessageContent* attribute), 60

proximity\_alert\_radius (*aiogram.methods.edit\_message\_live\_location.EditMessageLiveLocation* attribute), 365

- `proximity_alert_radius` (`aiogram.methods.send_location.SendLocation` attribute), 292
- `proximity_alert_radius` (`aiogram.types.inline_query_result_location.InlineQueryResultLocation` attribute), 47
- `proximity_alert_radius` (`aiogram.types.input_location_message_content.InputLocationMessageContent` attribute), 61
- `proximity_alert_radius` (`aiogram.types.location.Location` attribute), 158
- `PROXIMITY_ALERT_TRIGGERED` (`aiogram.enums.content_type.ContentType` attribute), 379
- `proximity_alert_triggered` (`aiogram.types.message.Message` attribute), 165
- `ProximityAlertTriggered` (class in `aiogram.types.proximity_alert_triggered`), 207
- `PYG` (`aiogram.enums.currency.Currency` attribute), 382
- Python Enhancement Proposals
- PEP 484, 3
  - PEP 492, 3
- ## Q
- `QAR` (`aiogram.enums.currency.Currency` attribute), 382
- `query` (`aiogram.types.chosen_inline_result.ChosenInlineResult` attribute), 17
- `query` (`aiogram.types.inline_query.InlineQuery` attribute), 18
- `query` (`aiogram.types.switch_inline_query_chosen_chat.SwitchInlineQueryChosenChat` attribute), 209
- `query_id` (`aiogram.utils.web_app.WebAppInitData` attribute), 456
- `question` (`aiogram.methods.send_poll.SendPoll` attribute), 299
- `question` (`aiogram.types.poll.Poll` attribute), 205
- `QUIZ` (`aiogram.enums.poll_type.PollType` attribute), 387

## R

`read()` (`aiogram.types.input_file.BufferedInputFile` method), 149

`read()` (`aiogram.types.input_file.FSInputFile` method), 149

`read()` (`aiogram.types.input_file.InputFile` method), 149

`read()` (`aiogram.types.input_file.URLInputFile` method), 149

`receiver` (`aiogram.utils.web_app.WebAppInitData` attribute), 457

`RECORD_VIDEO` (`aiogram.enums.chat_action.ChatAction` attribute), 377

`record_video()` (`aiogram.utils.chat_action.ChatActionSender` class method), 453

`RECORD_VIDEO_NOTE` (`aiogram.enums.chat_action.ChatAction` attribute), 377

`record_video_note()` (`aiogram.utils.chat_action.ChatActionSender` class method), 453

`RECORD_VOICE` (`aiogram.enums.chat_action.ChatAction` attribute), 377

`record_voice()` (`aiogram.utils.chat_action.ChatActionSender` class method), 453

`RED` (`aiogram.enums.topic_icon_color.TopicIconColor` attribute), 387

`RedisStorage` (class in `aiogram.fsm.storage.redis`), 431

`regex_match` (`aiogram.filters.command.CommandObject` attribute), 403

`register()` (`aiogram.webhook.aiohttp_server.BaseRequestHandler` method), 416

`register()` (`aiogram.webhook.aiohttp_server.SimpleRequestHandler` method), 416

`register()` (`aiogram.webhook.aiohttp_server.TokenBasedRequestHandler` method), 417

`REGULAR` (`aiogram.enums.poll_type.PollType` attribute), 387

`REGULAR` (`aiogram.enums.sticker_type.StickerType` attribute), 387

`remove_keyboard` (`aiogram.types.reply_keyboard_remove.ReplyKeyboardRemove` attribute), 208

`render()` (`aiogram.utils.formatting.Text` method), 464

`RENTAL_AGREEMENT` (`aiogram.enums.encrypted_passport_element.EncryptedPassportElement` attribute), 383

`ReopenForumTopic` (class in `aiogram.methods.reopen_forum_topic`), 276

`ReopenGeneralForumTopic` (class in `aiogram.methods.reopen_general_forum_topic`), 277

`reply()` (`aiogram.types.message.Message` method), 181

`reply_animation()` (`aiogram.types.message.Message` method), 166

`reply_audio()` (`aiogram.types.message.Message` method), 168

`reply_contact()` (`aiogram.types.message.Message` method), 170

`reply_dice()` (`aiogram.types.message.Message` method), 186

`reply_document()` (`aiogram.types.message.Message` method), 172

`reply_game()` (`aiogram.types.message.Message` method), 174

`reply_invoice()` (`aiogram.types.message.Message` method), 175

`reply_location()` (`aiogram.types.message.Message` method), 178

`reply_markup` (`aiogram.methods.copy_message.CopyMessage`

[attribute](#)), 244  
[reply\\_markup\(aiogram.methods.edit\\_message\\_caption.EditMessageCaption\)](#) ([aiogram.types.inline\\_query\\_result\\_cached\\_gif.InlineQueryResultCachedGif](#) attribute), 363  
[reply\\_markup\(aiogram.methods.edit\\_message\\_live\\_location.EditMessageLiveLocation\)](#) ([aiogram.types.inline\\_query\\_result\\_cached\\_mpeg4\\_gif.InlineQueryResultCachedMpeg4Gif](#) attribute), 365  
[reply\\_markup\(aiogram.methods.edit\\_message\\_media.EditMessageMedia\)](#) ([aiogram.types.inline\\_query\\_result\\_cached\\_photo.InlineQueryResultCachedPhoto](#) attribute), 366  
[reply\\_markup\(aiogram.methods.edit\\_message\\_reply\\_markup.EditMessageReplyMarkup\)](#) ([aiogram.types.inline\\_query\\_result\\_cached\\_sticker.InlineQueryResultCachedSticker](#) attribute), 367  
[reply\\_markup\(aiogram.methods.edit\\_message\\_text.EditMessageText\)](#) ([aiogram.types.inline\\_query\\_result\\_cached\\_video.InlineQueryResultCachedVideo](#) attribute), 369  
[reply\\_markup\(aiogram.methods.send\\_animation.SendAnimation\)](#) ([aiogram.types.inline\\_query\\_result\\_cached\\_voice.InlineQueryResultCachedVoice](#) attribute), 281  
[reply\\_markup\(aiogram.methods.send\\_audio.SendAudio\)](#) ([aiogram.types.inline\\_query\\_result\\_contact.InlineQueryResultContact](#) attribute), 284  
[reply\\_markup\(aiogram.methods.send\\_contact.SendContact\)](#) ([aiogram.types.inline\\_query\\_result\\_document.InlineQueryResultDocument](#) attribute), 287  
[reply\\_markup\(aiogram.methods.send\\_dice.SendDice\)](#) ([aiogram.types.inline\\_query\\_result\\_game.InlineQueryResultGame](#) attribute), 288  
[reply\\_markup\(aiogram.methods.send\\_document.SendDocument\)](#) ([aiogram.types.inline\\_query\\_result\\_gif.InlineQueryResultGif](#) attribute), 290  
[reply\\_markup\(aiogram.methods.send\\_game.SendGame\)](#) ([aiogram.types.inline\\_query\\_result\\_location.InlineQueryResultLocation](#) attribute), 353  
[reply\\_markup\(aiogram.methods.send\\_invoice.SendInvoice\)](#) ([aiogram.types.inline\\_query\\_result\\_mpeg4\\_gif.InlineQueryResultMpeg4Gif](#) attribute), 336  
[reply\\_markup\(aiogram.methods.send\\_location.SendLocation\)](#) ([aiogram.types.inline\\_query\\_result\\_photo.InlineQueryResultPhoto](#) attribute), 292  
[reply\\_markup\(aiogram.methods.send\\_message.SendMessage\)](#) ([aiogram.types.inline\\_query\\_result\\_venue.InlineQueryResultVenue](#) attribute), 296  
[reply\\_markup\(aiogram.methods.send\\_photo.SendPhoto\)](#) ([aiogram.types.inline\\_query\\_result\\_video.InlineQueryResultVideo](#) attribute), 298  
[reply\\_markup\(aiogram.methods.send\\_poll.SendPoll\)](#) ([aiogram.types.inline\\_query\\_result\\_voice.InlineQueryResultVoice](#) attribute), 300  
[reply\\_markup\(aiogram.methods.send\\_sticker.SendSticker\)](#) ([aiogram.types.message.Message](#) attribute), 343  
[reply\\_markup\(aiogram.methods.send\\_venue.SendVenue\)](#) ([aiogram.types.message.Message](#) attribute), 302  
[reply\\_markup\(aiogram.methods.send\\_video.SendVideo\)](#) ([aiogram.types.message.Message](#) attribute), 304  
[reply\\_markup\(aiogram.methods.send\\_video\\_note.SendVideoNote\)](#) ([aiogram.types.message.Message](#) attribute), 306  
[reply\\_markup\(aiogram.methods.send\\_voice.SendVoice\)](#) ([aiogram.types.message.Message](#) attribute), 308  
[reply\\_markup\(aiogram.methods.stop\\_message\\_live\\_location.StopMessageLiveLocation\)](#) ([aiogram.types.message.Message](#) attribute), 370  
[reply\\_markup\(aiogram.methods.stop\\_poll.StopPoll\)](#) ([aiogram.types.message.Message](#) attribute), 371  
[reply\\_markup\(aiogram.types.inline\\_query\\_result\\_article.InlineQueryResultArticle\)](#) ([aiogram.methods.copy\\_message.CopyMessage](#) attribute), 21  
[reply\\_markup\(aiogram.types.inline\\_query\\_result\\_audio.InlineQueryResultAudio\)](#) ([aiogram.methods.send\\_animation.SendAnimation](#) attribute), 23  
[reply\\_markup\(aiogram.types.inline\\_query\\_result\\_cached\\_audio.InlineQueryResultCachedAudio\)](#) ([aiogram.methods.send\\_animation.SendAnimation](#) attribute), 25  
[reply\\_markup\(aiogram.types.inline\\_query\\_result\\_cached\\_document.InlineQueryResultCachedDocument\)](#) ([aiogram.methods.send\\_audio](#)



