aiogram Documentation

Release 3.5.0

aiogram Team

Apr 24, 2024

CONTENTS

1	Featu	ires	3					
	1.1	Simple	usage					
	1.2	Usage v	vithout dispatcher					
2	Cont	ents	7					
	2.1	Installa	tion					
		2.1.1	From PyPI					
		2.1.2	From Arch Linux Repository					
		2.1.3	From PyPI					
		2.1.4	From GitHub					
	2.2	Migrati	on FAQ (2.x -> 3.0)					
		2.2.1	Dispatcher					
		2.2.2	Filtering events					
		2.2.3	Bot API					
		2.2.4	Middlewares					
		2.2.5	Keyboard Markup					
		2.2.6	Callbacks data					
		2.2.7	Finite State machine 9					
		2.2.8	Sending Files					
		2.2.9	Webhook					
		2.2.10	Telegram API Server 10					
	2.3	Bot AP	Bot API					
		2.3.1	Bot 10					
		2.3.2	Client session					
		2.3.3	Types					
		2.3.4	Methods					
		2.3.5	Enums					
		2.3.6	How to download file?					
		2.3.7	How to upload file?					
	2.4	Handlir	ng events					
		2.4.1	Router					
		2.4.2	Dispatcher					
		2.4.3	Dependency injection					
		2.4.4	Filtering events					
		2.4.5	Long-polling					
		2.4.6	Webhook					
		2.4.7	Finite State Machine					
		2.4.8	Middlewares					
		2.4.9	Errors					
		2.4.10	Flags					

		Class based handlers
2.5	Utils .	
	2.5.1	Keyboard builder
	2.5.2	Translation
	2.5.3	Chat action sender
	2.5.4	WebApp
	2.5.5	Callback answer
	2.5.6	Formatting
	2.5.7	Media group builder
	2.5.8	Deep Linking
2.6	Change	log
	2.6.1	3.5.0 (2024-04-23)
	2.6.2	3.4.1 (2024-02-17)
	2.6.3	3.4.0 (2024-02-16)
	2.6.4	3.3.0 (2023-12-31)
	2.6.5	3.2.0 (2023-11-24)
	2.6.6	3.1.1 (2023-09-25)
	2.6.7	3.1.0 (2023-09-22)
	2.6.8	3.0.0 (2023-09-01)
	2.6.9	3.0.0rc2 (2023-08-18)
	2.6.10	3.0.0rc1 (2023-08-06)
	2.6.11	3.0.0b9 (2023-07-30)
	2.6.12	3.0.0b8 (2023-07-17)
	2.6.13	3.0.0b7 (2023-02-18)
	2.6.14	3.0.0b6 (2022-11-18)
	2.6.15	3.0.0b5 (2022-10-02)
	2.6.16	3.0.0b4 (2022-08-14)
	2.6.17	3.0.0b3 (2022-04-19)
	2.6.18	3.0.0b2 (2022-02-19)
	2.6.19	3.0.0b1 (2021-12-12)
	2.6.20	3.0.0a18 (2021-11-10)
	2.6.21	3.0.0a17 (2021-09-24)
	2.6.22	3.0.0a16 (2021-09-22)
	2.6.23	3.0.0a15 (2021-09-10)
	2.6.24	3.0.0a14 (2021-08-17)
	2.6.25	2.14.3 (2021-07-21)
	2.6.26	2.14.2 (2021-07-26)
	2.6.27	
	2.6.28	2.13 (2021-04-28)
	2.6.29	2.12.1 (2021-03-22)
	2.6.30	2.12 (2021-03-14)
	2.6.31	2.11.2 (2021-11-10)
	2.6.32	2.11.1 (2021-11-10)
	2.6.33	2.11 (2021-11-08)
	2.6.34	2.10.1 (2021-09-14)
	2.6.35	2.10 (2021-09-13)
	2.6.36	2.9.2 (2021-06-13)
	2.6.37	2.9 (2021-06-08)
	2.6.38	2.8 (2021-04-26)
	2.6.39	2.7 (2021-04-07)
	2.6.40	2.6.1 (2021-01-25)
	2.6.41	2.6 (2021-01-23)
	2.6.42	2.5.3 (2021-01-05)
	2.6.43	2.5.2 (2021-01-01)

	2.6.44	2.5.1 (2021-01-01)	
	2.6.45	2.5 (2021-01-01)	
	2.6.46	2.4 (2021-10-29)	
	2.6.47	2.3 (2021-08-16)	
	2.6.48	2.2 (2021-06-09)	
	2.6.49	2.1 (2021-04-18)	
	2.6.50	2.0.1 (2021-12-31)	
	2.6.51	2.0 (2021-10-28)	. 604
	2.6.52	1.4 (2021-08-03)	
	2.6.53	1.3.3 (2021-07-16)	
	2.6.54	1.3.2 (2021-05-27)	
	2.6.55	1.3.1 (2018-05-27)	. 605
	2.6.56	1.3 (2021-04-22)	. 605
	2.6.57	1.2.3 (2018-04-14)	. 605
	2.6.58	1.2.2 (2018-04-08)	. 605
	2.6.59	1.2.1 (2018-03-25)	. 605
	2.6.60	1.2 (2018-02-23)	. 605
	2.6.61	1.1 (2018-01-27)	. 606
	2.6.62	1.0.4 (2018-01-10)	. 606
	2.6.63	1.0.3 (2018-01-07)	. 606
	2.6.64	1.0.2 (2017-11-29)	. 606
	2.6.65	1.0.1 (2017-11-21)	. 606
	2.6.66	1.0 (2017-11-19)	. 606
	2.6.67	0.4.1 (2017-08-03)	. 607
	2.6.68	0.4 (2017-08-05)	. 607
	2.6.69	0.3.4 (2017-08-04)	. 607
	2.6.70	0.3.3 (2017-07-05)	. 607
	2.6.71	0.3.2 (2017-07-04)	. 607
	2.6.72	0.3.1 (2017-07-04)	. 607
	2.6.73	0.2b1 (2017-06-00)	. 607
	2.6.74	0.1 (2017-06-03)	. 607
2.7	Contrib	puting	. 607
	2.7.1	Developing	. 607
	2.7.2	Star on GitHub	. 610
	2.7.3	Guides	. 610
	2.7.4	Take answers	
	2.7.5	Funding	. 610
Python N	Module I	Index	611

Index

615

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

CHAPTER

ONE

FEATURES

- Asynchronous (asyncio docs, PEP 492)
- Has type hints (PEP 484) and can be used with mypy
- Supports PyPy
- Supports Telegram Bot API 7.2 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, html
from aiogram.client.default import DefaultBotProperties
from aiogram.enums import ParseMode
from aiogram.filters import CommandStart
from aiogram.types import Message
# 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.
\hookrightarrow 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, {html.bold(message.from_user.full_name)}!")
@dp.message()
async def echo_handler(message: 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 default bot properties which will be passed to all_
    API calls
    bot = Bot(token=TOKEN, default=DefaultBotProperties(parse_mode=ParseMode.HTML))
    # And the run events dispatching
    await dp.start_polling(bot)
if __name__ == "__main__":
    logging.basicConfig(level=logging.INF0, stream=sys.stdout)
    asyncio.run(main())
```

1.2 Usage without dispatcher

Just only interact with Bot API, without handling events

```
import asyncio
from argparse import ArgumentParser
from aiogram import Bot
from aiogram.client.default import DefaultBotProperties
from aiogram.enums import ParseMode
def create_parser() -> ArgumentParser:
   parser = ArgumentParser()
   parser.add_argument("--token", help="Telegram Bot API Token")
   parser.add_argument("--chat-id", type=int, help="Target chat id")
   parser.add_argument("--message", "-m", help="Message text to sent", default="Hello,_
→World!")
   return parser
async def main():
   parser = create_parser()
   ns = parser.parse_args()
   token = ns.token
   chat_id = ns.chat_id
   message = ns.message
   async with Bot(
        token=token.
        default=DefaultBotProperties(
            parse_mode=ParseMode.HTML,
        ),
   ) as bot:
        await bot.send_message(chat_id=chat_id, text=message)
```

(continues on next page)

(continued from previous page)

```
if __name__ == "__main__":
    asyncio.run(main())
```

CHAPTER

TWO

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 numerous breaking changes and architectural improvements. It helps reduce the count of global variables in your code, provides useful mechanisms to modularize your code, and enables the creation of shareable modules via packages on PyPI. It also makes middlewares and filters more controllable, among other improvements.

On this page, you can read about the changes made in relation to the last stable 2.x version.

Note: This page more closely resembles 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

- The Dispatcher class no longer accepts a *Bot* instance in its initializer. Instead, the *Bot* instance should be passed to the dispatcher only for starting polling or handling events from webhooks. This approach also allows for the use of multiple bot instances simultaneously ("multibot").
- Dispatcher now can be extended with another Dispatcher-like thing named Router (Read more »).
- With routes, you can easily modularize your code and potentially share these modules between projects.
- Removed the _handler suffix from all event handler decorators and registering methods. (Read more »)
- The Executor has been entirely removed; you can now use the Dispatcher directly to start poll the API or handle webhooks from it.
- The throttling method has been completely removed; you can now use middlewares to control the execution context and implement any throttling mechanism you desire.
- Removed global context variables from the API types, Bot and Dispatcher object, From now on, if you want to access the current bot instance within handlers or filters, you should accept the argument bot: Bot and use it instead of Bot.get_current(). In middlewares, it can be accessed via data["bot"].
- To skip pending updates, you should now call the *aiogram.methods.delete_webhook.DeleteWebhook* method directly, rather than passing skip_updates=True to the start polling method.

2.2.2 Filtering events

- Keyword filters can no longer be used; use filters explicitly. (Read more »)
- Due to the removal of keyword filters, all previously enabled-by-default filters (such as state and content_type) are now disabled. You must specify them explicitly if you wish to use them. For example instead of using @dp. message_handler(content_types=ContentType.PHOTO) you should use @router.message(F.photo)
- Most common filters have been replaced with the "magic filter." (Read more »)
- By default, the message handler now receives any content type. If you want a specific one, simply add the appropriate filters (Magic or any other).
- The state filter is no longer enabled by default. This means that if you used state="*" in v2, you should not pass any state filter in v3. Conversely, if the state was not specified in v2, you will now need to specify it in v3.
- Added the possibility to register global filters for each router, which helps to reduce code repetition and provides an easier way to control the purpose of each router.

2.2.3 Bot API

- All API methods are now classes with validation, implemented via *pydantic <https://docs.pydantic.dev/>*. These API calls are also available as methods in the Bot class.
- More pre-defined Enums have been added and moved to the *aiogram.enums* sub-package. For example, the chat type enum is now aiogram.enums.ChatType instead of aiogram.types.chat.ChatType.
- The HTTP client session has been separated into a container that can be reused across different Bot instances within the application.
- API Exceptions are no longer classified by specific messages, as Telegram has no documented error codes. However, all errors are classified by HTTP status codes, and for each method, only one type of error can be associated with a given code. Therefore, in most cases, you should check only the error type (by status code) without inspecting the error message.

2.2.4 Middlewares

- Middlewares can now control an execution context, e.g., using context managers. (Read more »)
- All contextual data is now shared end-to-end between middlewares, filters, and handlers. 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 a mechanism named **flags** that helps 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 no longer have methods for extension, instead you have to use markup builders aiogram.utils.keyboard.ReplyKeyboardBuilder and aiogram.utils. keyboard.KeyboardBuilder respectively (*Read more* »)

2.2.6 Callbacks data

• The callback data factory is now strictly typed using pydantic models. (*Read more* »)

2.2.7 Finite State machine

- State filters will no longer be automatically added to all handlers; you will need to specify the state if you want to use it.
- Added the possibility to change the FSM strategy. For example, if you want to control the state for each user based on chat topics rather than the user in a chat, you can specify this in the Dispatcher.
- Now aiogram.fsm.state.State and aiogram.fsm.state.StateGroup don't have helper methods like . set(), .next(), etc.
- Instead, you should set states by passing them directly to aiogram.fsm.context.FSMContext (Read more »)
- The 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 on, you should wrap files in an InputFile object before sending them, instead of passing the IO object directly to the API method. (*Read more* »)

2.2.9 Webhook

- The aiohttp web app configuration has been simplified.
- By default, the ability to upload files has been added when you make requests in response to updates (available for webhook only).

2.2.10 Telegram API Server

- The server parameter has been moved from the Bot instance to api in BaseSession.
- The constant *aiogram.bot.api.TELEGRAM_PRODUCTION* has been moved to *aiogram.client.telegram.PRODUCTION*.

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

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, default: DefaultBotProperties | 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.
- **default** Default bot properties. 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*) \rightarrow AsyncIterator[Bot]

Generate bot context

Parameters auto_close – close session on exit

Returns

async me() $\rightarrow User$

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*) \rightarrow 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*) \rightarrow 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*) \rightarrow 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*) \rightarrow str

Generate URL for downloading files

Parameters

- token Bot token
- path file path

Returns URL

classmethod from_base(base: str, **kwargs: Any) \rightarrow 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: FilesPathWrapper = <aiogram.client.telegram.BareFilesPathWrapper 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.BareFilesPathWrapper 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*) \rightarrow Response[TelegramType]

Check response status

<code>abstract async close()</code> \rightarrow <code>None</code>

Close client session

abstract async make_request(*bot: Bot, method: TelegramMethod*[*TelegramType*], *timeout: int* | *None* = None) \rightarrow 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) \rightarrow Any
Prepare value before sendabstract async stream_content(url: str, headers: Dict[str, Any] | None = None, timeout: int = 30,
chunk_size: int = 65536, raise_for_status: bool = True) \rightarrow
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

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

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

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:

Available types

Animation

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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

Birthdate

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

day: int

Day of the user's birth; 1-31

month: int

Month of the user's birth; 1-12

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

year: int | None

Optional. Year of the user's birth

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.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(_*ModelMetaclass_context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

BotCommandScopeAllChatAdministrators

class aiogram.types.bot_command_scope_all_chat_administrators.BotCommandScopeAllChatAdministrators(*,

type:
Lit-
eral[
=
Bot-
Com
mana
Scop
**ex
tra_c
Any)

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*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

BotCommandScopeAllGroupChats

class aiogram.types.bot_command_scope_all_group_chats.BotCommandScopeAllGroupChats(*, type:

Literal[BotCommandScopeTy = Bot-Command-ScopeType.ALL_GROUP_ **extra_data: 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*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

 $model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

BotCommandScopeAllPrivateChats

class aiogram.types.bot_command_scope_all_private_chats.BotCommandScopeAllPrivateChats(*,

uscopentifitvatechats	(',
	type:
	Lit-
	eral[BotCommandSc
	=
	Bot-
	Com-
	mand-
	ScopeType.ALL_PRI
	**ex-
	tra_data:
	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*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

BotCommandScopeChat

class aiogram.types.bot_command_scope_chat.BotCommandScopeChat(*, type: Lit-

eral[BotCommandScopeType.CHAT]
= BotCommandScopeType.CHAT,
chat_id: int | str, **extra_data:
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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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:
Lit-
eral[BotComm
=
Bot-
Com-
mand-
ScopeType.CH
chat_id:
int
str,
**ex-
tra_data:
Any)

Represents the scope of bot commands, covering all administrators of a specific group or supergroup chat.

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

type: Literal[BotCommandScopeType.CHAT_ADMINISTRATORS]

Scope type, must be *chat_administrators*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

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: Lit-

eral[BotCommandScopeType.CHAT = BotCommand-ScopeType.CHAT_MEMBER, chat_id: int | str, user_id: int, **extra_data: Any)

Represents the scope of bot commands, covering a specific member of a group or supergroup chat.

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

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)

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(_*ModelMetaclass_context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

user_id: int

Unique identifier of the target user

BotCommandScopeDefault

class aiogram.types.bot_command_scope_default.BotCommandScopeDefault(*, type: Lit-

eral[BotCommandScopeType.DEFAULT]
= BotCommandScopeType.DEFAULT,
**extra_data: 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*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

BotShortDescription

This object represents the bot's short description.

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

short_description: str

The bot's short description

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(_*ModelMetaclass_context: Any*) → None

BusinessConnection

is_enabled: bool, **extra_data: Any)

Describes the connection of the bot with a business account.

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

id: str

Unique identifier of the business connection

user: User

Business account user that created the business connection

user_chat_id: int

Identifier of a private chat with the user who created the business connection. 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.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

date: DateTime

Date the connection was established in Unix time

can_reply: bool

True, if the bot can act on behalf of the business account in chats that were active in the last 24 hours

is_enabled: bool

True, if the connection is active

BusinessIntro