<code>attribute</code> ), 284	<code>ReplyKeyboardBuilder</code> (class in <code>aiogram.utils.keyboard</code> ), 447
<code>reply_to_message_id</code> ( <code>aiogram.methods.send_contact.SendContact</code> attribute), 287	<code>ReplyKeyboardMarkup</code> (class in <code>aiogram.types.reply_keyboard_markup</code> ), 207
<code>reply_to_message_id</code> ( <code>aiogram.methods.send_dice.SendDice</code> attribute), 288	<code>ReplyKeyboardRemove</code> (class in <code>aiogram.types.reply_keyboard_remove</code> ), 208
<code>reply_to_message_id</code> ( <code>aiogram.methods.send_document.SendDocument</code> attribute), 290	<code>request_chat</code> ( <code>aiogram.types.keyboard_button.KeyboardButton</code> attribute), 155
<code>reply_to_message_id</code> ( <code>aiogram.methods.send_game.SendGame</code> attribute), 353	<code>request_contact</code> ( <code>aiogram.types.keyboard_button.KeyboardButton</code> attribute), 155
<code>reply_to_message_id</code> ( <code>aiogram.methods.send_invoice.SendInvoice</code> attribute), 336	<code>request_id</code> ( <code>aiogram.types.chat_shared.ChatShared</code> attribute), 142
<code>reply_to_message_id</code> ( <code>aiogram.methods.send_location.SendLocation</code> attribute), 292	<code>request_id</code> ( <code>aiogram.types.keyboard_button_request_chat.KeyboardButton</code> attribute), 156
<code>reply_to_message_id</code> ( <code>aiogram.methods.send_media_group.SendMediaGroup</code> attribute), 294	<code>request_id</code> ( <code>aiogram.types.keyboard_button_request_user.KeyboardButton</code> attribute), 157
<code>reply_to_message_id</code> ( <code>aiogram.methods.send_message.SendMessage</code> attribute), 296	<code>request_id</code> ( <code>aiogram.types.user_shared.UserShared</code> attribute), 211
<code>reply_to_message_id</code> ( <code>aiogram.methods.send_photo.SendPhoto</code> attribute), 298	<code>request_location</code> ( <code>aiogram.types.keyboard_button.KeyboardButton</code> attribute), 155
<code>reply_to_message_id</code> ( <code>aiogram.methods.send_poll.SendPoll</code> attribute), 300	<code>request_poll</code> ( <code>aiogram.types.keyboard_button.KeyboardButton</code> attribute), 155
<code>reply_to_message_id</code> ( <code>aiogram.methods.send_sticker.SendSticker</code> attribute), 343	<code>request_user</code> ( <code>aiogram.types.keyboard_button.KeyboardButton</code> attribute), 155
<code>reply_to_message_id</code> ( <code>aiogram.methods.send_venue.SendVenue</code> attribute), 302	<code>request_write_access</code> ( <code>aiogram.types.login_url.LoginUrl</code> attribute), 158
<code>reply_to_message_id</code> ( <code>aiogram.methods.send_video.SendVideo</code> attribute), 304	<code>resize_keyboard</code> ( <code>aiogram.types.reply_keyboard_markup.ReplyKeyboardMarkup</code> attribute), 207
<code>reply_to_message_id</code> ( <code>aiogram.methods.send_video_note.SendVideoNote</code> attribute), 306	<code>resolve_bot()</code> ( <code>aiogram.webhook.aihttp_server.BaseRequestHandler</code> method), 416
<code>reply_to_message_id</code> ( <code>aiogram.methods.send_voice.SendVoice</code> attribute), 308	<code>resolve_bot()</code> ( <code>aiogram.webhook.aihttp_server.SimpleRequestHandler</code> method), 417
<code>reply_venue()</code> ( <code>aiogram.types.message.Message</code> method), 189	<code>resolve_bot()</code> ( <code>aiogram.webhook.aihttp_server.TokenBasedRequestHandler</code> method), 417
<code>reply_video()</code> ( <code>aiogram.types.message.Message</code> method), 191	<code>resolve_used_update_types()</code> ( <code>aiogram.dispatcher.router.Router</code> method), 393
<code>reply_video_note()</code> ( <code>aiogram.types.message.Message</code> method), 193	<code>ResponseParameters</code> (class in <code>aiogram.types.response_parameters</code> ), 208
<code>reply_voice()</code> ( <code>aiogram.types.message.Message</code> method), 195	<code>RestartingTelegram</code> , 437
	<code>restrict()</code> ( <code>aiogram.types.chat.Chat</code> method), 81
	<code>RestrictChatMember</code> (class in <code>aiogram.methods.restrict_chat_member</code> ), 278
	<code>RESTRICTED</code> ( <code>aiogram.enums.chat_member_status.ChatMemberStatus</code> attribute), 377
	<code>result</code> ( <code>aiogram.methods.answer_web_app_query.AnswerWebAppQuery</code> attribute), 375
	<code>result_id</code> ( <code>aiogram.types.chosen_inline_result.ChosenInlineResult</code> attribute), 17
	<code>results</code> ( <code>aiogram.methods.answer_inline_query.AnswerInlineQuery</code> attribute), 373

[retry\\_after\(aiogram.types.response\\_parameters.ResponseParameters attribute\), 208](#)  
[REVERSE\\_SIDE\(aiogram.enums.passport\\_element\\_error\\_type.PassportElementErrorType attribute\), 386](#)  
[reverse\\_side\(aiogram.types.encrypted\\_passport\\_element.EncryptedPassportElement attribute\), 217](#)  
[revoke\\_invite\\_link\(\)\(aiogram.types.chat.Chat method\), 75](#)  
[revoke\\_messages\(aiogram.methods.ban\\_chat\\_member.BanChatMember attribute\), 239](#)  
[RevokeChatInviteLink\(class in aiogram.methods.revoke\\_chat\\_invite\\_link\), 279](#)  
[rights\(aiogram.methods.set\\_my\\_default\\_administrator\\_rights.SetMyDefaultAdministratorRights attribute\), 318](#)  
[RON\(aiogram.enums.currency.Currency attribute\), 382](#)  
[ROSE\(aiogram.enums.topic\\_icon\\_color.TopicIconColor attribute\), 387](#)  
[Router\(class in aiogram.dispatcher.router\), 393](#)  
[row\(\)\(aiogram.utils.keyboard.InlineKeyboardBuilder method\), 447](#)  
[row\(\)\(aiogram.utils.keyboard.ReplyKeyboardBuilder method\), 448](#)  
[RSD\(aiogram.enums.currency.Currency attribute\), 382](#)  
[RUB\(aiogram.enums.currency.Currency attribute\), 382](#)  
[run\\_polling\(\)\(aiogram.dispatcher.dispatcher.Dispatcher method\), 398](#)

**S**

[safe\\_parse\\_webapp\\_init\\_data\(\)\(in module aiogram.utils.web\\_app\), 456](#)  
[SAR\(aiogram.enums.currency.Currency attribute\), 382](#)  
[scale\(aiogram.types.mask\\_position.MaskPosition attribute\), 228](#)  
[scope\(aiogram.methods.delete\\_my\\_commands.DeleteMyCommands attribute\), 252](#)  
[scope\(aiogram.methods.get\\_my\\_commands.GetMyCommands attribute\), 265](#)  
[scope\(aiogram.methods.set\\_my\\_commands.SetMyCommands attribute\), 316](#)  
[score\(aiogram.methods.set\\_game\\_score.SetGameScore attribute\), 355](#)  
[score\(aiogram.types.game\\_high\\_score.GameHighScore attribute\), 236](#)  
[secret\(aiogram.types.encrypted\\_credentials.EncryptedCredentials attribute\), 216](#)  
[secret\\_token\(aiogram.methods.set\\_webhook.SetWebhook attribute\), 359](#)  
[SEK\(aiogram.enums.currency.Currency attribute\), 382](#)  
[selective\(aiogram.types.force\\_reply.ForceReply attribute\), 145](#)  
[selective\(aiogram.types.reply\\_keyboard\\_markup.ReplyKeyboardMarkup attribute\), 207](#)

[selective\(aiogram.types.reply\\_keyboard\\_remove.ReplyKeyboardRemove attribute\), 208](#)  
[SELF\(aiogram.enums.passport\\_element\\_error\\_type.PassportElementErrorType attribute\), 386](#)  
[SELF\(aiogram.types.encrypted\\_passport\\_element.EncryptedPassportElement attribute\), 217](#)  
[send\\_copy\(\)\(aiogram.types.message.Message method\), 196](#)  
[send\\_animation\\_to\\_provider\(aiogram.methods.create\\_invoice\\_link.CreateInvoiceLink attribute\), 333](#)  
[send\\_email\\_to\\_provider\(aiogram.methods.send\\_invoice.SendInvoice attribute\), 335](#)  
[send\\_email\\_to\\_provider\(aiogram.types.input\\_invoice\\_message\\_content.InputInvoiceMessageContent attribute\), 61](#)  
[send\\_phone\\_number\\_to\\_provider\(aiogram.methods.create\\_invoice\\_link.CreateInvoiceLink attribute\), 333](#)  
[send\\_phone\\_number\\_to\\_provider\(aiogram.methods.send\\_invoice.SendInvoice attribute\), 335](#)  
[send\\_phone\\_number\\_to\\_provider\(aiogram.types.input\\_invoice\\_message\\_content.InputInvoiceMessageContent attribute\), 60](#)  
[SendAnimation\(class in aiogram.methods.send\\_animation\), 280](#)  
[SendAudio\(class in aiogram.methods.send\\_audio\), 282](#)  
[SendChatAction\(class in aiogram.methods.send\\_chat\\_action\), 285](#)  
[SendContact\(class in aiogram.methods.send\\_contact\), 286](#)  
[SendDice\(class in aiogram.methods.send\\_dice\), 288](#)  
[SendDocument\(class in aiogram.methods.send\\_document\), 289](#)  
[SENDER\(aiogram.enums.chat\\_type.ChatType attribute\), 378](#)  
[sender\\_chat\(aiogram.types.message.Message attribute\), 162](#)  
[sender\\_chat\\_id\(aiogram.methods.ban\\_chat\\_sender\\_chat.BanChatSenderChat attribute\), 240](#)  
[sender\\_chat\\_id\(aiogram.methods.unban\\_chat\\_sender\\_chat.UnbanChatSenderChat attribute\), 323](#)  
[SendGame\(class in aiogram.methods.send\\_game\), 353](#)  
[SendInvoice\(class in aiogram.methods.send\\_invoice\), 334](#)  
[SendLocation\(class in aiogram.methods.send\\_location\), 291](#)  
[SendMediaGroup\(class in aiogram.methods.send\\_media\\_group\), 293](#)  
[SendMessage\(class in aiogram.methods.send\\_message\), 295](#)  
[SendPhoto\(class in aiogram.methods.send\\_photo\), 297](#)

- SendPoll (class in aiogram.methods.send\_poll), 299
- SendSticker (class in aiogram.methods.send\_sticker), 342
- SendVenue (class in aiogram.methods.send\_venue), 301
- SendVideo (class in aiogram.methods.send\_video), 303
- SendVideoNote (class in aiogram.methods.send\_video\_note), 305
- SendVoice (class in aiogram.methods.send\_voice), 307
- SentWebAppMessage (class in aiogram.types.sent\_web\_app\_message), 63
- set\_administrator\_custom\_title() (aiogram.types.chat.Chat method), 79
- set\_data() (aiogram.fsm.storage.base.BaseStorage method), 432
- set\_description() (aiogram.types.chat.Chat method), 82
- set\_locale() (aiogram.utils.i18n.middleware.FSMI18nMiddleware method), 450
- set\_name (aiogram.types.sticker.Sticker attribute), 229
- set\_permissions() (aiogram.types.chat.Chat method), 80
- set\_photo() (aiogram.types.chat.Chat method), 83
- set\_position\_in\_set() (aiogram.types.sticker.Sticker method), 229
- set\_state() (aiogram.fsm.storage.base.BaseStorage method), 432
- set\_sticker\_set() (aiogram.types.chat.Chat method), 77
- set\_title() (aiogram.types.chat.Chat method), 83
- SetChatAdministratorCustomTitle (class in aiogram.methods.set\_chat\_administrator\_custom\_title), 309
- SetChatDescription (class in aiogram.methods.set\_chat\_description), 310
- SetChatMenuButton (class in aiogram.methods.set\_chat\_menu\_button), 311
- SetChatPermissions (class in aiogram.methods.set\_chat\_permissions), 312
- SetChatPhoto (class in aiogram.methods.set\_chat\_photo), 313
- SetChatStickerSet (class in aiogram.methods.set\_chat\_sticker\_set), 314
- SetChatTitle (class in aiogram.methods.set\_chat\_title), 315
- SetCustomEmojiStickerSetThumbnail (class in aiogram.methods.set\_custom\_emoji\_sticker\_set\_thumbnail), 344
- SetGameScore (class in aiogram.methods.set\_game\_score), 354
- SetMyCommands (class in aiogram.methods.set\_my\_commands), 316
- SetMyDefaultAdministratorRights (class in aiogram.methods.set\_my\_default\_administrator\_rights), 317
- SetMyDescription (class in aiogram.methods.set\_my\_description), 319
- SetMyName (class in aiogram.methods.set\_my\_name), 320
- SetMyShortDescription (class in aiogram.methods.set\_my\_short\_description), 321
- SetPassportDataErrors (class in aiogram.methods.set\_passport\_data\_errors), 360
- SetStickerEmojiList (class in aiogram.methods.set\_sticker\_emoji\_list), 345
- SetStickerKeywords (class in aiogram.methods.set\_sticker\_keywords), 346
- SetStickerMaskPosition (class in aiogram.methods.set\_sticker\_mask\_position), 347
- SetStickerPositionInSet (class in aiogram.methods.set\_sticker\_position\_in\_set), 348
- SetStickerSetThumbnail (class in aiogram.methods.set\_sticker\_set\_thumbnail), 349
- SetStickerSetTitle (class in aiogram.methods.set\_sticker\_set\_title), 350
- setup() (aiogram.utils.i18n.middleware.I18nMiddleware method), 451
- SetWebhook (class in aiogram.methods.set\_webhook), 358
- SGD (aiogram.enums.currency.Currency attribute), 382
- shifted\_id (aiogram.types.chat.Chat property), 73
- shipping\_address (aiogram.types.order\_info.OrderInfo attribute), 231
- shipping\_address (aiogram.types.shipping\_query.ShippingQuery attribute), 233
- shipping\_option\_id (aiogram.types.pre\_checkout\_query.PreCheckoutQuery attribute), 232
- shipping\_option\_id (aiogram.types.successful\_payment.SuccessfulPayment attribute), 234
- shipping\_options (aiogram.methods.answer\_shipping\_query.AnswerShippingQuery attribute), 330
- SHIPPING\_QUERY (aiogram.enums.update\_type.UpdateType attribute), 388
- shipping\_query (aiogram.types.update.Update attribute), 226
- shipping\_query\_id (aiogram.methods.answer\_shipping\_query.AnswerShippingQuery attribute), 330
- ShippingAddress (class in aiogram.types.shipping\_address), 232
- ShippingOption (class in aiogram.methods.set\_my\_default\_administrator\_rights), 317