```
class aiogram.types.business_intro.BusinessIntro(*, title: str | None = None, message: str | None =
```

None, sticker: Sticker | *None = None, **extra_data: Any*)

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

title: str | None

Optional. Title text of the business intro

message: str | None

Optional. Message text of the business intro

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

sticker: Sticker | None

Optional. Sticker of the business intro

BusinessLocation

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

address: str

Address of the business

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

location: Location | None

Optional. Location of the business

BusinessMessagesDeleted

class aiogram.types.business_messages_deleted.BusinessMessagesDeleted(*,

business_connection_id: str, chat: Chat, message_ids: List[int], **extra_data: Any)

This object is received when messages are deleted from a connected business account.

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

business_connection_id: str

Unique identifier of the business connection

chat: Chat

Information about a chat in the business account. The bot may not have access to the chat or the corresponding user.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

message_ids: List[int]

A JSON-serialized list of identifiers of deleted messages in the chat of the business account

BusinessOpeningHours

class aiogram.types.business_opening_hours.BusinessOpeningHours(*, time_zone_name: str,

opening_hours: List[BusinessOpeningHoursInterval], **extra_data: Any)

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

time_zone_name: str

Unique name of the time zone for which the opening hours are defined

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

 $model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

opening_hours: List[BusinessOpeningHoursInterval]

List of time intervals describing business opening hours

BusinessOpeningHoursInterval

class aiogram.types.business_opening_hours_interval.BusinessOpeningHoursInterval(*, open-

ing_minute: int, closing_minute: int, **extra_data: Any)

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

opening_minute: int

The minute's sequence number in a week, starting on Monday, marking the start of the time interval during which the business is open; 0 - 7 * 24 * 60

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

closing_minute: int

The minute's sequence number in a week, starting on Monday, marking the end of the time interval during which the business is open; 0 - 8 * 24 * 60

CallbackQuery

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 | InaccessibleMessage | None

Optional. Message sent by the bot with the callback button that originated the query

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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) \rightarrow 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	<pre>aiogram.methods.answer_callback_query.</pre>	
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*, *birthdate*: Birthdate | *None* = *None*, *business_intro*: BusinessIntro | None = None, business location: BusinessLocation | None = *None*, *business_opening_hours*: BusinessOpeningHours | *None* = *None*, *personal_chat:* Chat | *None = None, available_reactions: List*[ReactionTypeEmoji] | ReactionTypeCustomEmoji] | None = None, accent color id: int | None = None, background custom emoji id: str | None = None, profile_accent_color_id: int | None = None, profile_background_custom_emoji_id: str | None = None, emoji_status_custom_emoji_id: str | None = None, emoji_status_expiration_date: 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*, *unrestrict_boost_count: 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, has_visible_history: bool | None = None, sticker_set_name: str | None = None, can_set_sticker_set: bool | None = None, custom_emoji_sticker_set_name: str | 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*.

birthdate: Birthdate | None

Optional. For private chats, the date of birth of the user. Returned only in *aiogram.methods.get_chat*. *GetChat*.

business_intro: BusinessIntro | None

Optional. For private chats with business accounts, the intro of the business. Returned only in *aiogram*. *methods.get_chat.GetChat*.

business_location: BusinessLocation | None

Optional. For private chats with business accounts, the location of the business. Returned only in *aiogram*. *methods.get_chat.GetChat*.

business_opening_hours: BusinessOpeningHours | None

Optional. For private chats with business accounts, the opening hours of the business. Returned only in *aiogram.methods.get_chat.GetChat*.

personal_chat: Chat | None

Optional. For private chats, the personal channel of the user. Returned only in *aiogram.methods*. *get_chat.GetChat*.

available_reactions: List[ReactionTypeEmoji | ReactionTypeCustomEmoji] | None

Optional. List of available reactions allowed in the chat. If omitted, then all emoji reactions are allowed. Returned only in *aiogram.methods.get_chat.GetChat.*

accent_color_id: int | None

Optional. Identifier of the accent color for the chat name and backgrounds of the chat photo, reply header, and link preview. See accent colors for more details. Returned only in *aiogram.methods.get_chat*. *GetChat*. Always returned in *aiogram.methods.get_chat.GetChat*.

background_custom_emoji_id: str | None

Optional. Custom emoji identifier of emoji chosen by the chat for the reply header and link preview background. Returned only in *aiogram.methods.get_chat.GetChat.*

profile_accent_color_id: int | None

Optional. Identifier of the accent color for the chat's profile background. See profile accent colors for more details. Returned only in *aiogram.methods.get_chat.GetChat.*

profile_background_custom_emoji_id: str | None

Optional. Custom emoji identifier of the emoji chosen by the chat for its profile background. Returned only in *aiogram.methods.get_chat.GetChat*.

emoji_status_custom_emoji_id: str | None

Optional. Custom emoji identifier of the emoji status of the chat or 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 chat or the other party in a private chat, in Unix time, 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.*

unrestrict_boost_count: int | None

Optional. For supergroups, the minimum number of boosts that a non-administrator user needs to add in order to ignore slow mode and chat permissions. 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.*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_post_init($ModelMetaclass_context: Any$) \rightarrow None

We need to both initialize private attributes and call the user-defined model_post_init method.

has_visible_history: bool | None

Optional. True, if new chat members will have access to old messages; available only to chat administrators. 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.

custom_emoji_sticker_set_name: str | None

Optional. For supergroups, the name of the group's custom emoji sticker set. Custom emoji from this set can be used by all users and bots in the group. 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 $-1 * 10^{(number_length + 2)}$ Finally, self.id is substracted 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) \rightarrow 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

 ${\tt sender_chat_id} - Unique \ identifier \ of \ the \ target \ sender \ chat$

Returns

instance of method aiogram.methods.ban_chat_sender_chat.BanChatSenderChat

unban_sender_chat(sender_chat_id: int, **kwargs: Any) \rightarrow 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

 ${\tt sender_chat_id} - {\tt Unique \ identifier \ of \ the \ target \ sender \ chat}$

Returns

instance of method aiogram.methods.unban_chat_sender_chat. UnbanChatSenderChat

$\texttt{get_administrators(**kwargs: Any)} \rightarrow \textit{GetChatAdministrators}$

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

$delete_message(message_id: int, **kwargs: Any) \rightarrow DeleteMessage$

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) \rightarrow 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

 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	<pre>aiogram.methods.edit_chat_invite_link.</pre>
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.ExportChatInvite_link.ExportChatInvite_link.ExportChatInvite_link.ExportChatInvite_link.ExportChatInvite_link.ExportChatInvite_link.ExportChatInvite_link.ExportChatInvite_link.ExportChatInvite_link.ExportChatInviteLink again.

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

Returns

instance of method aiogram.methods.export_chat_invite_link. ExportChatInviteLink

do(action: str, business_connection_id: str | None = None, message_thread_id: int | None = None, **kwargs: Any) \rightarrow 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.
- **business_connection_id** Unique identifier of the business connection on behalf of which the action will be sent
- message_thread_id Unique identifier for the target message thread; for supergroups only

Returns

instance of method aiogram.methods.send_chat_action.SendChatAction

$delete_sticker_set(**kwargs: Any) \rightarrow DeleteChatStickerSet$

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

 $set_sticker_set(sticker_set_name: str, **kwargs: Any) \rightarrow SetChatStickerSet$

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) \rightarrow 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 GetChatMemberCount aiogram.methods.get_chat_member_count.

leave(***kwargs:* Any) \rightarrow 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) \rightarrow 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

 $unpin_message(message_id: int | None = None, **kwargs: Any) \rightarrow UnpinChatMessage$

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

 $\texttt{pin_message}(\textit{message_id: int, disable_notification: bool | None = None, **kwargs: Any}) \rightarrow \texttt{message}(\textit{message_id: int, disable_notification: bool | None = None, **kwargs: Any}) \rightarrow \texttt{message}(\textit{message_id: int, disable_notification: bool | None = None, **kwargs: Any}) \rightarrow \texttt{message}(\textit{message_id: int, disable_notification: bool | None = None, **kwargs: Any}) \rightarrow \texttt{message}(\textit{message_id: int, disable_notification: bool | None = None, **kwargs: Any}) \rightarrow \texttt{message}(\textit{message_id: int, disable_notification: bool | None = None, **kwargs: Any}) \rightarrow \texttt{message}(\textit{message_id: int, disable_notification: bool | None = None, **kwargs: Any}) \rightarrow \texttt{message}(\textit{message_id: int, disable_notification: bool | None = None, **kwargs: Any}) \rightarrow \texttt{message}(\textit{message_id: int, disable_notification: bool | None = None, **kwargs: Any}) \rightarrow \texttt{message}(\textit{message_id: int, disable_notification: bool | None = None, **kwargs: Any}) \rightarrow \texttt{message}(\textit{message_id: int, disable_notification: bool | None = None, **kwargs: Any}) \rightarrow \texttt{message_id: int, disable_notification: bool | None = None, **kwargs: Any})$

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) \rightarrow 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

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

in_restrict_members. boot | None = None, cut_promote_members. boot | None = None,

can_change_info: bool | None = None, can_invite_users: bool | None = None, can_post_stories: bool | None = None, can_edit_stories: bool | None = None, can_delete_stories: bool | None = None,

can post messages: bool | None = None, can edit messages: bool | None = None,

*can_pin_messages: bool | None = None, can_manage_topics: bool | None = None, **kwargs: Any)* \rightarrow *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

- 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, get boost list, see hidden supergroup and channel members, report spam messages and ignore slow mode. Implied by any other administrator privilege.
- 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, or access supergroup statistics
- **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_post_stories Pass True if the administrator can post stories to the chat
- can_edit_stories Pass True if the administrator can edit stories posted by other users
- can_delete_stories Pass True if the administrator can delete stories posted by other users
- **can_post_messages** Pass True if the administrator can post messages in the channel, or access channel statistics; for channels only
- **can_edit_messages** Pass True if the administrator can edit messages of other users and can pin messages; for channels only
- can_pin_messages Pass True if the administrator can pin messages; for supergroups only
- **can_manage_topics** Pass True if the user is allowed to create, rename, close, and reopen forum topics; for supergroups only

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*) \rightarrow *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*) \rightarrow *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*) \rightarrow *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*) \rightarrow *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*) \rightarrow *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

 $delete_photo(**kwargs: Any) \rightarrow DeleteChatPhoto$

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*) \rightarrow *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) \rightarrow 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_stories: bool, can_edit_stories: bool, can_delete_stories: bool. can_post_messages: $bool \mid None = None,$ can edit messages: bool |None = None,can pin messages: bool | None = None,can_manage_topics: $bool \mid None = None,$ **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, get boost list, see hidden supergroup and channel members, report spam messages 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, or access supergroup statistics

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

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

 $model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

can_post_stories: bool

True, if the administrator can post stories to the chat

can_edit_stories: bool

True, if the administrator can edit stories posted by other users

can_delete_stories: bool

True, if the administrator can delete stories posted by other users

can_post_messages: bool | None

Optional. True, if the administrator can post messages in the channel, or access channel statistics; for channels only

can_edit_messages: bool | None

Optional. True, if the administrator can edit messages of other users and can pin messages; for channels only

can_pin_messages: bool | None

Optional. True, if the user is allowed to pin messages; for groups and supergroups only

can_manage_topics: bool | None

Optional. True, if the user is allowed to create, rename, close, and reopen forum topics; for supergroups only

ChatBoost

This object contains information about a chat boost.

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

boost_id: str

Unique identifier of the boost

add_date: DateTime

Point in time (Unix timestamp) when the chat was boosted

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

expiration_date: DateTime

Point in time (Unix timestamp) when the boost will automatically expire, unless the booster's Telegram Premium subscription is prolonged

source: ChatBoostSourcePremium | ChatBoostSourceGiftCode | ChatBoostSourceGiveaway Source of the added boost

ChatBoostAdded

class aiogram.types.chat_boost_added.**ChatBoostAdded**(*, *boost_count: int*, ***extra_data: Any*) This object represents a service message about a user boosting a chat.

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

boost_count: int

Number of boosts added by the user

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

ChatBoostRemoved

This object represents a boost removed from a chat.

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

chat: Chat

Chat which was boosted

boost_id: str

Unique identifier of the boost

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

We need to both initialize private attributes and call the user-defined model_post_init method.

remove_date: DateTime

Point in time (Unix timestamp) when the boost was removed

source: ChatBoostSourcePremium | ChatBoostSourceGiftCode | ChatBoostSourceGiveaway
Source of the removed boost

ChatBoostSourceGiveaway, **extra_data:

Any)

ChatBoostSource

class aiogram.types.chat_boost_source.ChatBoostSource(**extra_data: Any)

This object describes the source of a chat boost. It can be one of

- aiogram.types.chat_boost_source_premium.ChatBoostSourcePremium
- aiogram.types.chat_boost_source_gift_code.ChatBoostSourceGiftCode
- aiogram.types.chat_boost_source_giveaway.ChatBoostSourceGiveaway

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(<u>ModelMetaclass_context</u>: Any) \rightarrow None We need to both initialize private attributes and call the user-defined model_post_init method.

ChatBoostSourceGiftCode

class aiogram.types.chat_boost_source_gift_code.ChatBoostSourceGiftCode(*, source: Lit-

eral[ChatBoostSourceType.GIFT_CODE = ChatBoostSource-Type.GIFT_CODE, user: User, **extra_data: Any)

The boost was obtained by the creation of Telegram Premium gift codes to boost a chat. Each such code boosts the chat 4 times for the duration of the corresponding Telegram Premium subscription.

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

source: Literal[ChatBoostSourceType.GIFT_CODE]

Source of the boost, always 'gift_code'

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

user: User

User for which the gift code was created

ChatBoostSourceGiveaway

class aiogram.types.chat_boost_source_giveaway.ChatBoostSourceGiveaway(*, source: Lit-

eral[ChatBoostSourceType.GIVEAWAY]
= ChatBoostSourceType.GIVEAWAY,
giveaway_message_id:
int, user: User | None =
None, is_unclaimed:
bool | None = None,
**extra_data: Any)

The boost was obtained by the creation of a Telegram Premium giveaway. This boosts the chat 4 times for the duration of the corresponding Telegram Premium subscription.

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

source: Literal[ChatBoostSourceType.GIVEAWAY]

Source of the boost, always 'giveaway'

giveaway_message_id: int

Identifier of a message in the chat with the giveaway; the message could have been deleted already. May be 0 if the message isn't sent yet.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

user: User | None

Optional. User that won the prize in the giveaway if any

is_unclaimed: bool | None

Optional. True, if the giveaway was completed, but there was no user to win the prize

ChatBoostSourcePremium

```
class aiogram.types.chat_boost_source_premium.ChatBoostSourcePremium(*, source: Lit-
```

eral[ChatBoostSourceType.PREMIUM] = ChatBoostSource-Type.PREMIUM, user: User, **extra_data: Any)

The boost was obtained by subscribing to Telegram Premium or by gifting a Telegram Premium subscription to another user.

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

source: Literal[ChatBoostSourceType.PREMIUM]

Source of the boost, always 'premium'

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

user: User

User that boosted the chat

ChatBoostUpdated

This object represents a boost added to a chat or changed.

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

chat: Chat

Chat which was boosted

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