aioogram.types.shipping_option),	233
ShippingQuery	(class in start_parameter (aioogram.types.invoice.Invoice attribute), 230)
short_description(aioogram.methods.set_my_short_description),	321
short_description(aioogram.types.bot_short_descriptions_base),	69
show_alert(aioogram.methods.answer_callback_query.CallbackAnswer),	236
show_alert(aioogram.utils.callback_answer.CallbackAnswer property),	460
SimpleMiddleware	(class in status (aioogram.types.chat_member_banned.ChatMemberBanned attribute), 450)
SimpleRequestHandler	(class in status (aioogram.types.chat_member_left.ChatMemberLeft attribute), 416)
SLOT_MACHINE (aioogram.enums.dice_emoji.DiceEmoji attribute),	383
SLOT_MACHINE (aioogram.types.dice.DiceEmoji attribute),	143
slow_mode_delay (aioogram.types.chat.Chat attribute),	73
small_file_id (aioogram.types.chat_photo.ChatPhoto attribute),	142
small_file_unique_id (aioogram.types.chat_photo.ChatPhoto attribute),	142
source(aioogram.types.passport_element_error_data_field.PassportElementErrorDataField attribute),	218
source(aioogram.types.passport_element_error_file.PassportElementErrorFile attribute),	219
source(aioogram.types.passport_element_error_files.PassportElementErrorFiles attribute),	219
source(aioogram.types.passport_element_error_front_side.PassportElementErrorFrontSide attribute),	220
source(aioogram.types.passport_element_error_reverse_side.PassportElementErrorReverseSide attribute),	221
source(aioogram.types.passport_element_error_selfie.PassportElementErrorSelfie attribute),	222
source(aioogram.types.passport_element_error_translation_file.PassportElementErrorTranslationFile attribute),	222
source(aioogram.types.passport_element_error_translation_files.PassportElementErrorTranslationFiles attribute),	223
source(aioogram.types.passport_element_error_unspecified.PassportElementErrorUnspecified attribute),	224
SPOILER(aioogram.enums.message_entity_type.MessageEntityType),	385
Spoiler(class in aioogram.utils.formatting),	466
start_date(aioogram.types.video_chat_scheduled.VideoChatScheduled attribute),	213
start_param (aioogram.utils.web_app.WebAppInitData attribute),	457
start_parameter(aioogram.methods.send_invoice.SendInvoice attribute),	335
start_parameter(aioogram.types.inline_query_results_button.Button attribute),	335
status(aioogram.types.invoice.Invoice attribute),	230
status(aioogram.types.shipping_address.ShippingAddress attribute),	232
sticker(aioogram.enums.sticker_format.StickerFormat attribute),	387
status(aioogram.types.chat_member_administrator.ChatMemberAdministrator attribute),	120
status(aioogram.types.chat_member_banned.ChatMemberBanned attribute),	121
status(aioogram.types.chat_member_left.ChatMemberLeft attribute),	122
status(aioogram.types.chat_member_member.ChatMemberMember attribute),	122
status(aioogram.types.chat_member_owner.ChatMemberOwner attribute),	122
status(aioogram.types.chat_member_restricted.ChatMemberRestricted attribute),	123
STICKER (aioogram.enums.content_type.ContentType attribute),	378
STICKER (aioogram.enums.inline_query_result_type.InlineQueryResultType attribute),	384
sticker(aioogram.methods.add_sticker_to_set.AddStickerToSet attribute),	339
sticker(aioogram.methods.delete_sticker_from_set.DeleteStickerFromSet attribute),	339
sticker(aioogram.methods.send_sticker.SendSticker attribute),	343
sticker(aioogram.methods.set_sticker_emoji_list.SetStickerEmojiList attribute),	345
sticker(aioogram.methods.set_sticker_keywords.SetStickerKeywords attribute),	346
sticker(aioogram.methods.set_sticker_mask_position.SetStickerMaskPosition attribute),	347
sticker(aioogram.methods.set_sticker_position_in_set.SetStickerPositionInSet attribute),	347
sticker(aioogram.methods.upload_sticker_file.UploadStickerFile attribute),	351
sticker_set_name(aioogram.methods.set_chat_sticker_set.SetChatStickerSet attribute),	314
sticker_file_id(aioogram.types.inline_query_result_cached_sticker.InlineQueryResultCachedSticker attribute),	35
sticker_format(aioogram.methods.create_new_sticker_set.CreateNewStickerSet attribute),	338
sticker_format(aioogram.methods.upload_sticker_file.UploadStickerFile attribute),	351
sticker_set_name(aioogram.methods.set_chat_sticker_set.SetChatStickerSet attribute),	314
sticker_set_name(aioogram.types.chat.Chat attribute),	314



73

sticker\_type (aiogram.methods.create\_new\_sticker\_set.CreateNewStickerSet attribute), 338

sticker\_type (aiogram.types.sticker\_set.StickerSet attribute), 230

StickerFormat (class in aiogram.enums.sticker\_format), 387

stickers (aiogram.methods.create\_new\_sticker\_set.CreateNewStickerSet attribute), 338

stickers (aiogram.types.sticker\_set.StickerSet attribute), 230

StickerSet (class in aiogram.types.sticker\_set), 230

StickerType (class in aiogram.enums.sticker\_type), 387

stop\_live\_location() (aiogram.types.message.Message method), 201

stop\_polling() (aiogram.dispatcher.dispatcher.Dispatcher method), 399

StopMessageLiveLocation (class in aiogram.methods.stop\_message\_live\_location), 370

StopPoll (class in aiogram.methods.stop\_poll), 371

STORY (aiogram.enums.content\_type.ContentType attribute), 378

story (aiogram.types.message.Message attribute), 163

Story (class in aiogram.types.story), 209

stream\_content() (aiogram.client.session.base.BaseSession method), 13

street\_line1 (aiogram.types.shipping\_address.ShippingAddress attribute), 233

street\_line2 (aiogram.types.shipping\_address.ShippingAddress attribute), 233

STRIKETHROUGH (aiogram.enums.message\_entity\_type.MessageEntityType attribute), 385

Strikethrough (class in aiogram.utils.formatting), 466

SUCCESSFUL\_PAYMENT (aiogram.enums.content\_type.ContentType attribute), 379

successful\_payment (aiogram.types.message.Message attribute), 165

SuccessfulPayment (class in aiogram.types.successful\_payment), 234

suggested\_tip\_amounts (aiogram.methods.create\_invoice\_link.CreateInvoiceLink attribute), 332

suggested\_tip\_amounts (aiogram.methods.send\_invoice.SendInvoice attribute), 335

suggested\_tip\_amounts (aiogram.types.input\_invoice\_message\_content.InputInvoiceMessageContent attribute), 60

SUPERGROUP (aiogram.enums.chat\_type.ChatType attribute), 378

SUPERGROUP\_CHAT\_CREATED (aiogram.enums.content\_type.ContentType attribute), 379

super\_group\_chat\_created (aiogram.types.message.Message attribute), 164

supports\_inline\_queries (aiogram.types.user.User attribute), 210

supports\_streaming (aiogram.methods.send\_video.SendVideo attribute), 304

supports\_streaming (aiogram.types.input\_media\_video.InputMediaVideo attribute), 154

switch\_inline\_query (aiogram.types.inline\_keyboard\_button.InlineKeyboardButton attribute), 148

switch\_inline\_query\_chosen\_chat (aiogram.types.inline\_keyboard\_button.InlineKeyboardButton attribute), 148

switch\_inline\_query\_current\_chat (aiogram.types.inline\_keyboard\_button.InlineKeyboardButton attribute), 148

switch\_pm\_parameter (aiogram.methods.answer\_inline\_query.AnswerInlineQuery attribute), 373

switch\_pm\_text (aiogram.methods.answer\_inline\_query.AnswerInlineQuery attribute), 373

SwitchInlineQueryChosenChat (class in aiogram.types.switch\_inline\_query\_chosen\_chat), 209

**T**

telegram\_payment\_charge\_id (aiogram.types.successful\_payment.SuccessfulPayment attribute), 234