```
model_post_init(_ModelMetaclass__context: Any) → None
```

We need to both initialize private attributes and call the user-defined model_post_init method.

boost: ChatBoost

Information about the chat boost

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 | 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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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, 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 5 minutes 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) \rightarrow 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

instance of method aiogram.methods.approve_chat_join_request. ApproveChatJoinRequest

decline(**kwargs: Any) \rightarrow 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, business_connection_id: Optional[str] = None, message_thread_id: Optional[int] = None, parse_mode: Optional[Union[str, Default]] = <Default('parse_mode')>, entities: Optional[List[MessageEntity]] = None, link_preview_options: Optional[Union[LinkPreviewOptions, Default]] = <Default('link_preview')>, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, disable_web_page_preview: Optional[Union[bool, Default]] = <Default('link_preview_is_disabled')>, reply_to_message_id: Optional[int] = 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

- text Text of the message to be sent, 1-4096 characters after entities parsing
- **business_connection_id** Unique identifier of the business connection on behalf of which the message will be sent
- **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*
- link_preview_options Link preview generation options for the 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_parameters** Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- allow_sending_without_reply Pass True if the message should be sent even if the specified replied-to message is not found
- disable_web_page_preview Disables link previews for links in this message
- reply_to_message_id If the message is a reply, ID of the original message

instance of method aiogram.methods.send_message.SendMessage

answer_pm(text: str, business_connection_id: Optional[str] = None, message_thread_id: Optional[int] =
 None, parse_mode: Optional[Union[str, Default]] = <Default('parse_mode')>, entities:
 Optional[List[MessageEntity]] = None, link_preview_options:
 Optional[Union[LinkPreviewOptions, Default]] = <Default('link_preview')>,
 disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] =
 <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None,
 reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup,
 ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] =
 None, disable_web_page_preview: Optional[Union[bool, Default]] =
 <Default('link_preview_is_disabled')>, reply_to_message_id: Optional[int] = 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

- text Text of the message to be sent, 1-4096 characters after entities parsing
- **business_connection_id** Unique identifier of the business connection on behalf of which the message will be sent
- **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*
- link_preview_options Link preview generation options for the 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_parameters Description of the message to reply to

- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- **allow_sending_without_reply** Pass True if the message should be sent even if the specified replied-to message is not found
- disable_web_page_preview Disables link previews for links in this message
- reply_to_message_id If the message is a reply, ID of the original message

instance of method aiogram.methods.send_message.SendMessage

answer_animation(animation: Union[InputFile, str], business_connection_id: Optional[str] = None, message_thread_id: Optional[int] = None, duration: Optional[int] = None, width: Optional[int] = None, height: Optional[int] = None, thumbnail: Optional[InputFile] = None, caption: Optional[str] = None, parse_mode: Optional[Union[str, Default]] = <Default('parse_mode')>, caption_entities: Optional[List[MessageEntity]] = None, has_spoiler: Optional[bool] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = 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

- 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* »
- **business_connection_id** Unique identifier of the business connection on behalf of which the message will be sent
- **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_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- allow_sending_without_reply Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

Returns

instance of method aiogram.methods.send_animation.SendAnimation

answer_animation_pm(animation: Union[InputFile, str], business_connection_id: Optional[str] = None, message_thread_id: Optional[int] = None, duration: Optional[int] = None, width: Optional[int] = None, height: Optional[int] = None, thumbnail: Optional[InputFile] = None, caption: Optional[str] = None, parse_mode: Optional[Union[str, Default]] = <Default('parse_mode')>, caption_entities: Optional[List[MessageEntity]] = None, has_spoiler: Optional[bool] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = 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 »*

- **business_connection_id** Unique identifier of the business connection on behalf of which the message will be sent
- **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_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- allow_sending_without_reply Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

instance of method aiogram.methods.send_animation.SendAnimation

answer_audio(audio: Union[InputFile, str], business_connection_id: Optional[str] = None, message_thread_id: Optional[int] = None, caption: Optional[str] = None, parse_mode: Optional[Union[str, Default]] = <Default('parse_mode')>, caption_entities: Optional[List[MessageEntity]] = None, duration: Optional[int] = None, performer: Optional[str] = None, title: Optional[str] = None, thumbnail: Optional[InputFile] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = 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* »
- **business_connection_id** Unique identifier of the business connection on behalf of which the message will be sent
- **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_parameters** Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- allow_sending_without_reply Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

Returns

instance of method aiogram.methods.send_audio.SendAudio

answer_audio_pm(audio: Union[InputFile, str], business_connection_id: Optional[str] = None, message_thread_id: Optional[int] = None, caption: Optional[str] = None, parse_mode: Optional[Union[str, Default]] = <Default('parse_mode')>, caption_entities: Optional[List[MessageEntity]] = None, duration: Optional[int] = None, performer: Optional[str] = None, title: Optional[str] = None, thumbnail: Optional[InputFile] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = 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

- **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* »
- **business_connection_id** Unique identifier of the business connection on behalf of which the message will be sent
- **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_parameters** Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- allow_sending_without_reply Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

instance of method aiogram.methods.send_audio.SendAudio

answer_contact (phone_number: str, first_name: str, business_connection_id: Optional[str] = None, message_thread_id: Optional[int] = None, last_name: Optional[str] = None, vcard: Optional[str] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = 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

- phone_number Contact's phone number
- **first_name** Contact's first name
- **business_connection_id** Unique identifier of the business connection on behalf of which the message will be sent
- **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_parameters** Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- **allow_sending_without_reply** Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

instance of method aiogram.methods.send_contact.SendContact

answer_contact_pm(phone_number: str, first_name: str, business_connection_id: Optional[str] = None, message_thread_id: Optional[int] = None, last_name: Optional[str] = None, vcard: Optional[str] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = 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
- **business_connection_id** Unique identifier of the business connection on behalf of which the message will be sent
- **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_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- allow_sending_without_reply Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

Returns

instance of method aiogram.methods.send_contact.SendContact

answer_document(document: Union[InputFile, str], business_connection_id: Optional[str] = None, message_thread_id: Optional[int] = None, thumbnail: Optional[InputFile] = None, caption: Optional[str] = None, parse_mode: Optional[Union[str, Default]] = <Default('parse_mode')>, caption_entities: Optional[List[MessageEntity]] = None, disable_content_type_detection: Optional[bool] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = 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

- 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* »
- **business_connection_id** Unique identifier of the business connection on behalf of which the message will be sent
- **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_parameters Description of the message to reply to

- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- **allow_sending_without_reply** Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

instance of method aiogram.methods.send_document.SendDocument

answer_document_pm(document: Union[InputFile, str], business_connection_id: Optional[str] = None, message_thread_id: Optional[int] = None, thumbnail: Optional[InputFile] = None, caption: Optional[str] = None, parse_mode: Optional[Union[str, Default]] = <Default('parse_mode')>, caption_entities: Optional[List[MessageEntity]] = None, disable_content_type_detection: Optional[bool] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = 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

- 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* »
- **business_connection_id** Unique identifier of the business connection on behalf of which the message will be sent
- **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_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- **allow_sending_without_reply** Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

instance of method aiogram.methods.send_document.SendDocument

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

- game_short_name Short name of the game, serves as the unique identifier for the game. Set up your games via @BotFather.
- **business_connection_id** Unique identifier of the business connection on behalf of which the message will be sent
- **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_parameters Description of the message to reply to
- **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. Not supported for messages sent on behalf of a business account.
- **allow_sending_without_reply** Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

instance of method aiogram.methods.send_game.SendGame

answer_game_pm(game_short_name: str, business_connection_id: Optional[str] = None,

 $message_thread_id: Optional[int] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[InlineKeyboardMarkup] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = None, **kwargs: Any) \rightarrow SendGame$

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.
- **business_connection_id** Unique identifier of the business connection on behalf of which the message will be sent
- **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_parameters Description of the message to reply to
- **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. Not supported for messages sent on behalf of a business account.
- allow_sending_without_reply Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

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: Optional[int] = None, max_tip_amount: Optional[int] = None, suggested_tip_amounts: Optional[List[int]] = None, start_parameter: Optional[str] = None, provider_data: Optional[str] = None, photo_url: Optional[str] = None, photo_size: Optional[int] = None, photo_width: Optional[int] = None, photo_height: Optional[int] = None, need_name: Optional[bool] = None, need_phone_number: Optional[bool] = None, need_email: Optional[bool] = None, need_shipping_address: Optional[bool] = None, send_phone_number_to_provider: Optional[bool] = None, send_email_to_provider: Optional[bool] = None, is_flexible: Optional[bool] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[InlineKeyboardMarkup] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = 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

- 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_parameters Description of the message to reply to
- **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.
- allow_sending_without_reply Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

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: Optional[int] = None, max_tip_amount: Optional[int] = None, suggested_tip_amounts: Optional[List[int]] = None, start_parameter: Optional[str] = None, provider_data: Optional[str] = None, photo_url: Optional[str] = None, photo_size: Optional[int] = None, photo_width: Optional[int] = None, photo_height: Optional[int] = None, need_name: Optional[bool] = None, need_phone_number: Optional[bool] = None, need_email: Optional[bool] = None, need_shipping_address: Optional[bool] = None, send_phone_number_to_provider: Optional[bool] = None, send_email_to_provider: Optional[bool] = None, is_flexible: Optional[bool] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[InlineKeyboardMarkup] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = 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

- 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_parameters Description of the message to reply to

- **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.
- allow_sending_without_reply Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

instance of method aiogram.methods.send_invoice.SendInvoice

answer_location(latitude: float, longitude: float, business_connection_id: Optional[str] = None, message_thread_id: Optional[int] = None, horizontal_accuracy: Optional[float] = None, live_period: Optional[int] = None, heading: Optional[int] = None, proximity_alert_radius: Optional[int] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = 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

- **latitude** Latitude of the location
- longitude Longitude of the location
- **business_connection_id** Unique identifier of the business connection on behalf of which the message will be sent
- **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_parameters Description of the message to reply to

- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- **allow_sending_without_reply** Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

instance of method aiogram.methods.send_location.SendLocation

answer_location_pm(latitude: float, longitude: float, business_connection_id: Optional[str] = None, message_thread_id: Optional[int] = None, horizontal_accuracy: Optional[float] = None, live_period: Optional[int] = None, heading: Optional[int] = None, proximity_alert_radius: Optional[int] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = 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

- latitude Latitude of the location
- longitude Longitude of the location
- **business_connection_id** Unique identifier of the business connection on behalf of which the message will be sent
- **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_parameters** Description of the message to reply to

- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- allow_sending_without_reply Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

instance of method aiogram.methods.send_location.SendLocation

answer_media_group(media: List[Union[InputMediaAudio, InputMediaDocument, InputMediaPhoto, InputMediaVideo]], business_connection_id: Optional[str] = None, message_thread_id: Optional[int] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = 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
- business_connection_id Unique identifier of the business connection on behalf of which the message will be sent
- **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_parameters Description of the message to reply to
- allow_sending_without_reply Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the messages are a reply, ID of the original message

Returns

instance of method aiogram.methods.send_media_group.SendMediaGroup

answer_media_group_pm(media: List[Union[InputMediaAudio, InputMediaDocument, InputMediaPhoto, InputMediaVideo]], business_connection_id: Optional[str] = None, message_thread_id: Optional[int] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = 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
- **business_connection_id** Unique identifier of the business connection on behalf of which the message will be sent
- **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_parameters Description of the message to reply to
- **allow_sending_without_reply** Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the messages are a reply, ID of the original message

Returns

instance of method aiogram.methods.send_media_group.SendMediaGroup

answer_photo(*photo: Union*[InputFile, str], business_connection_id: Optional[str] = None,

$$\begin{split} message_thread_id: \ Optional[int] = None, \ caption: \ Optional[str] = None, \ parse_mode: \\ Optional[Union[str, Default]] = <Default('parse_mode')>, \ caption_entities: \\ Optional[List[MessageEntity]] = None, \ has_spoiler: \ Optional[bool] = None, \\ disable_notification: \ Optional[bool] = None, \ protect_content: \ Optional[Union[bool, \\ Default]] = <Default('protect_content')>, \ reply_parameters: \ Optional[ReplyParameters] = \\ None, \ reply_markup: \ Optional[Union[InlineKeyboardMarkup, \ ReplyKeyboardMarkup, \\ ReplyKeyboardRemove, \ ForceReply]] = None, \ allow_sending_without_reply: \ Optional[bool] \\ = None, \ reply_to_message_id: \ Optional[int] = None, \ **kwargs: \ Any) \rightarrow \ SendPhoto \end{split}$$

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* »
- business_connection_id Unique identifier of the business connection on behalf of which the message will be sent
- **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_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- **allow_sending_without_reply** Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

Returns

instance of method aiogram.methods.send_photo.SendPhoto

answer_photo_pm(photo: Union[InputFile, str], business_connection_id: Optional[str] = None,

message_thread_id: Optional[int] = None, caption: Optional[str] = None, parse_mode: Optional[Union[str, Default]] = <Default('parse_mode')>, caption_entities: Optional[List[MessageEntity]] = None, has_spoiler: Optional[bool] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = 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* »
- **business_connection_id** Unique identifier of the business connection on behalf of which the message will be sent
- **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_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- **allow_sending_without_reply** Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

Returns

instance of method aiogram.methods.send_photo.SendPhoto

answer_poll(question: str, options: List[str], business_connection_id: Optional[str] = None,

message_thread_id: Optional[int] = None, is_anonymous: Optional[bool] = None, type: Optional[str] = None, allows_multiple_answers: Optional[bool] = None, correct_option_id: Optional[int] = None, explanation: Optional[str] = None, explanation_parse_mode: Optional[Union[str, Default]] = <Default('parse_mode')>, explanation_entities: Optional[List[MessageEntity]] = None, open_period: Optional[int] = None, close_date: Optional[Union[datetime.datetime, datetime.timedelta, int]] = None, is_closed: Optional[bool] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = 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
- **business_connection_id** Unique identifier of the business connection on behalf of which the message will be sent
- **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_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- allow_sending_without_reply Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

Returns

instance of method aiogram.methods.send_poll.SendPoll

answer_poll_pm(question: str, options: List[str], business_connection_id: Optional[str] = None, message_thread_id: Optional[int] = None, is_anonymous: Optional[bool] = None, type: Optional[str] = None, allows_multiple_answers: Optional[bool] = None, correct_option_id: Optional[int] = None, explanation: Optional[str] = None, explanation_parse_mode: Optional[Union[str, Default]] = <Default('parse_mode')>, explanation_entities: Optional[List[MessageEntity]] = None, open_period: Optional[int] = None, close_date: Optional[Union[datetime.datetime, datetime.timedelta, int]] = None, is_closed: Optional[bool] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = 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

- question Poll question, 1-300 characters
- options A JSON-serialized list of answer options, 2-10 strings 1-100 characters each
- **business_connection_id** Unique identifier of the business connection on behalf of which the message will be sent
- **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_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- **allow_sending_without_reply** Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

instance of method aiogram.methods.send_poll.SendPoll

answer_dice(business_connection_id: Optional[str] = None, message_thread_id: Optional[int] = None, emoji: Optional[str] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = 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

- **business_connection_id** Unique identifier of the business connection on behalf of which the message will be sent
- **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_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- **allow_sending_without_reply** Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

instance of method aiogram.methods.send_dice.SendDice

answer_dice_pm(business_connection_id: Optional[str] = None, message_thread_id: Optional[int] = None, emoji: Optional[str] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None,

 $reply_to_message_id: Optional[int] = None, **kwargs: Any) \rightarrow SendDice$ 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

- **business_connection_id** Unique identifier of the business connection on behalf of which the message will be sent
- **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_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- **allow_sending_without_reply** Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

Returns

instance of method aiogram.methods.send_dice.SendDice

answer_sticker (sticker: Union[InputFile, str], business_connection_id: Optional[str] = None, message_thread_id: Optional[int] = None, emoji: Optional[str] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = 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, .TGS, or .WEBM sticker using multipart/formdata. *More information on Sending Files* ». Video and animated stickers can't be sent via an HTTP URL.
- **business_connection_id** Unique identifier of the business connection on behalf of which the message will be sent
- **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_parameters Description of the message to reply to
- **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. Not supported for messages sent on behalf of a business account.
- **allow_sending_without_reply** Pass True if the message should be sent even if the specified replied-to message is not found
- **reply_to_message_id** If the message is a reply, ID of the original message

Returns

instance of method aiogram.methods.send_sticker.SendSticker

answer_sticker_pm(sticker: Union[InputFile, str], business_connection_id: Optional[str] = None, message_thread_id: Optional[int] = None, emoji: Optional[str] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = 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, .TGS, or .WEBM sticker using multipart/formdata. *More information on Sending Files* ». Video and animated stickers can't be sent via an HTTP URL.

- **business_connection_id** Unique identifier of the business connection on behalf of which the message will be sent
- **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_parameters Description of the message to reply to
- **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. Not supported for messages sent on behalf of a business account.
- allow_sending_without_reply Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

Returns

instance of method aiogram.methods.send_sticker.SendSticker

answer_venue(latitude: float, longitude: float, title: str, address: str, business_connection_id: Optional[str] = None, message_thread_id: Optional[int] = None, foursquare_id: Optional[str] = None, foursquare_type: Optional[str] = None, google_place_id: Optional[str] = None, google_place_type: Optional[str] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = 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

- latitude Latitude of the venue
- longitude Longitude of the venue
- title Name of the venue
- **address** Address of the venue
- **business_connection_id** Unique identifier of the business connection on behalf of which the message will be sent

- **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_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- allow_sending_without_reply Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

instance of method aiogram.methods.send_venue.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

- **latitude** Latitude of the venue
- longitude Longitude of the venue
- title Name of the venue
- address Address of the venue
- **business_connection_id** Unique identifier of the business connection on behalf of which the message will be sent

- **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_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- allow_sending_without_reply Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

instance of method aiogram.methods.send_venue.SendVenue

answer_video(video: Union[InputFile, str], business_connection_id: Optional[str] = None,

message_thread_id: Optional[int] = None, duration: Optional[int] = None, width: Optional[int] = None, height: Optional[int] = None, thumbnail: Optional[InputFile] = None, caption: Optional[str] = None, parse_mode: Optional[Union[str, Default]] = <Default('parse_mode')>, caption_entities: Optional[List[MessageEntity]] = None, has_spoiler: Optional[bool] = None, supports_streaming: Optional[bool] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = 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

- 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* »
- **business_connection_id** Unique identifier of the business connection on behalf of which the message will be sent

- **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_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- allow_sending_without_reply Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

instance of method aiogram.methods.send_video.SendVideo

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

answer_video_pm(video: Union[InputFile, str], business_connection_id: Optional[str] = None, message_thread_id: Optional[int] = None, duration: Optional[int] = None, width: Optional[int] = None, height: Optional[int] = None, thumbnail: Optional[InputFile] = None, caption: Optional[str] = None, parse_mode: Optional[Union[str, Default]] = <Default('parse_mode')>, caption_entities: Optional[List[MessageEntity]] = None, has_spoiler: Optional[bool] = None, supports_streaming: Optional[bool] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = 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

- **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* »
- **business_connection_id** Unique identifier of the business connection on behalf of which the message will be sent
- **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_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- **allow_sending_without_reply** Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

instance of method aiogram.methods.send_video.SendVideo

answer_video_note(video_note: Union[InputFile, str], business_connection_id: Optional[str] = None, message_thread_id: Optional[int] = None, duration: Optional[int] = None, length: Optional[int] = None, thumbnail: Optional[InputFile] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = 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

- 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
- **business_connection_id** Unique identifier of the business connection on behalf of which the message will be sent
- **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_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- allow_sending_without_reply Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

Returns

instance of method aiogram.methods.send_video_note.SendVideoNote

 $\begin{aligned} & \texttt{answer_video_note_pm}(video_note: Union[InputFile, str], business_connection_id: Optional[str] = None, \\ & message_thread_id: Optional[int] = None, duration: Optional[int] = None, \\ & length: Optional[int] = None, thumbnail: Optional[InputFile] = None, \\ & disable_notification: Optional[bool] = None, protect_content: \\ & Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: \\ & Optional[ReplyParameters] = None, reply_markup: \\ & Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, \\ & ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: \\ & Optional[bool] = None, reply_to_message_id: Optional[int] = None, **kwargs: \\ & Any) \rightarrow SendVideoNote \end{aligned}$

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

- 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
- **business_connection_id** Unique identifier of the business connection on behalf of which the message will be sent
- **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_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- **allow_sending_without_reply** Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

Returns

instance of method aiogram.methods.send_video_note.SendVideoNote

answer_voice(voice: Union[InputFile, str], business_connection_id: Optional[str] = None, message_thread_id: Optional[int] = None, caption: Optional[str] = None, parse_mode: Optional[Union[str, Default]] = <Default('parse_mode')>, caption_entities: Optional[List[MessageEntity]] = None, duration: Optional[int] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = 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

- **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* »
- **business_connection_id** Unique identifier of the business connection on behalf of which the message will be sent
- **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_parameters** Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- allow_sending_without_reply Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

instance of method aiogram.methods.send_voice.SendVoice

answer_voice_pm(voice: Union[InputFile, str], business_connection_id: Optional[str] = None,

message_thread_id: Optional[int] = None, caption: Optional[str] = None, parse_mode: Optional[Union[str, Default]] = <Default('parse_mode')>, caption_entities: Optional[List[MessageEntity]] = None, duration: Optional[int] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = 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

- **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* »
- **business_connection_id** Unique identifier of the business connection on behalf of which the message will be sent
- **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_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- **allow_sending_without_reply** Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

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.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

ChatMemberAdministrator

class aiogram.types.chat_member_administrator.ChatMemberAdministrator(*, status: Lit-

eral[ChatMemberStatus.ADMINISTRATOR = ChatMemberStatus.ADMINISTRATOR. 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_stories*: bool, can_edit_stories: bool, can delete stories: bool. can post messages: $bool \mid None = None,$ can_edit_messages: bool | None = None,can_pin_messages: bool |None = None,can_manage_topics: $bool \mid None = None,$ custom_title: str | None = None, **extra_data: 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, get boost list, see hidden supergroup and channel members, report spam messages 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, or access supergroup statistics

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

can_invite_users: bool

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

can_post_stories: bool

True, if the administrator can post stories to the chat

can_edit_stories: bool

True, if the administrator can edit stories posted by other users

can_delete_stories: bool

True, if the administrator can delete stories posted by other users

can_post_messages: bool | None

Optional. True, if the administrator can post messages in the channel, or access channel statistics; for channels only

can_edit_messages: bool | None

Optional. True, if the administrator can edit messages of other users and can pin messages; for channels only

can_pin_messages: bool | None

Optional. True, if the user is allowed to pin messages; for groups and supergroups only

can_manage_topics: bool | None

Optional. True, if the user is allowed to create, rename, close, and reopen forum topics; for supergroups only

custom_title: str | None

Optional. Custom title for this user

ChatMemberBanned

class aiogram.types.chat_member_banned.ChatMemberBanned(*, status:

Literal[ChatMemberStatus.KICKED] = ChatMemberStatus.KICKED, user: User, until_date: datetime, **extra_data: 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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

until_date: DateTime

Date when restrictions will be lifted for this user; Unix time. If 0, then the user is banned forever

ChatMemberLeft

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'

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

user: User

Information about the user

ChatMemberMember

```
class aiogram.types.chat_member_member.ChatMemberMember(*, status:
```

Literal[ChatMemberStatus.MEMBER] = ChatMemberStatus.MEMBER, user: User, **extra_data: 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'

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(_*ModelMetaclass_context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

user: User

Information about the user

ChatMemberOwner

class aiogram.types.chat_member_owner.ChatMemberOwner(*, status:

Literal[ChatMemberStatus.CREATOR] = ChatMemberStatus.CREATOR, user: User, is_anonymous: bool, custom_title: str | None = None, **extra_data: 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

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

```
model_post_init(ModelMetaclass\_context: Any) \rightarrow None
```

We need to both initialize private attributes and call the user-defined model_post_init method.

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: Lit-

eral[ChatMemberStatus.RESTRICTED] = ChatMemberStatus.RESTRICTED, 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, **extra_data: 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, giveaways, giveaway winners, 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

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(_*ModelMetaclass_context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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

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, business_connection_id: Optional[str] = None, message_thread_id: Optional[int] = None,
    parse_mode: Optional[Union[str, Default]] = <Default('parse_mode')>, entities:
    Optional[List[MessageEntity]] = None, link_preview_options: Optional[Union[LinkPreviewOptions,
    Default]] = <Default('link_preview')>, disable_notification: Optional[bool] = None,
    protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters:
    Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup,
    ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply:
    Optional[bool] = None, disable_web_page_preview: Optional[Union[bool, Default]] =
    <Default('link_preview_is_disabled')>, reply_to_message_id: Optional[int] = None, **kwargs:
    Any) → SendMessage
```

Shortcut for method *aiogram.methods.send_message.SendMessage* will automatically fill method at-tributes:

chat_id

Use this method to send text messages. On success, the sent *aiogram.types.message.Message* is re-turned.

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

- text Text of the message to be sent, 1-4096 characters after entities parsing
- **business_connection_id** Unique identifier of the business connection on behalf of which the message will be sent
- **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*
- link_preview_options Link preview generation options for the 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_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- allow_sending_without_reply Pass True if the message should be sent even if the specified replied-to message is not found
- disable_web_page_preview Disables link previews for links in this message
- reply_to_message_id If the message is a reply, ID of the original message

instance of method aiogram.methods.send_message.SendMessage

answer_animation (animation: Union[InputFile, str], business_connection_id: Optional[str] = None, message_thread_id: Optional[int] = None, duration: Optional[int] = None, width: Optional[int] = None, height: Optional[int] = None, thumbnail: Optional[InputFile] = None, caption: Optional[str] = None, parse_mode: Optional[Union[str, Default]] = <Default('parse_mode')>, caption_entities: Optional[List[MessageEntity]] = None, has_spoiler: Optional[bool] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = 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

- 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 »*
- business_connection_id Unique identifier of the business connection on behalf of which the message will be sent
- **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_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- allow_sending_without_reply Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

Returns

instance of method aiogram.methods.send_animation.SendAnimation

answer_audio(audio: Union[InputFile, str], business_connection_id: Optional[str] = None, message_thread_id: Optional[int] = None, caption: Optional[str] = None, parse_mode: Optional[Union[str, Default]] = <Default('parse_mode')>, caption_entities: Optional[List[MessageEntity]] = None, duration: Optional[int] = None, performer: Optional[str] = None, title: Optional[str] = None, thumbnail: Optional[InputFile] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = 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* »

- **business_connection_id** Unique identifier of the business connection on behalf of which the message will be sent
- **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_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- allow_sending_without_reply Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

Returns

instance of method aiogram.methods.send_audio.SendAudio

answer_contact(phone_number: str, first_name: str, business_connection_id: Optional[str] = None, message_thread_id: Optional[int] = None, last_name: Optional[str] = None, vcard: Optional[str] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = 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
- **business_connection_id** Unique identifier of the business connection on behalf of which the message will be sent
- **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_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- **allow_sending_without_reply** Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

Returns

instance of method aiogram.methods.send_contact.SendContact

answer_document(document: Union[InputFile, str], business_connection_id: Optional[str] = None, message_thread_id: Optional[int] = None, thumbnail: Optional[InputFile] = None, caption: Optional[str] = None, parse_mode: Optional[Union[str, Default]] = <Default('parse_mode')>, caption_entities: Optional[List[MessageEntity]] = None, disable_content_type_detection: Optional[bool] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = 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

- **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* »
- **business_connection_id** Unique identifier of the business connection on behalf of which the message will be sent
- **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_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- **allow_sending_without_reply** Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

instance of method aiogram.methods.send_document.SendDocument

answer_game(game_short_name: str, business_connection_id: Optional[str] = None, message_thread_id: Optional[int] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[InlineKeyboardMarkup] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = 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.
- **business_connection_id** Unique identifier of the business connection on behalf of which the message will be sent
- **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_parameters Description of the message to reply to
- **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. Not supported for messages sent on behalf of a business account.
- **allow_sending_without_reply** Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

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: Optional[int] = None, max_tip_amount: Optional[int] = None, suggested_tip_amounts: Optional[List[int]] = None, start_parameter: Optional[str] = None, provider_data: Optional[str] = None, photo_url: Optional[str] = None, photo_size: Optional[int] = None, photo_width: Optional[int] = None, photo_height: Optional[int] = None, need_name: Optional[bool] = None, need_phone_number: Optional[bool] = None, need_email: Optional[bool] = None, need_shipping_address: Optional[bool] = None, send_phone_number_to_provider: Optional[bool] = None, send_email_to_provider: Optional[bool] = None, is_flexible: Optional[bool] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[InlineKeyboardMarkup] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = 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

- 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_parameters** Description of the message to reply to

- **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.
- allow_sending_without_reply Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

instance of method aiogram.methods.send_invoice.SendInvoice

answer_location(latitude: float, longitude: float, business_connection_id: Optional[str] = None, message_thread_id: Optional[int] = None, horizontal_accuracy: Optional[float] = None, live_period: Optional[int] = None, heading: Optional[int] = None, proximity_alert_radius: Optional[int] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = 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

- **latitude** Latitude of the location
- longitude Longitude of the location
- **business_connection_id** Unique identifier of the business connection on behalf of which the message will be sent
- **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_parameters Description of the message to reply to

- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- allow_sending_without_reply Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

instance of method aiogram.methods.send_location.SendLocation

answer_media_group(media: List[Union[InputMediaAudio, InputMediaDocument, InputMediaPhoto, InputMediaVideo]], business_connection_id: Optional[str] = None, message_thread_id: Optional[int] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = 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
- business_connection_id Unique identifier of the business connection on behalf of which the message will be sent
- **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_parameters Description of the message to reply to
- **allow_sending_without_reply** Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the messages are a reply, ID of the original message

Returns

instance of method aiogram.methods.send_media_group.SendMediaGroup

answer_photo(photo: Union[InputFile, str], business_connection_id: Optional[str] = None, message_thread_id: Optional[int] = None, caption: Optional[str] = None, parse_mode: Optional[Union[str, Default]] = <Default('parse_mode')>, caption_entities: Optional[List[MessageEntity]] = None, has_spoiler: Optional[bool] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = 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* »
- **business_connection_id** Unique identifier of the business connection on behalf of which the message will be sent
- **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_parameters** Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- **allow_sending_without_reply** Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

Returns

instance of method aiogram.methods.send_photo.SendPhoto

answer_poll(question: str, options: List[str], business_connection_id: Optional[str] = None, message_thread_id: Optional[int] = None, is_anonymous: Optional[bool] = None, type: Optional[str] = None, allows_multiple_answers: Optional[bool] = None, correct_option_id: Optional[int] = None, explanation: Optional[str] = None, explanation_parse_mode: Optional[Union[str, Default]] = <Default('parse_mode')>, explanation_entities: Optional[List[MessageEntity]] = None, open_period: Optional[int] = None, close_date: Optional[Union[datetime.datetime, datetime.timedelta, int]] = None, is_closed: Optional[bool] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = 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

- question Poll question, 1-300 characters
- options A JSON-serialized list of answer options, 2-10 strings 1-100 characters each
- **business_connection_id** Unique identifier of the business connection on behalf of which the message will be sent
- **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_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- **allow_sending_without_reply** Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

instance of method aiogram.methods.send_poll.SendPoll

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

 $model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

answer_dice(business_connection_id: Optional[str] = None, message_thread_id: Optional[int] = None, emoji: Optional[str] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = 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

- **business_connection_id** Unique identifier of the business connection on behalf of which the message will be sent
- **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_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account

- **allow_sending_without_reply** Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

instance of method aiogram.methods.send_dice.SendDice

answer_sticker (sticker: Union[InputFile, str], business_connection_id: Optional[str] = None, message_thread_id: Optional[int] = None, emoji: Optional[str] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = 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, .TGS, or .WEBM sticker using multipart/form-data. *More information on Sending Files* ». Video and animated stickers can't be sent via an HTTP URL.
- **business_connection_id** Unique identifier of the business connection on behalf of which the message will be sent
- **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_parameters** Description of the message to reply to
- **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. Not supported for messages sent on behalf of a business account.
- allow_sending_without_reply Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

Returns

instance of method aiogram.methods.send_sticker.SendSticker

answer_venue(latitude: float, longitude: float, title: str, address: str, business_connection_id: Optional[str] = None, message_thread_id: Optional[int] = None, foursquare_id: Optional[str] = None, foursquare_type: Optional[str] = None, google_place_id: Optional[str] = None, google_place_type: Optional[str] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = 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
- **business_connection_id** Unique identifier of the business connection on behalf of which the message will be sent
- **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_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- allow_sending_without_reply Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

Returns

instance of method aiogram.methods.send_venue.SendVenue

answer_video(video: Union[InputFile, str], business_connection_id: Optional[str] = None, message_thread_id: Optional[int] = None, duration: Optional[int] = None, width: Optional[int] = None, height: Optional[int] = None, thumbnail: Optional[InputFile] = None, caption: Optional[str] = None, parse_mode: Optional[Union[str, Default]] = <Default('parse_mode')>, caption_entities: Optional[List[MessageEntity]] = None, has_spoiler: Optional[bool] = None, supports_streaming: Optional[bool] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = 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

- **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* »
- **business_connection_id** Unique identifier of the business connection on behalf of which the message will be sent
- **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_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- **allow_sending_without_reply** Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

instance of method aiogram.methods.send_video.SendVideo

answer_video_note(video_note: Union[InputFile, str], business_connection_id: Optional[str] = None, message_thread_id: Optional[int] = None, duration: Optional[int] = None, length: Optional[int] = None, thumbnail: Optional[InputFile] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = 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

- 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
- **business_connection_id** Unique identifier of the business connection on behalf of which the message will be sent
- **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

- **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_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- **allow_sending_without_reply** Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

instance of method aiogram.methods.send_video_note.SendVideoNote

answer_voice(*voice*: Union[InputFile, str], business_connection_id: Optional[str] = None,

message_thread_id: Optional[int] = None, caption: Optional[str] = None, parse_mode: Optional[Union[str, Default]] = <Default('parse_mode')>, caption_entities: Optional[List[MessageEntity]] = None, duration: Optional[int] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = 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

- **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* »
- **business_connection_id** Unique identifier of the business connection on behalf of which the message will be sent
- **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_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- **allow_sending_without_reply** Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

instance of method aiogram.methods.send_voice.SendVoice

ChatPermissions

can_send_documents: bool | None = None, can_send_documents: bool | None = None, can_send_photos: bool | None = None, can_send_videos: 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, giveaways, giveaway winners, 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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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

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.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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

This object contains information about a chat that 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.

title: str | None

Optional. Title of the chat, if the title was requested by the bot.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

username: str | None

Optional. Username of the chat, if the username was requested by the bot and available.

photo: List[PhotoSize] | None

Optional. Available sizes of the chat photo, if the photo was requested by the bot

Contact

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

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

 $model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

```
model_post_init(ModelMetaclass\_context: Any) \rightarrow None
```

We need to both initialize private attributes and call the user-defined model_post_init method.

class aiogram.types.dice.DiceEmoji

```
DICE = ''
DART = ''
BASKETBALL = ''
FOOTBALL = ''
SLOT_MACHINE = ''
```

BOWLING = ''

Document

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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.

ExternalReplyInfo

class aiogram.types.external_reply_info. ExternalReplyInfo (*, <i>origin:</i> MessageOriginUser	
1	MessageOriginHiddenUser
1	MessageOriginChat
1	MessageOriginChannel, chat: Chat
1	None = None, message_id: int None =
1	None, link_preview_options:
J	LinkPreviewOptions None = None,
	animation: Animation None = None,
	audio: Audio None = None, document:
J	Document <i>None</i> = <i>None</i> , <i>photo</i> :
	<i>List</i> [PhotoSize] <i>None</i> = <i>None</i> , <i>sticker</i> :
	Sticker None = None, story: Story
	None = None, video: Video None =
1	None, video_note: VideoNote None =
1	None, voice: Voice None = None,
I	has_media_spoiler: bool None = None,
	contact: Contact None = None, dice:
J	Dice <i>None</i> = <i>None</i> , <i>game</i> : Game <i>None</i>
:	= None, giveaway: Giveaway None =
1	None, giveaway_winners:
(GiveawayWinners None = None,
i	invoice: Invoice $None = None$,
I	location: Location None = None, poll:
I	Poll <i>None = None</i> , <i>venue</i> : Venue <i>None</i>
:	= None, **extra_data: Any)
	. 1.1 6 .1 1 .

This object contains information about a message that is being replied to, which may come from another chat or forum topic.

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

origin: MessageOriginUser | MessageOriginHiddenUser | MessageOriginChat | MessageOriginChannel

Origin of the message replied to by the given message

chat: Chat | None

Optional. Chat the original message belongs to. Available only if the chat is a supergroup or a channel.

message_id: int | None

Optional. Unique message identifier inside the original chat. Available only if the original chat is a supergroup or a channel.

link_preview_options: LinkPreviewOptions | None

Optional. Options used for link preview generation for the original message, if it is a text message

animation: Animation | None

Optional. Message is an animation, information about the animation

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

voice: Voice | None

Optional. Message is a voice message, information about the file

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 »

giveaway: Giveaway | None

Optional. Message is a scheduled giveaway, information about the giveaway

giveaway_winners: GiveawayWinners | None

Optional. A giveaway with public winners was completed

invoice: Invoice | None

Optional. Message is an invoice for a payment, information about the invoice. More about payments »

location: Location | None

Optional. Message is a shared location, information about the location

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

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>. 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.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

 $model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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 to a message in the same chat and forum topic, sender of the original message.

ForumTopic

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) \rightarrow None

We need to both initialize private attributes and call the user-defined model_post_init method.

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass_context: Any*) \rightarrow None We need to both initialize private attributes and call the user-defined model_post_init method.

ForumTopicCreated

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(_*ModelMetaclass_context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

icon_custom_emoji_id: str | None

Optional. Unique identifier of the custom emoji shown as the topic icon

ForumTopicEdited

**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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

 $model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
 A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass_context:* Any) \rightarrow None We need to both initialize private attributes and call the user-defined model_post_init method.

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context:* Any) \rightarrow None We need to both initialize private attributes and call the user-defined model_post_init method.

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

 $model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

Giveaway

This object represents a message about a scheduled giveaway.

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

chats: List[Chat]

The list of chats which the user must join to participate in the giveaway

winners_selection_date: DateTime

Point in time (Unix timestamp) when winners of the giveaway will be selected

winner_count: int

The number of users which are supposed to be selected as winners of the giveaway

only_new_members: bool | None

Optional. True, if only users who join the chats after the giveaway started should be eligible to win

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

has_public_winners: bool | None

Optional. True, if the list of giveaway winners will be visible to everyone

prize_description: str | None

Optional. Description of additional giveaway prize

country_codes: List[str] | None

Optional. A list of two-letter ISO 3166-1 alpha-2 country codes indicating the countries from which eligible users for the giveaway must come. If empty, then all users can participate in the giveaway. Users with a phone number that was bought on Fragment can always participate in giveaways.

premium_subscription_month_count: int | None

Optional. The number of months the Telegram Premium subscription won from the giveaway will be active for

GiveawayCompleted

class aiogram.types.giveaway_completed.GiveawayCompleted(*, winner_count: int,

unclaimed_prize_count: int | None =
None, giveaway_message: Message |
None = None, **extra_data: Any)

This object represents a service message about the completion of a giveaway without public winners.

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

winner_count: int

Number of winners in the giveaway

unclaimed_prize_count: int | None

Optional. Number of undistributed prizes

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

giveaway_message: Message | None

Optional. Message with the giveaway that was completed, if it wasn't deleted

GiveawayCreated

class aiogram.types.giveaway_created.GiveawayCreated(**extra_data: Any)

This object represents a service message about the creation of a scheduled giveaway. Currently holds no information.

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

 $model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

GiveawayWinners

class aiogram.types.giveaway_winners.GiveawayWinners(*, chat: Chat, giveaway_message_id: int,

winners_selection_date: datetime, winner_count: int, winners: List[User], additional_chat_count: int | None = None, premium_subscription_month_count: int | None = None, unclaimed_prize_count: int | None = None, only_new_members: bool | None = None, was_refunded: bool | None = None, prize_description: str | None = None, **extra_data: Any)

This object represents a message about the completion of a giveaway with public winners.

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

chat: Chat

The chat that created the giveaway

giveaway_message_id: int

Identifier of the message with the giveaway in the chat

winners_selection_date: DateTime

Point in time (Unix timestamp) when winners of the giveaway were selected

winner_count: int

Total number of winners in the giveaway

winners: List[User]

List of up to 100 winners of the giveaway

additional_chat_count: int | None

Optional. The number of other chats the user had to join in order to be eligible for the giveaway

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

premium_subscription_month_count: int | None

Optional. The number of months the Telegram Premium subscription won from the giveaway will be active for

unclaimed_prize_count: int | None

Optional. Number of undistributed prizes

only_new_members: bool | None

Optional. True, if only users who had joined the chats after the giveaway started were eligible to win

was_refunded: bool | None

Optional. True, if the giveaway was canceled because the payment for it was refunded

prize_description: str | None

Optional. Description of additional giveaway prize

InaccessibleMessage

This object describes a message that was deleted or is otherwise inaccessible to the bot.

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

chat: Chat

Chat the message belonged to

message_id: int

Unique message identifier inside the chat

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

date: Literal[0]

Always 0. The field can be used to differentiate regular and inaccessible messages.

InlineKeyboardButton

class aiogram.types.inline_keyboard_button. InlineKeyboardButton (*, <i>text: str</i> , <i>url: str</i> <i>None</i> =	
	None, callback_data: str None
	= None, web_app: WebAppInfo
	$ None = None, login_url:$
	LoginUrl None = None,
	switch_inline_query: str None
	= None,
	<pre>switch_inline_query_current_chat:</pre>
	$str \mid None = None,$
	<pre>switch_inline_query_chosen_chat:</pre>
	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 identifier 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.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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.