TelegramAPIError, 437

TelegramAPIServer (class in aiogram.client.telegram), 12

TelegramBadRequest, 437

TelegramConflictError, 437

TelegramEntityTooLarge, 437

TelegramForbiddenError, 437

TelegramMigrateToChat, 437

TelegramNetworkError, 437

TelegramNotFound, 437

TelegramRetryAfter, 437

TelegramServerError, 437

TelegramUnauthorizedError, 437

TEMPORARY\_REGISTRATION (aiogram.enums.encrypted\_passport\_element.EncryptedPassportElement attribute), 383

text (aiogram.filters.command.CommandObject property), 403

text (aiogram.methods.answer\_callback\_query.AnswerCallbackQuery attribute), 236

text (aiogram.methods.edit\_message\_text.EditMessageText attribute), 154  
 text (aiogram.methods.send\_message.SendMessage attribute), 368  
 text (aiogram.types.game.Game attribute), 235  
 text (aiogram.types.inline\_keyboard\_button.InlineKeyboardButton attribute), 147  
 text (aiogram.types.inline\_query\_results\_button.InlineQueryResultsButton attribute), 57  
 text (aiogram.types.keyboard\_button.KeyboardButton attribute), 155  
 text (aiogram.types.menu\_button.MenuButton attribute), 159  
 text (aiogram.types.menu\_button\_web\_app.MenuButtonWebApp attribute), 160  
 text (aiogram.types.message.Message attribute), 163  
 text (aiogram.types.poll\_option.PollOption attribute), 206  
 text (aiogram.utils.callback\_answer.CallbackAnswer property), 460  
 Text (class in aiogram.utils.formatting), 464  
 text\_entities (aiogram.types.game.Game attribute), 235  
 TEXT\_LINK (aiogram.enums.message\_entity\_type.MessageEntityType attribute), 386  
 TEXT\_MENTION (aiogram.enums.message\_entity\_type.MessageEntityType attribute), 386  
 TextLink (class in aiogram.utils.formatting), 466  
 TextMention (class in aiogram.utils.formatting), 467  
 THB (aiogram.enums.currency.Currency attribute), 382  
 thumbnail (aiogram.methods.send\_animation.SendAnimation attribute), 281  
 thumbnail (aiogram.methods.send\_audio.SendAudio attribute), 283  
 thumbnail (aiogram.methods.send\_document.SendDocument attribute), 290  
 thumbnail (aiogram.methods.send\_video.SendVideo attribute), 304  
 thumbnail (aiogram.methods.send\_video\_note.SendVideoNote attribute), 306  
 thumbnail (aiogram.methods.set\_sticker\_set\_thumbnail.SetStickerSetThumbnail attribute), 349  
 thumbnail (aiogram.types.animation.Animation attribute), 64  
 thumbnail (aiogram.types.audio.Audio attribute), 65  
 thumbnail (aiogram.types.document.Document attribute), 144  
 thumbnail (aiogram.types.input\_media\_animation.InputMediaAnimation attribute), 150  
 thumbnail (aiogram.types.input\_media\_audio.InputMediaAudio attribute), 151  
 thumbnail (aiogram.types.input\_media\_document.InputMediaDocument attribute), 152  
 thumbnail (aiogram.types.input\_media\_video.InputMediaVideo attribute), 154  
 thumbnail (aiogram.types.sticker.Sticker attribute), 229  
 thumbnail (aiogram.types.sticker\_set.StickerSet attribute), 230  
 thumbnail (aiogram.types.video.Video attribute), 212  
 thumbnail (aiogram.types.video\_note.VideoNote attribute), 214  
 thumbnail\_height (aiogram.types.inline\_query\_result\_article.InlineQueryResultArticle attribute), 21  
 thumbnail\_height (aiogram.types.inline\_query\_result\_contact.InlineQueryResultContact attribute), 41  
 thumbnail\_height (aiogram.types.inline\_query\_result\_document.InlineQueryResultDocument attribute), 43  
 thumbnail\_height (aiogram.types.inline\_query\_result\_location.InlineQueryResultLocation attribute), 47  
 thumbnail\_height (aiogram.types.inline\_query\_result\_venue.InlineQueryResultVenue attribute), 53  
 thumbnail\_mime\_type (aiogram.types.inline\_query\_result\_gif.InlineQueryResultGif attribute), 45  
 thumbnail\_mime\_type (aiogram.types.inline\_query\_result\_mpeg4\_gif.InlineQueryResultMpeg4Gif attribute), 49  
 thumbnail\_url (aiogram.types.inline\_query\_result\_article.InlineQueryResultArticle attribute), 21  
 thumbnail\_url (aiogram.types.inline\_query\_result\_contact.InlineQueryResultContact attribute), 41  
 thumbnail\_url (aiogram.types.inline\_query\_result\_document.InlineQueryResultDocument attribute), 43  
 thumbnail\_url (aiogram.types.inline\_query\_result\_gif.InlineQueryResultGif attribute), 45  
 thumbnail\_url (aiogram.types.inline\_query\_result\_location.InlineQueryResultLocation attribute), 47  
 thumbnail\_url (aiogram.types.inline\_query\_result\_mpeg4\_gif.InlineQueryResultMpeg4Gif attribute), 49  
 thumbnail\_url (aiogram.types.inline\_query\_result\_photo.InlineQueryResultPhoto attribute), 50  
 thumbnail\_url (aiogram.types.inline\_query\_result\_venue.InlineQueryResultVenue attribute), 53  
 thumbnail\_url (aiogram.types.inline\_query\_result\_video.InlineQueryResultVideo attribute), 55  
 thumbnail\_width (aiogram.types.inline\_query\_result\_article.InlineQueryResultArticle attribute), 21  
 thumbnail\_width (aiogram.types.inline\_query\_result\_contact.InlineQueryResultContact attribute), 41  
 thumbnail\_width (aiogram.types.inline\_query\_result\_document.InlineQueryResultDocument attribute), 43  
 thumbnail\_width (aiogram.types.inline\_query\_result\_location.InlineQueryResultLocation attribute), 47  
 thumbnail\_width (aiogram.types.inline\_query\_result\_venue.InlineQueryResultVenue attribute), 53  
 thumbnail\_width (aiogram.types.inline\_query\_result\_video.InlineQueryResultVideo attribute), 55  
 time (aiogram.methods.get\_updates.GetUpdates attribute), 357  
 title (aiogram.methods.create\_invoice\_link.CreateInvoiceLink attribute), 357