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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) \rightarrow *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*) \rightarrow AsyncGenerator[bytes, 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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

 $model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

InputMediaAnimation

```
class aiogram.types.input_media_animation.InputMediaAnimation(*, type: ~typ-
```

ing.Literal[InputMediaType.ANIMATION] = InputMediaType.ANIMATION, media: str | ~aiogram.types.input file.InputFile, thumbnail: ~aiogram.types.input_file.InputFile | None = None, caption: str | None = None, parse_mode: str | ~aiogram.client.default.Default | *None* = <*Default('parse_mode')*>, caption_entities: ~typing.List[~aiogram.types.message_entity.MessageEntit | None = None, width: int | None = *None*, *height: int* | *None* = *None*, duration: $int \mid None = None$, 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 | None

Optional. Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported serverside. 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 | Default | None

Optional. Mode for parsing entities in the animation caption. See formatting options for more details.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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[InputMediaType.AUDIO] =
InputMediaType.AUDIO, media: str |
~aiogram.types.input_file.InputFile,
thumbnail:
~aiogram.types.input_file.InputFile | None =
None, caption: str | None = None,
parse_mode: str |
~aiogram.client.default.Default | None =
<Default('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.AUDI0]

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 | None

Optional. Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported serverside. 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 | Default | None

Optional. Mode for parsing entities in the audio caption. See formatting options for more details.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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: ~typ-

ing.Literal[InputMediaType.DOCUMENT] = InputMediaType.DOCUMENT, media: str | ~aiogram.types.input_file.InputFile, thumbnail: ~aiogram.types.input file.InputFile | None = None, caption: str | None = None, parse mode: str | ~aiogram.client.default.Default | None = <Default('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 | None

Optional. Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported serverside. 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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

parse_mode: str | Default | 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/formdata. Always **True**, if the document is sent as part of an album.

InputMediaPhoto

class aiogram.types.input_media_photo.InputMediaPhoto(*, type:

~typing.Literal[InputMediaType.PHOTO] =
InputMediaType.PHOTO, media: str |
~aiogram.types.input_file.InputFile, caption:
str | None = None, parse_mode: str |
~aiogram.client.default.Default | None =
<Default('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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

parse_mode: str | Default | 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[InputMediaType.VIDEO] = InputMediaType.VIDEO, media: str | ~aiogram.types.input_file.InputFile, thumbnail: ~aiogram.types.input file.InputFile | None = *None*, *caption: str* | *None* = *None*, parse mode: str ~aiogram.client.default.Default | None = <Default('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.VIDE0]

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 | None

Optional. Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported serverside. 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 | Default | 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

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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_users:

KeyboardButtonRequestUsers | 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, request_user: KeyboardButtonRequestUser | 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_users*, *request_chat*, *request_contact*, *request_location*, and *request_poll* are mutually exclusive. **Note:** *request_users* 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_users: KeyboardButtonRequestUsers | None

Optional. If specified, pressing the button will open a list of suitable users. Identifiers of selected users will be sent to the bot in a 'users_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.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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.

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.

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

KeyboardButtonPollType

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.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(_*ModelMetaclass_context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

KeyboardButtonRequestChat

class aiogram.types.keyboard_button_request_chat.KeyboardButtonRequestChat(*, request_id: int, *chat_is_channel:* bool, chat_is_forum: $bool \mid None =$ None, chat has username: $bool \mid None =$ None. chat_is_created: $bool \mid None =$ None, user_administrator_rights: ChatAdministratorRights | None =None, *bot_administrator_rights:* ChatAdministratorRights | None =None, bot_is_member: $bool \mid None =$ None. request title: bool |None = None, request username: $bool \mid None =$ None. request_photo: $bool \mid None =$ None. **extra data: Any)

This object defines the criteria used to request a suitable chat. Information about the selected chat will be shared with the bot when the corresponding button is pressed. The bot will be granted requested rights in the hat if appropriate 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.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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.

request_title: bool | None

Optional. Pass True to request the chat's title

request_username: bool | None

Optional. Pass True to request the chat's username

request_photo: bool | None

Optional. Pass True to request the chat's photo

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 »

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

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.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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.

KeyboardButtonRequestUsers

class aiogram.types.keyboard_button_request_users.KeyboardButtonRequestUsers(*, request_id:

int, *user_is_bot*: $bool \mid None =$ None, user_is_premium: $bool \mid None =$ None. *max_quantity:* $int \mid None =$ None, request name: $bool \mid None =$ None, request_username: bool | None = None, *request_photo:* $bool \mid None =$ None. **extra_data: Any)

This object defines the criteria used to request suitable users. Information about the selected users will be shared with the bot when the corresponding button is pressed. More about requesting users »

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

request_id: int

Signed 32-bit identifier of the request that will be received back in the *aiogram.types.users_shared*. *UsersShared* object. Must be unique within the message

user_is_bot: bool | None

Optional. Pass True to request bots, pass False to request regular users. If not specified, no additional restrictions are applied.

user_is_premium: bool | None

Optional. Pass True to request premium users, pass False to request non-premium users. If not specified, no additional restrictions are applied.

max_quantity: int | None

Optional. The maximum number of users to be selected; 1-10. Defaults to 1.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

request_name: bool | None

Optional. Pass True to request the users' first and last name

request_username: bool | None

Optional. Pass True to request the users' username

request_photo: bool | None

Optional. Pass True to request the users' photo

LinkPreviewOptions

class aiogram.types.link_preview_options.LinkPreviewOptions(*, is_disabled: bool |

~aiogram.client.default.Default | None = < Default('link_preview_is_disabled')>, *url:* $str \mid None = None$, prefer_small_media: bool | ~aiogram.client.default.Default | None = < Default('link_preview_prefer_small_media')>, prefer large media: bool ~aiogram.client.default.Default | None $= \langle De \rangle$ fault('link_preview_prefer_large_media')>, show above text: bool ~aiogram.client.default.Default | None $= \langle De \rangle$ fault('link_preview_show_above_text')>, **extra_data: ~typing.Any)

Describes the options used for link preview generation.

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

is_disabled: bool | Default | None

Optional. True, if the link preview is disabled

url: str | None

Optional. URL to use for the link preview. If empty, then the first URL found in the message text will be used

prefer_small_media: bool | Default | None

Optional. True, if the media in the link preview is supposed to be shrunk; ignored if the URL isn't explicitly specified or media size change isn't supported for the preview

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init($ModelMetaclass_context: Any$) \rightarrow None

We need to both initialize private attributes and call the user-defined model_post_init method.

prefer_large_media: bool | Default | None

Optional. True, if the media in the link preview is supposed to be enlarged; ignored if the URL isn't explicitly specified or media size change isn't supported for the preview

show_above_text: bool | Default | None

Optional. True, if the link preview must be shown above the message text; otherwise, the link preview will be shown below the message text

Location

This object represents a point on the map.

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

latitude: float

Latitude as defined by sender

longitude: float

Longitude as defined by sender

horizontal_accuracy: float | None

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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

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.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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.

MaybelnaccessibleMessage

class aiogram.types.maybe_inaccessible_message.MaybeInaccessibleMessage(**extra_data: Any)

This object describes a message that can be inaccessible to the bot. It can be one of

- aiogram.types.message.Message
- aiogram.types.inaccessible_message.InaccessibleMessage

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

We need to both initialize private attributes and call the user-defined model_post_init method.

MenuButton

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(_*ModelMetaclass_context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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: Lit-

eral[MenuButtonType.COMMANDS] = MenuButtonType.COMMANDS, text: str | None = None, web_app: WebAppInfo | None = None, **extra_data: 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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

MenuButtonDefault

class aiogram.types.menu_button_default.MenuButtonDefault(*, type:

Literal[MenuButtonType.DEFAULT] = MenuButtonType.DEFAULT, text: str | None = None, web_app: WebAppInfo | None = None, **extra_data: 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*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(_*ModelMetaclass_context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

MenuButtonWebApp

class aiogram.types.menu_button_web_app.MenuButtonWebApp(*, type:

Literal[MenuButtonType.WEB_APP] = MenuButtonType.WEB_APP, text: str, web_app: WebAppInfo, **extra_data: 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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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*.

Message

class aiogram.types.message.Message(*, message id: int, date: datetime, chat: Chat, message thread id: *int* | *None* = *None*, *from_user*: User | *None* = *None*, *sender_chat*: Chat | None = None, sender boost count: int | None = None, *sender_business_bot:* User | *None = None, business_connection_id: str* | *None* = *None*, *forward_origin*: MessageOriginUser | MessageOriginHiddenUser | MessageOriginChat | MessageOriginChannel | None = None, is topic message: bool | *None* = *None*, *is automatic forward: bool* | *None* = *None*, reply to message: Message | None = None, external reply: ExternalReplyInfo | *None* = *None*, *quote*: TextQuote | *None* = *None*, *reply_to_story:* Story | *None = None, via_bot:* User | *None = None,* edit_date: int | None = None, has_protected_content: bool | None = *None*, *is_from_offline: bool* | *None* = *None*, *media_group_id: str* | *None* = *None*, *author_signature: str* | *None* = *None*, *text: str* | *None* = *None*, *entities: List*[MessageEntity] | *None* = *None*, *link_preview_options:* LinkPreviewOptions | *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 | InaccessibleMessage | *None* = *None*, *invoice*: Invoice | *None* = *None*, *successful_payment*: SuccessfulPayment | None = None, users shared: UsersShared | *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*, *boost* added: ChatBoostAdded | *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*, *giveaway_created*: GiveawayCreated | *None* = *None*, giveaway: Giveaway | None = None, giveaway_winners: GiveawayWinners | *None* = *None*, *giveaway_completed*: GiveawayCompleted | 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: **Ghapter**yBoa **Gontents** | None = None, forward date: datetime | None = None, forward from: User | None = None, forward from chat: Chat | None = None, forward_from_message_id: int | None = None,

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. It is always a positive number, representing a valid date.

chat: Chat

Chat 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.

sender_boost_count: int | None

Optional. If the sender of the message boosted the chat, the number of boosts added by the user

sender_business_bot: User | None

Optional. The bot that actually sent the message on behalf of the business account. Available only for outgoing messages sent on behalf of the connected business account.

business_connection_id: str | None

Optional. Unique identifier of the business connection from which the message was received. If non-empty, the message belongs to a chat of the corresponding business account that is independent from any potential bot chat which might share the same identifier.

forward_origin: MessageOriginUser | MessageOriginHiddenUser | MessageOriginChat | MessageOriginChannel | None

Optional. Information about the original message for forwarded messages

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 in the same chat and message thread, 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.

external_reply: ExternalReplyInfo | None

Optional. Information about the message that is being replied to, which may come from another chat or forum topic

quote: TextQuote | None

Optional. For replies that quote part of the original message, the quoted part of the message

reply_to_story: Story | None

Optional. For replies to a story, the original story

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

is_from_offline: bool | None

Optional. True, if the message was sent by an implicit action, for example, as an away or a greeting business message, or as a scheduled message

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

link_preview_options: LinkPreviewOptions | None

Optional. Options used for link preview generation for the message, if it is a text message and link preview options were changed

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 | InaccessibleMessage | 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 itself is 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 »

users_shared: UsersShared | None

Optional. Service message: users were 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 to write messages after adding it to the attachment or side menu, launching a Web App from a link, or accepting an explicit request from a Web App sent by the method requestWriteAccess

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.

boost_added: ChatBoostAdded | None Optional. Service message: user boosted the chat

- model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

- **model_post_init**($ModelMetaclass_context: Any$) \rightarrow None We need to both initialize private attributes and call the user-defined model_post_init method.
- giveaway_created: GiveawayCreated | None

Optional. Service message: a scheduled giveaway was created

giveaway: Giveaway | None

Optional. The message is a scheduled giveaway message

giveaway_winners: GiveawayWinners | None

Optional. A giveaway with public winners was completed

giveaway_completed: GiveawayCompleted | None

Optional. Service message: a giveaway without public winners was completed

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.

forward_date: DateTime | None

Optional. For forwarded messages, date the original message was sent in Unix time

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

forward_from: User | None

Optional. For forwarded messages, sender of the original message

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

forward_from_chat: Chat | None

Optional. For messages forwarded from channels or from anonymous administrators, information about the original sender chat

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

forward_from_message_id: int | None

Optional. For messages forwarded from channels, identifier of the original message in the channel

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

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

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

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

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

user_shared: UserShared | None

Optional. Service message: a user was shared with the bot

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

property content_type: str

property html_text: str

property md_text: str

reply_animation(animation: Union[InputFile, str], duration: Optional[int] = None, width: Optional[int] =
None, height: Optional[int] = None, thumbnail: Optional[InputFile] = None, caption:
Optional[str] = None, parse_mode: Optional[Union[str, Default]] =
<Default('parse_mode')>, caption_entities: Optional[List[MessageEntity]] = None,
has_spoiler: Optional[bool] = None, disable_notification: Optional[bool] = None,
protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>,
reply_parameters: Optional[ReplyParameters] = None, reply_markup:
Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove,
ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, **kwargs:
Any) → SendAnimation

Shortcut for method *aiogram.methods.send_animation.SendAnimation* will automatically fill method attributes:

- chat_id
- message_thread_id

- business_connection_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
- reply_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- 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_animation.SendAnimation

answer_animation (animation: Union[InputFile, str], duration: Optional[int] = None, width: Optional[int] = None, height: Optional[int] = None, thumbnail: Optional[InputFile] = None, caption: Optional[str] = None, parse_mode: Optional[Union[str, Default]] = <Default('parse_mode')>, caption_entities: Optional[List[MessageEntity]] = None, has_spoiler: Optional[bool] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = None, **kwargs: Any) → SendAnimation

Shortcut for method *aiogram.methods.send_animation.SendAnimation* will automatically fill method attributes:

- chat_id
- message_thread_id
- business_connection_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

- **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_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- allow_sending_without_reply Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

instance of method aiogram.methods.send_animation.SendAnimation

reply_audio(audio: Union[InputFile, str], caption: Optional[str] = None, parse_mode: Optional[Union[str, Default]] = <Default('parse_mode')>, caption_entities: Optional[List[MessageEntity]] = None, duration: Optional[int] = None, performer: Optional[str] = None, title: Optional[str] = None, thumbnail: Optional[InputFile] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, **kwargs: Any) → SendAudio

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

- chat_id
- message_thread_id
- business_connection_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

- **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_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- allow_sending_without_reply Pass True if the message should be sent even if the specified replied-to message is not found

instance of method aiogram.methods.send_audio.SendAudio

answer_audio(*audio: Union*[InputFile, str], caption: Optional[str] = None, parse_mode:

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

- chat_id
- message_thread_id
- business_connection_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

- **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_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- allow_sending_without_reply Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

instance of method aiogram.methods.send_audio.SendAudio

 $\begin{aligned} \textbf{reply_contact}(phone_number: str, first_name: str, last_name: Optional[str] = None, vcard: Optional[str] \\ = None, disable_notification: Optional[bool] = None, protect_content: \\ Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: \\ Optional[ReplyParameters] = None, reply_markup: \\ Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, \\ ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, **kwargs: \\ Any) \rightarrow SendContact \end{aligned}$

Shortcut for method *aiogram.methods.send_contact.SendContact* will automatically fill method attributes:

- chat_id
- message_thread_id
- business_connection_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

- 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_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- allow_sending_without_reply Pass True if the message should be sent even if the specified replied-to message is not found

instance of method aiogram.methods.send_contact.SendContact

answer_contact(phone_number: str, first_name: str, last_name: Optional[str] = None, vcard: Optional[str] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = None, **kwargs: Any) → SendContact

Shortcut for method *aiogram.methods.send_contact.SendContact* will automatically fill method attributes:

- chat_id
- message_thread_id
- business_connection_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

- 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_parameters** Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- **allow_sending_without_reply** Pass True if the message should be sent even if the specified replied-to message is not found

• reply_to_message_id – If the message is a reply, ID of the original message

Returns

instance of method aiogram.methods.send_contact.SendContact

reply_document(*document*: Union[InputFile, str], thumbnail: Optional[InputFile] = None, caption:

 $\begin{array}{l} Optional[str] = None, parse_mode: Optional[Union[str, Default]] = \\ < Default('parse_mode')>, caption_entities: Optional[List[MessageEntity]] = None, \\ disable_content_type_detection: Optional[bool] = None, disable_notification: \\ Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = \\ < Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, \\ reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, \\ ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: \\ Optional[bool] = None, **kwargs: Any) \rightarrow SendDocument \end{array}$

Shortcut for method *aiogram.methods.send_document.SendDocument* will automatically fill method attributes:

- chat_id
- message_thread_id
- business_connection_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

- 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_parameters** Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- allow_sending_without_reply Pass True if the message should be sent even if the specified replied-to message is not found

instance of method aiogram.methods.send_document.SendDocument

answer_document(document: Union[InputFile, str], thumbnail: Optional[InputFile] = None, caption: Optional[str] = None, parse_mode: Optional[Union[str, Default]] = <Default('parse_mode')>, caption_entities: Optional[List[MessageEntity]] = None, disable_content_type_detection: Optional[bool] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = None, **kwargs: Any) → SendDocument

Shortcut for method *aiogram.methods.send_document.SendDocument* will automatically fill method attributes:

- chat_id
- message_thread_id
- business_connection_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

- 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_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- allow_sending_without_reply Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

instance of method aiogram.methods.send_document.SendDocument

Shortcut for method aiogram.methods.send_game.SendGame will automatically fill method attributes:

- chat_id
- message_thread_id
- business_connection_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
- reply_parameters Description of the message to reply to
- **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. Not supported for messages sent on behalf of a business account.
- 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_game.SendGame

Shortcut for method *aiogram.methods.send_game.SendGame* will automatically fill method attributes:

- chat_id
- message_thread_id
- business_connection_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_parameters Description of the message to reply to
- **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. Not supported for messages sent on behalf of a business account.
- **allow_sending_without_reply** Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

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: Optional[int] = None, suggested_tip_amounts: Optional[List[int]] = None, start_parameter: Optional[str] = None, provider_data: Optional[str] = None, photo_url: Optional[str] = None, photo_size: Optional[int] = None, photo_width: Optional[int] = None, photo_height: Optional[int] = None, need_name: Optional[bool] = None, need_phone_number: Optional[bool] = None, need_email: Optional[bool] = None, need_shipping_address: Optional[bool] = None, send_phone_number_to_provider: Optional[bool] = None, send_email_to_provider: Optional[bool] = None, is_flexible: Optional[bool] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[InlineKeyboardMarkup] = None, allow_sending_without_reply: Optional[bool] = None, **kwargs: Any) → SendInvoice

Shortcut for method *aiogram.methods.send_invoice.SendInvoice* will automatically fill method attributes:

- chat_id
- message_thread_id
- business_connection_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

- 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_parameters Description of the message to reply to
- **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.
- **allow_sending_without_reply** Pass True if the message should be sent even if the specified replied-to message is not found

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: Optional[int] = None, suggested_tip_amounts: Optional[List[int]] = None, start_parameter: Optional[str] = None, provider_data: Optional[str] = None, photo_url: Optional[str] = None, photo_size: Optional[int] = None, photo_width: Optional[int] = None, photo_height: Optional[int] = None, need_name: Optional[bool] = None, need_phone_number: Optional[bool] = None, need_email: Optional[bool] = None, need_shipping_address: Optional[bool] = None, send_phone_number_to_provider: Optional[bool] = None, send_email_to_provider: Optional[bool] = None, is_flexible: Optional[bool] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[InlineKeyboardMarkup] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = None, **kwargs: Any) → SendInvoice

Shortcut for method *aiogram.methods.send_invoice.SendInvoice* will automatically fill method attributes:

- chat_id
- message_thread_id
- business_connection_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

- 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_parameters Description of the message to reply to
- **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.
- allow_sending_without_reply Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

instance of method aiogram.methods.send_invoice.SendInvoice

reply_location(*latitude: float, longitude: float, horizontal_accuracy: Optional[float] = None, live_period:*

Optional[int] = None, heading: Optional[int] = None, proximity_alert_radius: Optional[int] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, **kwargs: Any) → SendLocation

Shortcut for method *aiogram.methods.send_location.SendLocation* will automatically fill method attributes:

- chat_id
- message_thread_id
- business_connection_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
- reply_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- 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_location.SendLocation

answer_location(latitude: float, longitude: float, horizontal_accuracy: Optional[float] = None, live_period: Optional[int] = None, heading: Optional[int] = None, proximity_alert_radius: Optional[int] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = None, **kwargs: Any) → SendLocation

Shortcut for method *aiogram.methods.send_location.SendLocation* will automatically fill method attributes:

- chat_id
- message_thread_id
- business_connection_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_parameters** Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- **allow_sending_without_reply** Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

Returns

instance of method aiogram.methods.send_location.SendLocation

 $\label{eq:reply_media_group(media: List[Union[InputMediaAudio, InputMediaDocument, InputMediaPhoto, InputMediaVideo]], disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, allow_sending_without_reply: Optional[bool] = None, **kwargs: Any) \rightarrow SendMediaGroup$

Shortcut for method *aiogram.methods.send_media_group.SendMediaGroup* will automatically fill method attributes:

- chat_id
- message_thread_id
- business_connection_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
- reply_parameters Description of the message to reply to
- **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[Union[InputMediaAudio, InputMediaDocument, InputMediaPhoto, InputMediaVideo]], disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = None, **kwargs: Any) → SendMediaGroup

Shortcut for method *aiogram.methods.send_media_group.SendMediaGroup* will automatically fill method attributes:

- chat_id
- message_thread_id
- business_connection_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_parameters Description of the message to reply to
- **allow_sending_without_reply** Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the messages are a reply, ID of the original message

 $instance\ of\ method\ aiogram.methods.send_media_group.SendMediaGroup$

reply(text: str, parse_mode: Optional[Union[str, Default]] = <Default('parse_mode')>, entities: Optional[List[MessageEntity]] = None, link_preview_options: Optional[Union[LinkPreviewOptions, Default]] = <Default('link_preview')>, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, disable_web_page_preview: Optional[Union[bool, Default]] = <Default('link_preview_is_disabled')>, **kwargs: Any) → SendMessage

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

- chat_id
- message_thread_id
- business_connection_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

- 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*
- link_preview_options Link preview generation options for the 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_parameters** Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- **allow_sending_without_reply** Pass True if the message should be sent even if the specified replied-to message is not found

• disable_web_page_preview – Disables link previews for links in this message

Returns

instance of method aiogram.methods.send_message.SendMessage

answer(text: str, parse_mode: Optional[Union[str, Default]] = <Default('parse_mode')>, entities: Optional[List[MessageEntity]] = None, link_preview_options: Optional[Union[LinkPreviewOptions, Default]] = <Default('link_preview')>, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, disable_web_page_preview: Optional[Union[bool, Default]] = <Default('link_preview_is_disabled')>, reply_to_message_id: Optional[int] = None, **kwargs: Any) → SendMessage

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

- chat_id
- message_thread_id
- business_connection_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*
- link_preview_options Link preview generation options for the 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_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- **allow_sending_without_reply** Pass True if the message should be sent even if the specified replied-to message is not found
- disable_web_page_preview Disables link previews for links in this message
- reply_to_message_id If the message is a reply, ID of the original message

Returns

instance of method aiogram.methods.send_message.SendMessage

reply_photo(photo: Union[InputFile, str], caption: Optional[str] = None, parse_mode: Optional[Union[str, Default]] = <Default('parse_mode')>, caption_entities: Optional[List[MessageEntity]] = None, has_spoiler: Optional[bool] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, **kwargs: Any) → SendPhoto

Shortcut for method *aiogram.methods.send_photo.SendPhoto* will automatically fill method attributes:

- chat_id
- message_thread_id
- business_connection_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
- reply_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- 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_photo.SendPhoto

answer_photo(photo: Union[InputFile, str], caption: Optional[str] = None, parse_mode:

Optional[Union[str, Default]] = <Default('parse_mode')>, caption_entities: Optional[List[MessageEntity]] = None, has_spoiler: Optional[bool] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = None, **kwargs: Any) → SendPhoto

Shortcut for method *aiogram.methods.send_photo.SendPhoto* will automatically fill method attributes:

- chat_id
- message_thread_id
- business_connection_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_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- **allow_sending_without_reply** Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

Returns

instance of method aiogram.methods.send_photo.SendPhoto

reply_poll(question: str, options: List[str], is_anonymous: Optional[bool] = None, type: Optional[str] = None, allows_multiple_answers: Optional[bool] = None, correct_option_id: Optional[int] = None, explanation: Optional[str] = None, explanation_parse_mode: Optional[Union[str, Default]] = <Default('parse_mode')>, explanation_entities: Optional[List[MessageEntity]] = None, open_period: Optional[int] = None, close_date: Optional[Union[datetime.datetime, datetime.timedelta, int]] = None, is_closed: Optional[bool] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, **kwargs: Any) → SendPoll

Shortcut for method *aiogram.methods.send_poll.SendPoll* will automatically fill method attributes:

- chat_id
- message_thread_id
- business_connection_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

- 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_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- allow_sending_without_reply Pass True if the message should be sent even if the specified replied-to message is not found

instance of method aiogram.methods.send_poll.SendPoll

answer_poll(question: str, options: List[str], is_anonymous: Optional[bool] = None, type: Optional[str] =
None, allows_multiple_answers: Optional[bool] = None, correct_option_id: Optional[int] =
None, explanation: Optional[str] = None, explanation_parse_mode: Optional[Union[str,
Default]] = <Default('parse_mode')>, explanation_entities: Optional[List[MessageEntity]] =
None, open_period: Optional[int] = None, close_date: Optional[Union[datetime.datetime,
datetime.timedelta, int]] = None, is_closed: Optional[bool] = None, disable_notification:
Optional[bool] = None, protect_content: Optional[Union[bool, Default]] =
<Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None,
reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup,
ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool]
= None, reply_to_message_id: Optional[int] = None, **kwargs: Any) → SendPoll

Shortcut for method *aiogram.methods.send_poll.SendPoll* will automatically fill method attributes:

- chat_id
- message_thread_id
- business_connection_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

- 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_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- allow_sending_without_reply Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

instance of method aiogram.methods.send_poll.SendPoll

 $\begin{aligned} \textbf{reply_dice}(emoji: Optional[str] = None, disable_notification: Optional[bool] = None, protect_content: \\ Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: \\ Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, \\ allow_sending_without_reply: Optional[bool] = None, **kwargs: Any) \rightarrow SendDice \end{aligned}$

Shortcut for method *aiogram.methods.send_dice.SendDice* will automatically fill method attributes:

- chat_id
- message_thread_id
- business_connection_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

- **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_parameters** Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account

• **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_dice.SendDice

Shortcut for method *aiogram.methods.send_dice.SendDice* will automatically fill method attributes:

- chat_id
- message_thread_id
- business_connection_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_parameters** Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- **allow_sending_without_reply** Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

Returns

instance of method aiogram.methods.send_dice.SendDice

reply_sticker(*sticker*: Union[InputFile, str], emoji: Optional[str] = None, disable_notification: Optional[bool] = None, protect content: Optional[Union[bool, Default]] =

 $Optional[bool] = None, protect_content: Optional[Union[bool, Default]] =$ $<Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None,$ $reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup,$ $ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply:$ $Optional[bool] = None, **kwargs: Any) <math>\rightarrow$ SendSticker

Shortcut for method *aiogram.methods.send_sticker.SendSticker* will automatically fill method attributes:

- chat_id
- message_thread_id

- business_connection_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, .TGS, or .WEBM sticker using multipart/form-data. *More information on Sending Files* ». Video and 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_parameters** Description of the message to reply to
- **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. Not supported for messages sent on behalf of a business account.
- **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_sticker.SendSticker

answer_sticker (sticker: Union[InputFile, str], emoji: Optional[str] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = None, **kwargs: Any) → SendSticker

Shortcut for method *aiogram.methods.send_sticker.SendSticker* will automatically fill method attributes:

- chat_id
- message_thread_id
- business_connection_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, .TGS, or .WEBM sticker using multipart/form-data. *More information on Sending Files* ». Video and 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_parameters Description of the message to reply to
- **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. Not supported for messages sent on behalf of a business account.
- allow_sending_without_reply Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

instance of method aiogram.methods.send_sticker.SendSticker

reply_venue(latitude: float, longitude: float, title: str, address: str, foursquare_id: Optional[str] = None, foursquare_type: Optional[str] = None, google_place_id: Optional[str] = None, google_place_type: Optional[str] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, **kwargs: Any) → SendVenue

Shortcut for method *aiogram.methods.send_venue.SendVenue* will automatically fill method attributes:

- chat_id
- message_thread_id
- business_connection_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

- **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_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- allow_sending_without_reply Pass True if the message should be sent even if the specified replied-to message is not found

instance of method aiogram.methods.send_venue.SendVenue

answer_venue(latitude: float, longitude: float, title: str, address: str, foursquare_id: Optional[str] = None, foursquare_type: Optional[str] = None, google_place_id: Optional[str] = None, google_place_type: Optional[str] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = None, **kwargs: Any) → SendVenue

Shortcut for method *aiogram.methods.send_venue.SendVenue* will automatically fill method attributes:

- chat_id
- message_thread_id
- business_connection_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

- 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_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account

- **allow_sending_without_reply** Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

instance of method aiogram.methods.send_venue.SendVenue

reply_video(video: Union[InputFile, str], duration: Optional[int] = None, width: Optional[int] = None, height: Optional[int] = None, thumbnail: Optional[InputFile] = None, caption: Optional[str] = None, parse_mode: Optional[Union[str, Default]] = <Default('parse_mode')>, caption_entities: Optional[List[MessageEntity]] = None, has_spoiler: Optional[bool] = None, supports_streaming: Optional[bool] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, **kwargs: Any) → SendVideo

Shortcut for method *aiogram.methods.send_video.SendVideo* will automatically fill method attributes:

- chat_id
- message_thread_id
- business_connection_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

- **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_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- allow_sending_without_reply Pass True if the message should be sent even if the specified replied-to message is not found

instance of method aiogram.methods.send_video.SendVideo

answer_video(video: Union[InputFile, str], duration: Optional[int] = None, width: Optional[int] = None, height: Optional[int] = None, thumbnail: Optional[InputFile] = None, caption: Optional[str] = None, parse_mode: Optional[Union[str, Default]] = <Default('parse_mode')>, caption_entities: Optional[List[MessageEntity]] = None, has_spoiler: Optional[bool] = None, supports_streaming: Optional[bool] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = None, **kwargs: Any) → SendVideo

Shortcut for method *aiogram.methods.send_video.SendVideo* will automatically fill method attributes:

- chat_id
- message_thread_id
- business_connection_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

- 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_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- **allow_sending_without_reply** Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

instance of method aiogram.methods.send_video.SendVideo

Shortcut for method *aiogram.methods.send_video_note.SendVideoNote* will automatically fill method attributes:

- chat_id
- message_thread_id
- business_connection_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

- 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_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- **allow_sending_without_reply** Pass True if the message should be sent even if the specified replied-to message is not found

instance of method aiogram.methods.send_video_note.SendVideoNote

answer_video_note(video_note: Union[InputFile, str], duration: Optional[int] = None, length: Optional[int] = None, thumbnail: Optional[InputFile] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = None, **kwargs: Any) → SendVideoNote

Shortcut for method *aiogram.methods.send_video_note.SendVideoNote* will automatically fill method attributes:

- chat_id
- message_thread_id
- business_connection_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_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- allow_sending_without_reply Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

instance of method aiogram.methods.send_video_note.SendVideoNote

reply_voice(voice: Union[InputFile, str], caption: Optional[str] = None, parse_mode: Optional[Union[str, Default]] = <Default('parse_mode')>, caption_entities: Optional[List[MessageEntity]] = None, duration: Optional[int] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, **kwargs: Any) → SendVoice

Shortcut for method *aiogram.methods.send_voice.SendVoice* will automatically fill method attributes:

- chat_id
- message_thread_id
- business_connection_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 \gg

- 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_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- 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_voice.SendVoice

answer_voice(*voice*: Union[InputFile, str], caption: Optional[str] = None, parse_mode:

Shortcut for method *aiogram.methods.send_voice.SendVoice* will automatically fill method attributes:

- chat_id
- message_thread_id
- business_connection_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

- **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_parameters Description of the message to reply to
- **reply_markup** Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account
- **allow_sending_without_reply** Pass True if the message should be sent even if the specified replied-to message is not found
- **reply_to_message_id** If the message is a reply, ID of the original message

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_parameters: ReplyParameters | None = None, reply_markup: InlineKeyboardMarkup
| ReplyKeyboardMarkup | None = None, allow_sending_without_reply: bool | None = None,
message_thread_id: int | None = None, business_connection_id: str | None = None, parse_mode:
str | 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_parameters -
- reply_markup -
- allow_sending_without_reply -
- message_thread_id -
- parse_mode -

Returns

copy_to(chat_id: Union[int, str], message_thread_id: Optional[int] = None, caption: Optional[str] = None, parse_mode: Optional[Union[str, Default]] = <Default('parse_mode')>, caption_entities: Optional[List[MessageEntity]] = None, disable_notification: Optional[bool] = None, protect_content: Optional[Union[bool, Default]] = <Default('protect_content')>, reply_parameters: Optional[ReplyParameters] = None, reply_markup: Optional[Union[InlineKeyboardMarkup, ReplyKeyboardMarkup, ReplyKeyboardRemove, ForceReply]] = None, allow_sending_without_reply: Optional[bool] = None, reply_to_message_id: Optional[int] = 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, giveaway messages, giveaway winners 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_parameters Description of the message to reply to
- **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.
- allow_sending_without_reply Pass True if the message should be sent even if the specified replied-to message is not found
- reply_to_message_id If the message is a reply, ID of the original message

Returns

instance of method aiogram.methods.copy_message.CopyMessage

edit_text: str, inline_message_id: Optional[str] = None, parse_mode: Optional[Union[str, Default]] = <Default('parse_mode')>, entities: Optional[List[MessageEntity]] = None, link_preview_options: Optional[LinkPreviewOptions] = None, reply_markup: Optional[InlineKeyboardMarkup] = None, disable_web_page_preview: Optional[Union[bool, Default]] = <Default('link_preview_is_disabled')>, **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*
- link_preview_options Link preview generation options for the message
- reply_markup A JSON-serialized object for an inline keyboard.
- disable_web_page_preview Disables link previews for links in this message

Returns

instance of method aiogram.methods.edit_message_text.EditMessageText

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 and messages with protected content can't be forwarded. On success, the sent *aiogram.types.message.Message* is returned.

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

- **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) \rightarrow 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*) \rightarrow *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*

Shortcutformethodaiogram.methods.edit_message_live_location.EditMessageLiveLocationwill 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

- **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.

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) \rightarrow *StopMessageLiveLocation*

Shortcutformethodaiogram.methods.stop_message_live_location.StopMessageLiveLocationwill 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

 $edit_caption(inline_message_id: Optional[str] = None, caption: Optional[str] = None, parse_mode: Optional[Union[str, Default]] = <Default('parse_mode')>, caption_entities: Optional[List[MessageEntity]] = None, reply_markup: Optional[InlineKeyboardMarkup] = None, **kwargs: Any) → EditMessageCaption$

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

$delete(**kwargs: Any) \rightarrow DeleteMessage$

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) \rightarrow 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*) \rightarrow 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 Otherwise (or if {force_private} flag is set), returns 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

react(*reaction: List*[ReactionTypeEmoji | ReactionTypeCustomEmoji] | None = None, is_big: bool | None = None, **kwargs: Any) → SetMessageReaction

Shortcut for method *aiogram.methods.set_message_reaction.SetMessageReaction* will automatically fill method attributes:

- chat_id
- message_id

Use this method to change the chosen reactions on a message. Service messages can't be reacted to. Automatically forwarded messages from a channel to its discussion group have the same available reactions as messages in the channel. Returns **True** on success.

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

Parameters

- **reaction** A JSON-serialized list of reaction types to set on the message. Currently, as non-premium users, bots can set up to one reaction per message. A custom emoji reaction can be used if it is either already present on the message or explicitly allowed by chat administrators.
- **is_big** Pass True to set the reaction with a big animation

Returns

instance of method aiogram.methods.set_message_reaction.SetMessageReaction

MessageAutoDeleteTimerChanged

class aiogram.types.message_auto_delete_timer_changed.MessageAutoDeleteTimerChanged(*,

message_auto_delete_time: int, **extra_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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

MessageEntity

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), 'blockquote' (block quotation), '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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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*) \rightarrow str

Messageld

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

 $model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

MessageOrigin

class aiogram.types.message_origin.MessageOrigin(**extra_data: Any)

This object describes the origin of a message. It can be one of

- aiogram.types.message_origin_user.MessageOriginUser
- aiogram.types.message_origin_hidden_user.MessageOriginHiddenUser
- aiogram.types.message_origin_chat.MessageOriginChat
- aiogram.types.message_origin_channel.MessageOriginChannel

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

We need to both initialize private attributes and call the user-defined model_post_init method.