- attribute), 332
- title (aiogram.methods.create\_new\_sticker\_set.CreateNewStickerSet attribute), 338
- title (aiogram.methods.send\_audio.SendAudio attribute), 283
- title (aiogram.methods.send\_invoice.SendInvoice attribute), 334
- title (aiogram.methods.send\_venue.SendVenue attribute), 301
- title (aiogram.methods.set\_chat\_title.SetChatTitle attribute), 315
- title (aiogram.methods.set\_sticker\_set\_title.SetStickerSetTitle attribute), 350
- title (aiogram.types.audio.Audio attribute), 65
- title (aiogram.types.chat.Chat attribute), 71
- title (aiogram.types.game.Game attribute), 235
- title (aiogram.types.inline\_query\_result\_article.InlineQueryResultArticle attribute), 20
- title (aiogram.types.inline\_query\_result\_audio.InlineQueryResultAudio attribute), 22
- title (aiogram.types.inline\_query\_result\_cached\_document.InlineQueryResultCachedDocument attribute), 27
- title (aiogram.types.inline\_query\_result\_cached\_gif.InlineQueryResultCachedGif attribute), 28
- title (aiogram.types.inline\_query\_result\_cached\_mpeg4\_gif.InlineQueryResultCachedMpeg4Gif attribute), 31
- title (aiogram.types.inline\_query\_result\_cached\_photo.InlineQueryResultCachedPhoto attribute), 33
- title (aiogram.types.inline\_query\_result\_cached\_video.InlineQueryResultCachedVideo attribute), 37
- title (aiogram.types.inline\_query\_result\_cached\_voice.InlineQueryResultCachedVoice attribute), 39
- title (aiogram.types.inline\_query\_result\_document.InlineQueryResultDocument attribute), 42
- title (aiogram.types.inline\_query\_result\_gif.InlineQueryResultGif attribute), 45
- title (aiogram.types.inline\_query\_result\_location.InlineQueryResultLocation attribute), 47
- title (aiogram.types.inline\_query\_result\_mpeg4\_gif.InlineQueryResultMpeg4Gif attribute), 49
- title (aiogram.types.inline\_query\_result\_photo.InlineQueryResultPhoto attribute), 51
- title (aiogram.types.inline\_query\_result\_venue.InlineQueryResultVenue attribute), 52
- title (aiogram.types.inline\_query\_result\_video.InlineQueryResultVideo attribute), 55
- title (aiogram.types.inline\_query\_result\_voice.InlineQueryResultVoice attribute), 56
- title (aiogram.types.input\_invoice\_message\_content.InputInvoiceMessageContent attribute), 59
- title (aiogram.types.input\_media\_audio.InputMediaAudio attribute), 152
- title (aiogram.types.input\_venue\_message\_content.InputVenueMessageContent attribute), 63
- title (aiogram.types.invoice.Invoice attribute), 230
- title (aiogram.types.shipping\_option.ShippingOption attribute), 233
- title (aiogram.types.sticker\_set.StickerSet attribute), 230
- title (aiogram.types.venue.Venue attribute), 212
- TJS (aiogram.enums.currency.Currency attribute), 382
- TokenBasedRequestHandler (class in aiogram.webhook.aiohttp\_server), 417
- TopicIconColor (class in aiogram.enums.topic\_icon\_color), 387
- total\_amount (aiogram.types.invoice.Invoice attribute), 230
- total\_amount (aiogram.types.pre\_checkout\_query.PreCheckoutQuery attribute), 232
- total\_amount (aiogram.types.successful\_payment.SuccessfulPayment attribute), 234
- total\_count (aiogram.types.user\_profile\_photos.UserProfilePhotos attribute), 211
- total\_voter\_count (aiogram.types.poll.Poll attribute), 225
- translation (aiogram.types.encrypted\_passport\_element.EncryptedPassportElement attribute), 205
- TRANSLATION\_FILE (aiogram.enums.passport\_element\_error\_type.PassportElementErrorType attribute), 205
- TRANSLATION\_FILES (aiogram.enums.passport\_element\_error\_type.PassportElementErrorType attribute), 205
- traveler (aiogram.types.proximity\_alert\_triggered.ProximityAlertTriggered attribute), 201
- TRY (aiogram.enums.currency.Currency attribute), 382
- TWD (aiogram.enums.currency.Currency attribute), 382
- type (aiogram.methods.send\_poll.SendPoll attribute), 299
- type (aiogram.types.bot\_command\_scope\_all\_chat\_administrators.BotCommandScopeAllChatAdministrators attribute), 66
- type (aiogram.types.bot\_command\_scope\_all\_group\_chats.BotCommandScopeAllGroupChats attribute), 66
- type (aiogram.types.bot\_command\_scope\_all\_private\_chats.BotCommandScopeAllPrivateChats attribute), 67
- type (aiogram.types.bot\_command\_scope\_chat.BotCommandScopeChat attribute), 67
- type (aiogram.types.bot\_command\_scope\_chat\_administrators.BotCommandScopeChatAdministrators attribute), 68
- type (aiogram.types.bot\_command\_scope\_chat\_member.BotCommandScopeChatMember attribute), 68
- type (aiogram.types.bot\_command\_scope\_default.BotCommandScopeDefault attribute), 69
- type (aiogram.types.chat.Chat attribute), 71
- type (aiogram.types.encrypted\_passport\_element.EncryptedPassportElement attribute), 216
- type (aiogram.types.inline\_query\_result\_article.InlineQueryResultArticle attribute), 20
- type (aiogram.types.inline\_query\_result\_audio.InlineQueryResultAudio attribute), 22

`attribute`), 22  
**type** (`aiogram.types.inline_query_result_cached_audio.InlineQueryResultCachedAudio` `attribute`), 24  
**type** (`aiogram.types.inline_query_result_cached_document.InlineQueryResultCachedDocument` `attribute`), 27  
**type** (`aiogram.types.inline_query_result_cached_gif.InlineQueryResultCachedGif` `attribute`), 28  
**type** (`aiogram.types.inline_query_result_cached_mpeg4_gif.InlineQueryResultCachedMpeg4Gif` `attribute`), 31  
**type** (`aiogram.types.inline_query_result_cached_photo.InlineQueryResultCachedPhoto` `attribute`), 33  
**type** (`aiogram.types.inline_query_result_cached_sticker.InlineQueryResultCachedSticker` `attribute`), 34  
**type** (`aiogram.types.inline_query_result_cached_video.InlineQueryResultCachedVideo` `attribute`), 36  
**type** (`aiogram.types.inline_query_result_cached_voice.InlineQueryResultCachedVoice` `attribute`), 38  
**type** (`aiogram.types.inline_query_result_contact.InlineQueryResultContact` `attribute`), 40  
**type** (`aiogram.types.inline_query_result_document.InlineQueryResultDocument` `attribute`), 42  
**type** (`aiogram.types.inline_query_result_game.InlineQueryResultGame` `attribute`), 43  
**type** (`aiogram.types.inline_query_result_gif.InlineQueryResultGif` `attribute`), 44  
**type** (`aiogram.types.inline_query_result_location.InlineQueryResultLocation` `attribute`), 46  
**type** (`aiogram.types.inline_query_result_mpeg4_gif.InlineQueryResultMpeg4Gif` `attribute`), 49  
**type** (`aiogram.types.inline_query_result_photo.InlineQueryResultPhoto` `attribute`), 50  
**type** (`aiogram.types.inline_query_result_venue.InlineQueryResultVenue` `attribute`), 52  
**type** (`aiogram.types.inline_query_result_video.InlineQueryResultVideo` `attribute`), 54  
**type** (`aiogram.types.inline_query_result_voice.InlineQueryResultVoice` `attribute`), 56  
**type** (`aiogram.types.input_media_animation.InputMediaAnimation` `attribute`), 150  
**type** (`aiogram.types.input_media_audio.InputMediaAudio` `attribute`), 151  
**type** (`aiogram.types.input_media_document.InputMediaDocument` `attribute`), 152  
**type** (`aiogram.types.input_media_photo.InputMediaPhoto` `attribute`), 153  
**type** (`aiogram.types.input_media_video.InputMediaVideo` `attribute`), 154  
**type** (`aiogram.types.keyboard_button_poll_type.KeyboardButtonPollType` `attribute`), 156  
**type** (`aiogram.types.menu_button.MenuButton` `attribute`), 159  
**type** (`aiogram.types.menu_button_commands.MenuButtonCommands` `attribute`), 159  
**type** (`aiogram.types.menu_button_default.MenuButtonDefault` `attribute`), 160  
**type** (`aiogram.types.result_cached_audio.ResultCachedAudio` `attribute`), 160  
**type** (`aiogram.types.result_cached_document.ResultCachedDocument` `attribute`), 160  
**type** (`aiogram.types.result_cached_gif.ResultCachedGif` `attribute`), 160  
**type** (`aiogram.types.result_cached_mpeg4_gif.ResultCachedMpeg4Gif` `attribute`), 160  
**type** (`aiogram.types.result_cached_photo.ResultCachedPhoto` `attribute`), 160  
**type** (`aiogram.types.result_cached_sticker.ResultCachedSticker` `attribute`), 160  
**type** (`aiogram.types.result_cached_video.ResultCachedVideo` `attribute`), 160  
**type** (`aiogram.types.result_cached_voice.ResultCachedVoice` `attribute`), 160  
**type** (`aiogram.types.passport_element_error_data_field.PassportElementErrorDataField` `attribute`), 204  
**type** (`aiogram.types.passport_element_error_file.PassportElementErrorFile` `attribute`), 218  
**type** (`aiogram.types.passport_element_error_files.PassportElementErrorFiles` `attribute`), 219  
**type** (`aiogram.types.passport_element_error_front_side.PassportElementErrorFrontSide` `attribute`), 220  
**type** (`aiogram.types.passport_element_error_reverse_side.PassportElementErrorReverseSide` `attribute`), 221  
**type** (`aiogram.types.passport_element_error_selfie.PassportElementErrorSelfie` `attribute`), 222  
**type** (`aiogram.types.passport_element_error_translation_file.PassportElementErrorTranslationFile` `attribute`), 222  
**type** (`aiogram.types.passport_element_error_translation_files.PassportElementErrorTranslationFiles` `attribute`), 223  
**type** (`aiogram.types.passport_element_error_unspecified.PassportElementErrorUnspecified` `attribute`), 224  
**type** (`aiogram.types.poll.Poll` `attribute`), 205  
**type** (`aiogram.types.sticker.Sticker` `attribute`), 228  
**type** (`aiogram.enums.chat_action.ChatAction` `attribute`), 377  
**type** (`aiogram.utils.chat_actions.ChatActionSender` `attribute`), 454  
**type** (`aiogram.enums.currency.Currency` `attribute`), 382  
**type** (`aiogram.enums.currency.Currency` `attribute`), 382  
**type** (`aiogram.methods.unban_chat_member.UnbanChatMember` `attribute`), 322  
**type** (`aiogram.methods.unban_chat_sender_chat.UnbanChatSenderChat` `attribute`), 323  
**type** (`aiogram.enums.message_entity_type.MessageEntityType` `attribute`), 385  
**type** (`aiogram.utils.formatting.Underline` `attribute`), 466  
**type** (`aiogram.methods.unhide_general_forum_topic.UnhideGeneralForumTopic` `attribute`), 324  
**type** (`aiogram.enums.content_type.ContentType` `attribute`), 378  
**type** (`aiogram.filters.callback_data.CallbackData` `attribute`), 409  
**type** (`aiogram.types.message.Message` `attribute`), 203