MessageOriginChannel

class aiogram.types.message_origin_channel.MessageOriginChannel(*, type: Lit-

eral[MessageOriginType.CHANNEL]

-MessageOriginType.CHANNEL, date: datetime, chat: Chat, message_id: int, author_signature: str | None = None, **extra_data: Any)

The message was originally sent to a channel chat.

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

type: Literal[MessageOriginType.CHANNEL]

Type of the message origin, always 'channel'

date: DateTime

Date the message was sent originally in Unix time

chat: Chat

Channel chat to which the message was originally sent

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(_*ModelMetaclass_context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

message_id: int

Unique message identifier inside the chat

author_signature: str | None

Optional. Signature of the original post author

MessageOriginChat

class aiogram.types.message_origin_chat.MessageOriginChat(*, type:

Literal[MessageOriginType.CHAT] = MessageOriginType.CHAT, date: datetime, sender_chat: Chat, author_signature: str | None = None, **extra_data: Any)

The message was originally sent on behalf of a chat to a group chat.

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

type: Literal[MessageOriginType.CHAT]

Type of the message origin, always 'chat'

date: DateTime

Date the message was sent originally in Unix time

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) \rightarrow None

We need to both initialize private attributes and call the user-defined model_post_init method.

sender_chat: Chat

Chat that sent the message originally

author_signature: str | None

Optional. For messages originally sent by an anonymous chat administrator, original message author signature

MessageOriginHiddenUser

class aiogram.types.message_origin_hidden_user.MessageOriginHiddenUser(*, type: Lit-

eral[MessageOriginType.HIDDEN_USER = MessageOrigin-Type.HIDDEN_USER, date: datetime, sender_user_name: str, **extra_data: Any)

The message was originally sent by an unknown user.

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

type: Literal[MessageOriginType.HIDDEN_USER]

Type of the message origin, always 'hidden_user'

date: datetime

Date the message was sent originally in Unix time

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

 $model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

sender_user_name: str

Name of the user that sent the message originally

MessageOriginUser

class aiogram.types.message_origin_user.MessageOriginUser(*, type:

Literal[MessageOriginType.USER] = MessageOriginType.USER, date: datetime, sender_user: User, **extra_data: Any)

The message was originally sent by a known user.

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

type: Literal[MessageOriginType.USER]

Type of the message origin, always 'user'

date: DateTime

Date the message was sent originally in Unix time

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

sender_user: User

User that sent the message originally

MessageReactionCountUpdated

class aiogram.types.message_reaction_count_updated.MessageReactionCountUpdated(*, chat: Chat,

message_id: int, date: datetime, reactions: List[ReactionCount], **extra_data: Any)

This object represents reaction changes on a message with anonymous reactions.

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

chat: Chat

The chat containing the message

message_id: int

Unique message identifier inside the chat

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

date: DateTime

Date of the change in Unix time

reactions: List[ReactionCount]

List of reactions that are present on the message

MessageReactionUpdated

class aiogram.types.message_reaction_updated.MessageReactionUpdated(*, chat: Chat, message_id:

int, date: datetime, old_reaction: List[ReactionTypeEmoji | ReactionTypeCustomEmoji], new_reaction: List[ReactionTypeEmoji | ReactionTypeCustomEmoji], user: User | None = None, actor_chat: Chat | None = None, **extra_data: Any)

This object represents a change of a reaction on a message performed by a user.

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

chat: Chat

The chat containing the message the user reacted to

message_id: int

Unique identifier of the message inside the chat

date: DateTime

Date of the change in Unix time

old_reaction: List[ReactionTypeEmoji | ReactionTypeCustomEmoji]

Previous list of reaction types that were set by the user

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init($_ModelMetaclass_context: Any$) \rightarrow None

We need to both initialize private attributes and call the user-defined model_post_init method.

new_reaction: List[ReactionTypeEmoji | ReactionTypeCustomEmoji]

New list of reaction types that have been set by the user

user: User | None

Optional. The user that changed the reaction, if the user isn't anonymous

actor_chat: Chat | None

Optional. The chat on behalf of which the reaction was changed, if the user is anonymous

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

height: int

Photo height

file_size: int | None

Optional. File size in bytes

Poll

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'

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

voter_count: int

Number of users that voted for this option

ProximityAlertTriggered

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

 $model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

distance: int

The distance between the users

ReactionCount

**extra data: Any)

Represents a reaction added to a message along with the number of times it was added.

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

type: ReactionTypeEmoji | ReactionTypeCustomEmoji

Type of the reaction

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

total_count: int

Number of times the reaction was added

ReactionType

class aiogram.types.reaction_type.ReactionType(**extra_data: Any)

This object describes the type of a reaction. Currently, it can be one of

- aiogram.types.reaction_type_emoji.ReactionTypeEmoji
- aiogram.types.reaction_type_custom_emoji.ReactionTypeCustomEmoji

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

ReactionTypeCustomEmoji

class aiogram.types.reaction_type_custom_emoji.ReactionTypeCustomEmoji(*, type: Lit-

eral[ReactionTypeType.CUSTOM_EMOJI = ReactionType-Type.CUSTOM_EMOJI, custom_emoji_id: str, **extra_data: Any)

The reaction is based on a custom emoji.

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

type: Literal[ReactionTypeType.CUSTOM_EMOJI]

Type of the reaction, always 'custom_emoji'

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

custom_emoji_id: str

Custom emoji identifier

ReactionTypeEmoji

class aiogram.types.reaction_type_emoji.ReactionTypeEmoji(*, type:

Literal[ReactionTypeType.EMOJI] = ReactionTypeType.EMOJI, emoji: str, **extra_data: Any)

The reaction is based on an emoji.

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

type: Literal[ReactionTypeType.EMOJI]

Type of the reaction, always 'emoji'

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

emoji: str

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.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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 to a message in the same chat and forum topic, sender of the original message.

ReplyKeyboardRemove

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*)

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

We need to both initialize private attributes and call the user-defined model_post_init method.

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 to a message in the same chat and forum topic, sender of the original message.

ReplyParameters

Describes reply parameters for the message that is being sent.

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

message_id: int

Identifier of the message that will be replied to in the current chat, or in the chat *chat_id* if it is specified

chat_id: int | str | None

Optional. If the message to be replied to is from a different chat, unique identifier for the chat or username of the channel (in the format @channelusername). Not supported for messages sent on behalf of a business account.

allow_sending_without_reply: bool | Default | None

Optional. Pass True if the message should be sent even if the specified message to be replied to is not

found. Always False for replies in another chat or forum topic. Always True for messages sent on behalf of a business account.

quote: str | None

Optional. Quoted part of the message to be replied to; 0-1024 characters after entities parsing. The quote must be an exact substring of the message to be replied to, including *bold*, *italic*, *underline*, *strikethrough*, *spoiler*, and *custom_emoji* entities. The message will fail to send if the quote isn't found in the original message.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(_*ModelMetaclass_context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

quote_parse_mode: str | Default | None

Optional. Mode for parsing entities in the quote. See formatting options for more details.

quote_entities: List[MessageEntity] | None

Optional. A JSON-serialized list of special entities that appear in the quote. It can be specified instead of *quote_parse_mode*.

quote_position: int | None

Optional. Position of the quote in the original message in UTF-16 code units

ResponseParameters

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.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

 $model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

retry_after: int | None

Optional. In case of exceeding flood control, the number of seconds left to wait before the request can be repeated

SharedUser

This object contains information about a user that was shared with the bot using a *aiogram.types.* keyboard_button_request_user.KeyboardButtonRequestUser button.

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

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 64-bit integers or double-precision float types are safe for storing these identifiers. 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.

first_name: str | None

Optional. First name of the user, if the name was requested by the bot

```
last_name: str | None
```

Optional. Last name of the user, if the name was requested by the bot

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

username: str | None

Optional. Username of the user, if the username was requested by the bot

photo: List[PhotoSize] | None

Optional. Available sizes of the chat photo, if the photo was requested by the bot

Story

```
class aiogram.types.story.Story(*, chat: Chat, id: int, **extra_data: Any)
```

This object represents a story.

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

chat: Chat

Chat that posted the story

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

id: int

Unique identifier for the story in the chat

SwitchInlineQueryChosenChat

class aiogram.types.switch_inline_query_chosen_chat.SwitchInlineQueryChosenChat(*, query: str

|None| =None, allow_user_chats: bool | None = None, allow bot chats: bool | None = None, al*low_group_chats:* bool | None = None. allow channel chats: bool | None = None,**extra_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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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

TextQuote

class aiogram.types.text_quote.TextQuote(*, text: str, position: int, entities: List[MessageEntity] | None = None, is_manual: bool | None = None, **extra_data: Any)

This object contains information about the quoted part of a message that is replied to by the given message.

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

text: str

Text of the quoted part of a message that is replied to by the given message

position: int

Approximate quote position in the original message in UTF-16 code units as specified by the sender

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

entities: List[MessageEntity] | None

Optional. Special entities that appear in the quote. Currently, only *bold*, *italic*, *underline*, *strikethrough*, *spoiler*, and *custom_emoji* entities are kept in quotes.

is_manual: bool | None

Optional. True, if the quote was chosen manually by the message sender. Otherwise, the quote was added automatically by the server.

User

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.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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.

can_connect_to_business: bool | None

Optional. True, if the bot can be connected to a Telegram Business account to receive its messages. Returned only in *aiogram.methods.get_me.GetMe*.

property full_name: str

property url: str

mention_markdown(*name: str* | *None* = *None*) \rightarrow str

```
mention_html(name: str | None = None) \rightarrow str
```

```
get_profile_photos(offset: int | None = None, limit: int | None = None, **kwargs: Any) \rightarrow 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
```

UserChatBoosts

```
class aiogram.types.user_chat_boosts.UserChatBoosts(*, boosts: List[ChatBoost], **extra_data: Any)
This object represents a list of boosts added to a chat by a user.
```

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

boosts: List[ChatBoost]

The list of boosts added to the chat by the user

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

```
model_post_init(_ModelMetaclass__context: Any) → None
```

We need to both initialize private attributes and call the user-defined model_post_init method.

UserProfilePhotos

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

```
model_post_init(_ModelMetaclass__context: Any) → None
```

We need to both initialize private attributes and call the user-defined model_post_init method.

```
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.

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

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

request_id: int

Identifier of the request

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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.

UsersShared

This object contains information about the users whose identifiers were shared with the bot using a *aiogram*. *types.keyboard_button_request_users.KeyboardButtonRequestUsers* button.

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

request_id: int

Identifier of the request

users: List[SharedUser]

Information about users shared with the bot.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

user_ids: List[int] | None

Identifiers of the shared users. These numbers may have more than 32 significant bits and some programming languages may have difficulty/silent defects in interpreting them. But they have at most 52 significant bits, so 64-bit integers or double-precision float types are safe for storing these identifiers. The bot may not have access to the users and could be unable to use these identifiers, unless the users are already known to the bot by some other means.

Deprecated since version API:7.2: https://core.telegram.org/bots/api-changelog#march-31-2024

Venue

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

VideoChatParticipantsInvited

**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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

 $model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

VideoChatScheduled

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

VideoNote

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

WriteAccessAllowed

This object represents a service message about a user allowing a bot to write messages after adding it to the attachment menu, launching a Web App from a link, or accepting an explicit request from a Web App sent by the method requestWriteAccess.

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

from_request: bool | None

Optional. True, if the access was granted after the user accepted an explicit request from a Web App sent by the method requestWriteAccess

web_app_name: str | None

Optional. Name of the Web App, if the access was granted when the Web App was launched from a link

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

from_attachment_menu: bool | None

Optional. True, if the access was granted when the bot was added to the attachment or side menu

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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) \rightarrow 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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

InlineQueryResultArticle

class aiogram.types.inline_query_result_article.InlineQueryResultArticle(*, type: Lit-

eral[InlineQueryResultType.ARTICLE] = InlineQueryResult-Type.ARTICLE, id: str, title: str, input_message_content: InputTextMessage-Content | InputLocationMessageContent | InputVenueMessage-Content | InputContactMessageContent | InputInvoiceMessageContent, reply_markup: InlineKeyboardMarkup | None = None, url: $str \mid None = None,$ *hide_url: bool* | *None* = *None*, *description*: $str \mid None = None,$ *thumbnail_url: str* | None = None, thumbnail_width: int | None = None,*thumbnail_height:* $int \mid None = None,$ **extra_data: 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

A dictionary of computed field names and their corresponding <i>ComputedFieldInfo</i> objects.
model_post_init ($ModelMetaclass_context: Any$) \rightarrow None
We need to both initialize private attributes and call the user-defined model_post_init method.
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 (*, <i>type: ~typ-</i>

```
ing.Literal[InlineQueryResultType.AUDIO]
= InlineQueryResult-
Type.AUDIO, id: str,
audio_url: str, title: str,
caption: str | None =
None, parse_mode: str |
~aiogram.client.default.Default
|None| =
<Default('parse_mode')>,
caption_entities: ~typ-
ing.List[~aiogram.types.message_entity.Mes.
| None = None, performer:
str \mid None = None,
audio_duration: int | None
= None, reply_markup:
~aiogram.types.inline_keyboard_markup.Inl
| None = None,
input_message_content:
~aiogram.types.input_text_message_content.
~aiogram.types.input_location_message_com
~aiogram.types.input_venue_message_contex
~aiogram.types.input_contact_message_cont
~aiogram.types.input_invoice_message_cont
| 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.

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

type: Literal[InlineQueryResultType.AUDI0]

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 | Default | None

Optional. Mode for parsing entities in the audio caption. See formatting options for more details.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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[InlineQueryResi = Inline-QueryResult-Type.AUDIO, id: str, au*dio_file_id:* str, caption: str | None =None,parse_mode: str | ~aiogram.client.default.Def |None| =<Default('parse_mode')>, caption_entities: $\sim typ$ ing.List[~aiogram.types.me. |None| =None, re*ply_markup:* ~aiogram.types.inline_keyb |None| =None, input_message_content: ~aiogram.types.input_text_r ~aiogram.types.input_locat ~aiogram.types.input_venue ~aiogram.types.input_conta ~aiogram.types.input_invoi |None| =None, **extra_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.

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

type: Literal[InlineQueryResultType.AUDI0]

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

 $model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

parse_mode: str | Default | 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

<pre>class aiogram.types.inline_query_result_cached_document</pre>	.InlineQueryResultCachedDocument(*,
--	-------------------------------------

ultCachedDocument	.(*,
	type:
	~typ-
	ing.Literal[InlineQu _
	_ In-
	line-
	QueryRe-
	sult-
	Type.DOCUMENT,
	id:
	str,
	ti-
	tle:
	str, doc-
	аос- и-
	<i>u-</i> <i>ment_file_id:</i>
	str,
	de-
	scrip-
	tion:
	str
	None
	=
	None,
	cap- tion:
	str
	None
	= None
	None, parse_mode:
	str
	~aiogram.client.def
	None
	=
	<de-< th=""></de-<>
	fault('parse_mode'):
	cap-
	tion_entities: ~typ-
	~1yp- ing.List[~aiogram.t
	None
	=
	None,
	re-
	ply_markup:
	~aiogram.types.inlir
	 None
Chapter 2. Conte	
	None,
	in-
	put_message_conter
	1

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.

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

caption: str | None

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

parse_mode: str | Default | 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

<pre>class aiogram.types.inline_query_result_cached_gif.InlineQueryResultCachedGif</pre>	E(*. type: ~typ-
	ing.Literal[InlineQueryResultTyp
	= Inline-
	QueryResult-
	\tilde{T} ype.GIF, id:
	str, gif_file_id:
	str, title: str
	None = None,
	caption: str
	None = None,
	parse_mode:
	str
	~aiogram.client.default.Default
	$ None = \langle De -$
	fault('parse_mode')>,
	cap-
	tion_entities:
	~typ-
	ing.List[~aiogram.types.message]
	None =
	None,
	reply_markup:
	~aiogram.types.inline_keyboard_
	None =
	None, in-
	put_message_content:
	~aiogram.types.input_text_messa
	1
	~aiogram.types.input_location_n
	1
	~aiogram.types.input_venue_mes
	1
	~aiogram.types.input_contact_m
	~aiogram.types.input_invoice_m
	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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

parse_mode: str | Default | 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_quer	ry_result_cached_mpeg4_gif. InlineQueryResultCachedMp	eg4Gif(*,
--	--	-----------

sultCachedripeg4G11	(*,
	type:
	~typ-
	ing.Literal[InlineQ
	=
	In-
	line-
	QueryRe-
	sult-
	Type.MPEG4_GIF
	id:
	str,
	mpeg4_file_id:
	str,
	ti-
	tle:
	str
	None
	= N/
	None,
	cap-
	tion:
	str
	None
	None =
	– None,
	parse_mode:
	str
	~aiogram.client.de
	None
	=
	- <de-< th=""></de-<>
	fault('parse_mode'
	cap-
	tion_entities:
	~typ-
	ing.List[~aiogram.
	None
	=
	None,
	re-
	ply_markup:
	~aiogram.types.inl
	None
	=
	None,
	in-
	put_message_cont
	~aiogram.types.inp
Chapter 2. Conten	ts_aiogram.types.inp
	~aiogram.types.inf

I

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

parse_mode: str | Default | 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:

to (*, type:
	~typ-
i	ing.Literal[InlineQueryRest
:	= Inline-
	QueryRe-
	sult-
	Туре.РНОТО,
	id: str,
	photo_file_id:
	str, title:
	str None
	= None,
	descrip-
	tion: str
	None =
	None,
	caption:
	str None
:	= None,
	parse_mode:
-	str
	~aiogram.client.default.Def
	None =
	<de-< th=""></de-<>
	fault('parse_mode')>,
	cap-
	tion_entities:
	~typ-
	ing.List[~aiogram.types.me.
	None =
	None, re-
Ì	ply_markup:
	~aiogram.types.inline_keyb
	None =
	None, in-
j	put_message_content:
-	~aiogram.types.input_text_1
	~aiogram.types.input_locat
	- ~aiogram.types.input_venue
	~alogram.types.thput_venue
	• • • • •
	~aiogram.types.input_conta
	l
	~aiogram.types.input_invoi
	None =
	None,
	** <i>ex</i> -
i	tra_data:
	~typ-
	ing.Any)
	· ·
e us	er with an

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

- title: str | None Optional. Title for the result
- description: str | None

Optional. Short description of the result

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

caption: str | None

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

parse_mode: str | Default | 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

<pre>class aiogram.types.inline_query_result_cached_sticker.InlineQueryResul</pre>	ltCachedSticker(*,
	type:
	Lit-
	eral[InlineQueryResul
	=
	In-
	line-
	QueryRe- sult-
	Type.STICKER,
	id:
	str,
	sticker_file_id:
	str,
	re-
	ply_markup:
	In-
	lineKey-
	board-
	Markup
	Nora
	None =
	_ None,
	in-
	put_message_content:
	In-
	put-
	TextMes-
	sage-
	Con-
	tent
	In-
	put-
	Lo- ca-
	tion-
	Mes-
	sage-
	Con-
	tent
	In-
	putV-
	enueMes-
	sage-
	Con- tent
	In-
	put-
	Con-
	tactMes-
	sage-
	Con-
	tent
	In-
	putIn-
232	Chapter 2. ContentseMes-
	sage-
	Con-

tent

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.