`unpin_all_general_forum_topic_messages()` (*aiogram.types.chat.Chat* method), 84  
`unpin_all_messages()` (*aiogram.types.chat.Chat* method), 78  
`unpin_message()` (*aiogram.types.chat.Chat* method), 79  
`UnpinAllChatMessages` (class in *aiogram.methods.unpin\_all\_chat\_messages*), 325  
`UnpinAllForumTopicMessages` (class in *aiogram.methods.unpin\_all\_forum\_topic\_messages*), 326  
`UnpinAllGeneralForumTopicMessages` (class in *aiogram.methods.unpin\_all\_general\_forum\_topic\_messages*), 327  
`UnpinChatMessage` (class in *aiogram.methods.unpin\_chat\_message*), 328  
`UNSPECIFIED` (*aiogram.enums.passport\_element\_error\_type.PassportElementErrorType* attribute), 386  
`UnsupportedKeywordArgument`, 437  
`until_date` (*aiogram.methods.ban\_chat\_member.BanChatMember* attribute), 239  
`until_date` (*aiogram.methods.restrict\_chat\_member.RestrictChatMember* attribute), 278  
`until_date` (*aiogram.types.chat\_member\_banned.ChatMemberBanned* attribute), 121  
`until_date` (*aiogram.types.chat\_member\_restricted.ChatMemberRestricted* attribute), 124  
`update` (*aiogram.types.error\_event.ErrorEvent* attribute), 436  
`Update` (class in *aiogram.types.update*), 225  
`update_data()` (*aiogram.fsm.storage.base.BaseStorage* method), 433  
`update_handler_flags()` (*aiogram.filters.base.Filter* method), 412  
`update_id` (*aiogram.types.update.Update* attribute), 225  
`UpdateType` (class in *aiogram.enums.update\_type*), 388  
`UpdateTypeLookupError`, 226  
`UPLOAD_DOCUMENT` (*aiogram.enums.chat\_action.ChatAction* attribute), 377  
`upload_document()` (*aiogram.utils.chat\_action.ChatActionSender* class method), 454  
`UPLOAD_PHOTO` (*aiogram.enums.chat\_action.ChatAction* attribute), 377  
`upload_photo()` (*aiogram.utils.chat\_action.ChatActionSender* class method), 454  
`UPLOAD_VIDEO` (*aiogram.enums.chat\_action.ChatAction* attribute), 377  
`upload_video()` (*aiogram.utils.chat\_action.ChatActionSender* class method), 454  
`UPLOAD_VIDEO_NOTE` (*aiogram.enums.chat\_action.ChatAction* attribute), 377  
`upload_video_note()` (*aiogram.utils.chat\_action.ChatActionSender* class method), 454  
`UPLOAD_VOICE` (*aiogram.enums.chat\_action.ChatAction* attribute), 377  
`upload_voice()` (*aiogram.utils.chat\_action.ChatActionSender* class method), 454  
`UploadStickerFile` (class in *aiogram.methods.upload\_sticker\_file*), 351  
`URL` (*aiogram.enums.message\_entity\_type.MessageEntityType* attribute), 385  
`url` (*aiogram.methods.answer\_callback\_query.AnswerCallbackQuery* attribute), 236  
`url` (*aiogram.methods.set\_webhook.SetWebhook* attribute), 359  
`url` (*aiogram.types.inline\_keyboard\_button.InlineKeyboardButton* attribute), 147  
`url` (*aiogram.types.inline\_query\_result\_article.InlineQueryResultArticle* attribute), 21  
`url` (*aiogram.types.user.User* property), 210  
`url` (*aiogram.types.web\_app\_info.WebAppInfo* attribute), 215  
`url` (*aiogram.types.webhook\_info.WebhookInfo* attribute), 226  
`url` (*aiogram.utils.callback\_answer.CallbackAnswer* property), 460  
`Url` (class in *aiogram.utils.formatting*), 465  
`URLInputFile` (class in *aiogram.types.input\_file*), 149, 392  
`USD` (*aiogram.enums.currency.Currency* attribute), 382  
`use_independent_chat_permissions` (*aiogram.methods.restrict\_chat\_member.RestrictChatMember* attribute), 278  
`use_independent_chat_permissions` (*aiogram.methods.set\_chat\_permissions.SetChatPermissions* attribute), 313  
`user` (*aiogram.types.chat\_member\_administrator.ChatMemberAdministrator* attribute), 120  
`user` (*aiogram.types.chat\_member\_banned.ChatMemberBanned* attribute), 121  
`user` (*aiogram.types.chat\_member\_left.ChatMemberLeft* attribute), 122  
`user` (*aiogram.types.chat\_member\_member.ChatMemberMember* attribute), 122  
`user` (*aiogram.types.chat\_member\_owner.ChatMemberOwner* attribute), 122  
`user` (*aiogram.types.chat\_member\_restricted.ChatMemberRestricted* attribute), 123  
`user` (*aiogram.types.game\_high\_score.GameHighScore* attribute), 236  
`user` (*aiogram.types.message\_entity.MessageEntity* attribute), 204  
`user` (*aiogram.types.poll\_answer.PollAnswer* attribute), 204

206

`user` (`aiogram.utils.web_app.WebAppInitData` attribute), 457

`User` (class in `aiogram.types.user`), 210

`user_administrator_rights` (`aiogram.types.keyboard_button_request_chat.KeyboardButtonRequestChat` attribute), 157

`user_chat_id` (`aiogram.types.chat_join_request.ChatJoinRequest` attribute), 86

`user_id` (`aiogram.methods.add_sticker_to_set.AddStickerToSet` attribute), 337

`user_id` (`aiogram.methods.approve_chat_join_request.ApproveChatJoinRequest` attribute), 238

`user_id` (`aiogram.methods.ban_chat_member.BanChatMember` attribute), 239

`user_id` (`aiogram.methods.create_new_sticker_set.CreateNewStickerSet` attribute), 338

`user_id` (`aiogram.methods.decline_chat_join_request.DeclineChatJoinRequest` attribute), 248

`user_id` (`aiogram.methods.get_chat_member.GetChatMember` attribute), 260

`user_id` (`aiogram.methods.get_game_high_scores.GetGameHighScores` attribute), 352

`user_id` (`aiogram.methods.get_user_profile_photos.GetUserProfilePhotos` attribute), 269

`user_id` (`aiogram.methods.promote_chat_member.PromoteChatMember` attribute), 274

`user_id` (`aiogram.methods.restrict_chat_member.RestrictChatMember` attribute), 278

`user_id` (`aiogram.methods.set_chat_administrator_custom_title.SetChatAdministratorCustomTitle` attribute), 310

`user_id` (`aiogram.methods.set_game_score.SetGameScore` attribute), 354

`user_id` (`aiogram.methods.set_passport_data_errors.SetPassportDataErrors` attribute), 360

`user_id` (`aiogram.methods.set_sticker_set_thumbnail.SetStickerSetThumbnail` attribute), 349

`user_id` (`aiogram.methods.unban_chat_member.UnbanChatMember` attribute), 322

`user_id` (`aiogram.methods.upload_sticker_file.UploadStickerFile` attribute), 351

`user_id` (`aiogram.types.bot_command_scope_chat_member.BotCommandScopeChatMember` attribute), 68

`user_id` (`aiogram.types.contact.Contact` attribute), 143

`user_id` (`aiogram.types.user_shared.UserShared` attribute), 211

`user_is_bot` (`aiogram.types.keyboard_button_request_user.KeyboardButtonRequestUser` attribute), 157

`user_is_premium` (`aiogram.types.keyboard_button_request_user.KeyboardButtonRequestUser` attribute), 157

`USER_SHARED` (`aiogram.enums.content_type.ContentType` attribute), 379

`user_shared` (`aiogram.types.message.Message` attribute), 165

`username` (`aiogram.types.chat.Chat` attribute), 71

`username` (`aiogram.types.user.User` attribute), 210

`username` (`aiogram.utils.web_app.WebAppUser` attribute), 457

`UserProfilePhotos` (class in `aiogram.types.user_profile_photos`), 211

`users` (`aiogram.types.video_chat_participants_invited.VideoChatParticipantsInvited` attribute), 383

`UserShared` (class in `aiogram.types.user_shared`), 211

`UTILITY_BILL` (`aiogram.enums.encrypted_passport_element.EncryptedPassportElement` attribute), 383

`VIDEO_CHAT_JOIN_REQUEST` (`aiogram.enums.content_type.ContentType` attribute), 382

`UZS` (`aiogram.enums.currency.Currency` attribute), 382

## V

`value` (`aiogram.types.dice.Dice` attribute), 143

`vcard` (`aiogram.methods.send_contact.SendContact` attribute), 186

`vcard` (`aiogram.types.contact.Contact` attribute), 143

`vcard` (`aiogram.types.inline_query_result_contact.InlineQueryResultContact` attribute), 40

`vcard` (`aiogram.types.input_contact_message_content.InputContactMessageContent` attribute), 58

`VIDEO` (`aiogram.enums.content_type.ContentType` attribute), 379

`VIDEO_CHAT_ENDED` (`aiogram.enums.inline_query_result_type.InlineQueryResultType` attribute), 384

`venue` (`aiogram.types.message.Message` attribute), 164

`Venue` (class in `aiogram.types.venue`), 212

`via_chat_folder_invite_link` (`aiogram.types.chat_member_updated.ChatMemberUpdated` attribute), 125

`VIDEO_CHAT_PARTICIPANTS_INVITED` (`aiogram.enums.content_type.ContentType` attribute), 378

`VIDEO_CHAT_ENDED` (`aiogram.enums.inline_query_result_type.InlineQueryResultType` attribute), 384

`VIDEO_CHAT_PARTICIPANTS_INVITED` (`aiogram.enums.input_media_type.InputMediaType` attribute), 384

`VIDEO_CHAT_PARTICIPANTS_INVITED` (`aiogram.enums.sticker_format.StickerFormat` attribute), 387

`video` (`aiogram.methods.send_video.SendVideo` attribute), 303

`video` (`aiogram.types.message.Message` attribute), 163

`Video` (class in `aiogram.types.video`), 212

`VIDEO_CHAT_ENDED` (`aiogram.enums.content_type.ContentType` attribute), 380

`video_chat_participants_invited` (`aiogram.types.message.Message` attribute), 380

- 166  
**VIDEO\_CHAT\_SCHEDULED** (*aiogram.enums.content\_type.ContentType* attribute), 379  
**video\_chat\_scheduled** (*aiogram.types.message.Message* attribute), 165  
**VIDEO\_CHAT\_STARTED** (*aiogram.enums.content\_type.ContentType* attribute), 379  
**video\_chat\_started** (*aiogram.types.message.Message* attribute), 165  
**video\_duration** (*aiogram.types.inline\_query\_result\_video.InlineQueryResultVideo* attribute), 55  
**video\_file\_id** (*aiogram.types.inline\_query\_result\_cached\_video.InlineQueryResultCachedVideo* attribute), 37  
**video\_height** (*aiogram.types.inline\_query\_result\_video.InlineQueryResultVideo* attribute), 55  
**VIDEO\_NOTE** (*aiogram.enums.content\_type.ContentType* attribute), 378  
**video\_note** (*aiogram.methods.send\_video\_note.SendVideoNote* attribute), 306  
**video\_note** (*aiogram.types.message.Message* attribute), 163  
**video\_url** (*aiogram.types.inline\_query\_result\_video.InlineQueryResultVideo* attribute), 54  
**video\_width** (*aiogram.types.inline\_query\_result\_video.InlineQueryResultVideo* attribute), 55  
**VideoChatEnded** (class in *aiogram.types.video\_chat\_ended*), 213  
**VideoChatParticipantsInvited** (class in *aiogram.types.video\_chat\_participants\_invited*), 213  
**VideoChatScheduled** (class in *aiogram.types.video\_chat\_scheduled*), 213  
**VideoChatStarted** (class in *aiogram.types.video\_chat\_started*), 214  
**VideoNote** (class in *aiogram.types.video\_note*), 214  
**VIOLET** (*aiogram.enums.topic\_icon\_color.TopicIconColor* attribute), 387  
**VND** (*aiogram.enums.currency.Currency* attribute), 382  
**VOICE** (*aiogram.enums.content\_type.ContentType* attribute), 378  
**VOICE** (*aiogram.enums.inline\_query\_result\_type.InlineQueryResultType* attribute), 384  
**voice** (*aiogram.methods.send\_voice.SendVoice* attribute), 308  
**voice** (*aiogram.types.message.Message* attribute), 163  
**Voice** (class in *aiogram.types.voice*), 214  
**voice\_duration** (*aiogram.types.inline\_query\_result\_voice.InlineQueryResultVoice* attribute), 57  
**voice\_file\_id** (*aiogram.types.inline\_query\_result\_cached\_voice.InlineQueryResultCachedVoice* attribute), 39  
**voice\_url** (*aiogram.types.inline\_query\_result\_voice.InlineQueryResultVoice* attribute), 56  
**voter\_chat** (*aiogram.types.poll\_answer.PollAnswer* attribute), 206  
**voter\_count** (*aiogram.types.poll\_option.PollOption* attribute), 206
- ## W
- watcher** (*aiogram.types.proximity\_alert\_triggered.ProximityAlertTriggered* attribute), 207  
**WEB\_APP** (*aiogram.enums.menu\_button\_type.MenuButtonType* attribute), 385  
**web\_app** (*aiogram.types.inline\_keyboard\_button.InlineKeyboardButton* attribute), 159  
**web\_app** (*aiogram.types.inline\_query\_results\_button.InlineQueryResultsButton* attribute), 159  
**web\_app** (*aiogram.types.keyboard\_button.KeyboardButton* attribute), 159  
**web\_app** (*aiogram.types.menu\_button.MenuButton* attribute), 159  
**web\_app** (*aiogram.types.menu\_button\_web\_app.MenuButtonWebApp* attribute), 160  
**WEB\_APP\_DATA** (*aiogram.enums.content\_type.ContentType* attribute), 380  
**web\_app\_data** (*aiogram.types.message.Message* attribute), 166  
**web\_app\_name** (*aiogram.types.write\_access\_allowed.WriteAccessAllowed* attribute), 215  
**web\_app\_query\_id** (*aiogram.methods.answer\_web\_app\_query.AnswerWebAppQuery* attribute), 375  
**WebAppData** (class in *aiogram.types.web\_app\_data*), 215  
**WebAppInfo** (class in *aiogram.types.web\_app\_info*), 215  
**WebAppInitData** (class in *aiogram.utils.web\_app*), 456  
**WebAppUser** (class in *aiogram.utils.web\_app*), 457  
**WebhookInfo** (class in *aiogram.types.webhook\_info*), 226  
**width** (*aiogram.methods.send\_animation.SendAnimation* attribute), 281  
**width** (*aiogram.methods.send\_video.SendVideo* attribute), 304  
**width** (*aiogram.types.animation.Animation* attribute), 64  
**width** (*aiogram.types.input\_media\_animation.InputMediaAnimation* attribute), 150  
**width** (*aiogram.types.input\_media\_video.InputMediaVideo* attribute), 154  
**width** (*aiogram.types.photo\_size.PhotoSize* attribute), 205  
**width** (*aiogram.types.sticker.Sticker* attribute), 228  
**width** (*aiogram.types.video.Video* attribute), 212  
**wrap\_local\_file** (*aiogram.client.telegram.TelegramAPIServer* attribute), 215  
**WRITE\_ACCESS\_ALLOWED** (*aiogram.enums.content\_type.ContentType* attribute), 379  
**write\_access\_allowed** (*aiogram.types.message.Message* attribute),

165

`WriteAccessAllowed` (class in `aiogram.types.write_access_allowed`), 215

## X

`x_shift` (`aiogram.types.mask_position.MaskPosition` attribute), 228

## Y

`y_shift` (`aiogram.types.mask_position.MaskPosition` attribute), 228

`YELLOW` (`aiogram.enums.topic_icon_color.TopicIconColor` attribute), 387

`YER` (`aiogram.enums.currency.Currency` attribute), 382

## Z

`ZAR` (`aiogram.enums.currency.Currency` attribute), 382