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(_*ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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:

50 (;	*, type:
	-typ-
i	ng.Literal[InlineQueryResi
	= Inline-
Ç	QueryRe-
	sult-
7	Type.VIDEO,
i	d: str,
ı	video_file_id:
5	str, title:
5	str, de-
5	scription:
	str None
=	= None,
C	caption:
5	str None
=	= None,
ŀ	parse_mode:
5	str
~	-aiogram.client.default.Def
I	None =
	<de-< th=""></de-<>
f	fault('parse_mode')>,
C	cap-
t	tion_entities:
~	-typ-
i	ing.List[~aiogram.types.mes
I	None =
	None, re-
-	oly_markup:
	-aiogram.types.inline_keyb
I	None =
1	None, in-
-	out_message_content:
~	-aiogram.types.input_text_r
I	
~	-aiogram.types.input_locati
I	
~	-aiogram.types.input_venue
I	
~	-aiogram.types.input_conta
	-aiogram.types.input_invoid
	None =
	None,
,	**ex-
t	ra_data:
	-typ-
i	ng.Any)
nt by	y the user

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.VIDE0]

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

caption: str | None

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

parse_mode: str | Default | 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[InlineQueryResi = InlineQueryResult-Type.VOICE, *id: str*, *voice_file_id:* str, title: str, caption: str | None = None, parse_mode: str | ~aiogram.client.default.Def |None| =<Default('parse_mode')>, caption_entities: ~typing.List[~aiogram.types.me. |None| =None, reply_markup: ~aiogram.types.inline_keyb |None| =None, input_message_content: ~aiogram.types.input_text_r ~aiogram.types.input_locat ~aiogram.types.input_venue ~aiogram.types.input_conta ~aiogram.types.input_invoi |None| =None. **extra_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.

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

parse_mode: str | Default | 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: Lit-

eral[InlineQueryResultType.CONTACT = InlineQueryResult-Type.CONTACT, id: *str*, *phone_number*: str, first_name: str, *last_name: str* | *None* = None, vcard: str | None = None, reply_markup: InlineKeyboardMarkup | None = None, input message content: InputTextMessage-Content | InputLocationMessageContent InputVenueMessage-Content | InputContactMessageContent | InputInvoiceMessageContent | None = *None*, *thumbnail_url*: $str \mid None = None,$ thumbnail_width: int | None = None,*thumbnail_height:* $int \mid None = None,$ **extra_data: 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.

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

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(_*ModelMetaclass_context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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.D = InlineQueryResult-Type.DOCUMENT, id: str, title: str, document_url: str, mime_type: str, caption: str | None =None,*parse_mode: str* | ~aiogram.client.default.Default $|None = \langle De$ fault('parse_mode')>, caption_entities: ~typing.List[~aiogram.types.message_ent |None = None,description: str | None = None, reply_markup: ~aiogram.types.inline_keyboard_max | None = None, input_message_content: ~aiogram.types.input_text_message_ ~aiogram.types.input_location_mess ~aiogram.types.input_venue_messag ~aiogram.types.input_contact_messa ~aiogram.types.input_invoice_messa | None = None,thumbnail url: str | None = None,thumbnail width: $int \mid None = None,$ *thumbnail_height:* $int \mid None = None,$ **extra data:

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.

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

~typing.Any)

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 | Default | None

Optional. Mode for parsing entities in the document caption. See formatting options for more details.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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: Lit-

eral[InlineQueryResultType.GAME]
= InlineQueryResultType.GAME, id: str,
game_short_name: str,
reply_markup:
InlineKeyboardMarkup |
None = None, **extra_data:
Any)

Represents a Game.

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

game_short_name: str

Short name of the game

reply_markup: InlineKeyboardMarkup | None

Optional. Inline keyboard attached to the message

InlineQueryResultGif

| None = None, **extra_data:

```
class aiogram.types.inline_query_result_gif.InlineQueryResultGif(*, type: ~typ-
                                                                                 ing.Literal[InlineQueryResultType.GIF]
                                                                                 = InlineQueryResultType.GIF,
                                                                                 id: str, gif_url: str,
                                                                                 thumbnail_url: str, gif_width:
                                                                                 int | None = None, gif_height:
                                                                                 int \mid None = None,
                                                                                 gif_duration: int | None =
                                                                                 None, thumbnail_mime_type:
                                                                                 str | None = None, title: str |
                                                                                 None = None, caption: str |
                                                                                 None = None, parse_mode: str
                                                                                 ~aiogram.client.default.Default
                                                                                 |None| =
                                                                                 <Default('parse_mode')>,
                                                                                 caption_entities: ~typ-
                                                                                 ing.List[~aiogram.types.message_entity.MessageE
                                                                                 | None = None, reply_markup:
                                                                                 ~aiogram.types.inline_keyboard_markup.InlineKe
                                                                                 | None = None,
                                                                                 input_message_content:
                                                                                 ~aiogram.types.input_text_message_content.Input
                                                                                 ~aiogram.types.input_location_message_content.l
                                                                                 ~aiogram.types.input_venue_message_content.Inp
                                                                                 ~aiogram.types.input_contact_message_content.li
                                                                                 ~aiogram.types.input_invoice_message_content.lr
```

~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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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 | Default | 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: Lit-

eral[InlineQueryResultType.LOCAT] = InlineQueryResult-Type.LOCATION, *id: str, latitude:* float, longitude: float, title: str, horizontal_accuracy: float | None = None,live_period: int | *None* = *None*, heading: int | *None* = *None*, proxim*ity_alert_radius:* $int \mid None = None$, reply_markup: InlineKeyboard-Markup | None = None, in*put_message_content:* InputTextMessageContent InputLocation-MessageContent | InputVenueMessageContent | InputContactMessageContent InputInvoiceMessageContent *None* = *None*, thumbnail url: str | None = None,thumbnail width: $int \mid None = None,$ *thumbnail_height:* $int \mid None = None,$ **extra data: 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.

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.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(_*ModelMetaclass_context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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: ~typ-

ing.Literal[InlineQueryResultType. = InlineQueryResult-Type.MPEG4_GIF, id: str, mpeg4_url: str, thumbnail_url: str, mpeg4_width: $int \mid None =$ None, *mpeg4_height:* $int \mid None =$ None, mpeg4_duration: $int \mid None =$ None, thumbnail_mime_type: $str \mid None =$ None, title: str | None = None, caption: str | None = None, *parse_mode: str* | ~aiogram.client.default.Default $|None = \langle De$ fault('parse_mode')>, caption_entities: ~typing.List[~aiogram.types.message_e | None = None,reply_markup: ~aiogram.types.inline_keyboard_m | None = None,input_message_content: ~aiogram.types.input_text_message ~aiogram.types.input_location_mes Ι ~aiogram.types.input_venue_messa Ι ~aiogram.types.input_contact_mess Т ~aiogram.types.input_invoice_mess | 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
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
     A dictionary of computed field names and their corresponding ComputedFieldInfo objects.
model_post_init(_ModelMetaclass__context: Any) → None
     We need to both initialize private attributes and call the user-defined model_post_init method.
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 | Default | 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: ~typ-
```

ing.Literal[InlineQueryResultType.PHOTO] = InlineQueryResult-Type.PHOTO, id: str, photo_url: str, thumbnail url: str. *photo_width: int | None = None*, *photo_height: int* | *None* = *None*, *title*: *str* \mid *None* = *None*, *description*: $str \mid None = None,$ caption: str | None = None, parse mode: str | ~aiogram.client.default.Default |None| =<Default('parse_mode')>, caption_entities: ~typing.List[~aiogram.types.message_entity.Mes. | None = None,reply_markup: ~aiogram.types.inline_keyboard_markup.Inl | None = None,input message content: ~aiogram.types.input_text_message_content. Т ~aiogram.types.input_location_message_com T ~aiogram.types.input_venue_message_conten ~aiogram.types.input_contact_message_cont T ~aiogram.types.input_invoice_message_cont | 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.PHOT0]

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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 | Default | 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: Lit-

eral[InlineQueryResultType.VENUE] = InlineQueryResult-Type.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: InlineKeyboardMarkup | None = None, *input_message_content:* InputTextMessageContent InputLocationMessage-Content | InputVenueMessageContent InputContactMessageContent | InputInvoiceMessageContent | None = None, *thumbnail_url: str* | *None* = None, thumbnail width: $int \mid None = None,$ *thumbnail_height: int* | None = None, **extra_data: Any)

Represents a venue. By default, the venue will be sent by the user. Alternatively, you can use *in-put_message_content* to send a message with the specified content instead of the venue.

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'.)

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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: ~typ-

ing.Literal[InlineQueryResultType.VIDEO] = InlineQueryResult-Type.VIDEO, id: str, video_url: str, mime_type: str, thumbnail url: str, *title: str, caption: str* None = None, parse_mode: str | ~aiogram.client.default.Default |None| =<*Default('parse_mode')*>, caption_entities: ~typing.List[~aiogram.types.message_entity.Mes. | 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.Inl | None = None.input_message_content: ~aiogram.types.input_text_message_content. ~aiogram.types.input_location_message_com ~aiogram.types.input_venue_message_conten ~aiogram.types.input_contact_message_cont ~aiogram.types.input_invoice_message_cont | 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.VIDE0]

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 | Default | None

Optional. Mode for parsing entities in the video caption. See formatting options for more details.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

caption_entities: List[MessageEntity] | None

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

- 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 InlineQueryRe-sultVideo 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: ~typ-

```
ing.Literal[InlineQueryResultType.VOICE]
= InlineQueryResult-
Type.VOICE, id: str,
voice_url: str, title: str,
caption: str | None =
None, parse mode: str |
~aiogram.client.default.Default
|None| =
<Default('parse_mode')>,
caption_entities: ~typ-
ing.List[~aiogram.types.message_entity.Mes.
| None = None,
voice_duration: int | None
= None, reply_markup:
~aiogram.types.inline_keyboard_markup.Inl
| None = None,
input message content:
~aiogram.types.input_text_message_content.
~aiogram.types.input_location_message_com
Т
~aiogram.types.input_venue_message_conten
~aiogram.types.input_contact_message_cont
Т
~aiogram.types.input_invoice_message_cont
| 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.

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

parse_mode: str | Default | 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

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.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

 $model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(_*ModelMetaclass_context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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:* $int \mid None =$ None, suggested_tip_amounts: *List[int]* | *None* = None,provider_data: $str \mid None =$ None, photo_url: str | None = None, photo_size: int | None = None, *photo_width:* $int \mid None =$ None, photo_height: $int \mid None =$ None. need_name: $bool \mid None =$ None, need_phone_number: $bool \mid None =$ None. need_email: $bool \mid None =$ None. need_shipping_address: $bool \mid None =$ None, send_phone_number_to_provider: $bool \mid None =$ None, send_email_to_provider: $bool \mid None =$ None, is_flexible: bool | None = None,**extra_data:

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

Any)

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.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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, horizon*tal_accuracy:* float | None = None, *live_period:* $int \mid None =$ None. heading: int | None =None, proximity alert radius: $int \mid None =$ 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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass_context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

We need to both initialize private attributes and call the user-defined model_post_init method.

InputTextMessageContent

class aiogram.types.input_text_message_content.InputTextMessageContent(*, message_text: str,

parse mode: str ~aiogram.client.default.Default $|None = \langle De$ fault('parse mode')>, entities: ~typing.List[~aiogram.types.message_entity.M | None = None,*link_preview_options:* ~aiogram.types.link_preview_options.Link | None = None, disable_web_page_preview: bool | ~aiogram.client.default.Default $|None = \langle De$ fault('disable_web_page_preview')>, **extra_data: ~typing.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 | Default | 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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

link_preview_options: LinkPreviewOptions | None

Optional. Link preview generation options for the message

disable_web_page_preview: bool | Default | None

Optional. Disables link previews for links in the sent message

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

 $model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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

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.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

Stickers

InputSticker

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* »

format: str

Format of the added sticker, must be one of 'static' for a **.WEBP** or **.PNG** image, 'animated' for a **.TGS** animation, 'video' for a **WEBM** video

emoji_list: List[str]

List of 1-20 emoji associated with the sticker

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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

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.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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*) \rightarrow *SetStickerPositionInSet*

Shortcutformethodaiogram.methods.set_sticker_position_in_set.SetStickerPositionInSetwill 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

$delete_from_set(**kwargs: Any) \rightarrow DeleteStickerFromSet$

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, stickers: List[Sticker], thumbnail: PhotoSize | None = None, is_animated: bool | None = None, is_video: bool | 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'

stickers: List[Sticker]

List of all set stickers

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init($ModelMetaclass_context: Any$) \rightarrow None

We need to both initialize private attributes and call the user-defined model_post_init method.

thumbnail: PhotoSize | None

Optional. Sticker set thumbnail in the .WEBP, .TGS, or .WEBM format

is_animated: bool | None

True, if the sticker set contains animated stickers

Deprecated since version API:7.2: https://core.telegram.org/bots/api-changelog#march-31-2024

is_video: bool | None

True, if the sticker set contains video stickers

Deprecated since version API:7.2: https://core.telegram.org/bots/api-changelog#march-31-2024

Telegram Passport

EncryptedCredentials

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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: $str \mid None =$ *None*, *phone_number*: $str \mid None = None$, 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 only 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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

files: List[PassportFile] | None

Optional. Array of encrypted files with documents provided by the user; available only 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 only 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 only 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 if requested 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

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

We need to both initialize private attributes and call the user-defined model_post_init method.

PassportElementErrorDataField

class aiogram.types.passport_element_error_data_field.PassportElementErrorDataField(*,

 ,
source:
Lit-
eral[PassportElementEr
=
Pass-
portEle-
mentEr-
rorType.DATA,
type:
str,
field_name:
str,
data_hash:
str,
mes-
sage:
str,
**ex-
tra_data:
Any)
1 1

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

data_hash: str

Base64-encoded data hash

message: str

Error message

PassportElementErrorFile

```
class aiogram.types.passport_element_error_file.PassportElementErrorFile(*, source: Lit-
```

eral[PassportElementErrorType.FILE]
=
PassportElementErrorType.FILE, type:
str, file_hash: str,
message: str,
**extra_data: 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'

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

file_hash: str

Base64-encoded file hash

message: str

Error message

PassportElementErrorFiles

class aiogram.types.passport_element_error_files.PassportElementErrorFiles(*, source: Lit-

eral[PassportElementErrorType.FIL] = PassportElementErrorType.FILES, type: str, file_hashes: List[str], message: str, **extra_data: 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'

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

file_hashes: List[str]

List of base64-encoded file hashes

message: str

Error message

PassportElementErrorFrontSide

s aiogram.types.passport_element_error_front_side. PassportElementEr	<pre>rorFrontSide(*,</pre>
	source:
	Lit-
	eral[PassportElementEr
	=
	Pass-
	portEle-
	mentEr-
	rorType.FRONT_SIDE,
	type:
	str,
	file_hash:
	str,
	mes-
	sage:
	str,
	** <i>ex</i> -
	tra_data:
	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[PassportElementErrorType.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'

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

We need to both initialize private attributes and call the user-defined model_post_init method.

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:
Lit-
eral[PassportEleme
=
Pass-
portEle-
mentEr-
rorType.REVERSE_
type:
str,
file_hash:
str,
mes-
sage:
str,
**ex-
tra_data:
Any)
- 1

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'

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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: Lit-

eral[PassportElementErrorType.S. = PassportElementErrorType.SELFIE, type: str, file_hash: str, message: str, **extra_data: 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[PassportElementErrorType.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'

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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: Literal[Pass PassportElementErrorType.7 type: str, file_hash. str, message: str, **extra_data: 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[PassportElementErrorType.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'

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

file_hash: str

Base64-encoded file hash

message: str

Error message

PassportElementErrorTranslationFiles

class aiogram.types.passport_element_error_translation_files.PassportElementErrorTranslationFiles(*,

Jiii 1100(,
source
Lit-
eral[Pa
=
Pass-
portEle
mentE
rorTyp
type:
str,
file_ha
List[st
mes-
sage:
str,
**ex-
tra_da
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'

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

file_hashes: List[str]

List of base64-encoded file hashes

message: str

Error message

PassportElementErrorUnspecified

class aiogram.types.passport_element_error_unspecified.PassportElementErrorUnspecified(*,

	source:
	Lit-
	eral[PassportElemen
	=
	Pass-
	portEle-
	mentEr-
	rorType.UNSPECIFI
	type:
	str,
	el-
	е-
	ment_hash:
	str,
	mes-
	sage:
	str,
	**ex-
	tra_data:
	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[PassportElementErrorType.UNSPECIFIED]

Error source, must be unspecified

type: str

Type of element of the user's Telegram Passport which has the issue

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

element_hash: str

Base64-encoded element hash

message: str

Error message

PassportFile

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.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

 $model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

file_size: int

File size in bytes

file_date: DateTime

Unix time when the file was uploaded

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_post_init($ModelMetaclass_context: Any$) \rightarrow None

We need to both initialize private attributes and call the user-defined model_post_init method.

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

 $model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

email: str | None

Optional. User email

shipping_address: ShippingAddress | None

Optional. User shipping address

PreCheckoutQuery

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).

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

$model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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) \rightarrow 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

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

street_line1: str

First line for the address

street_line2: str

Second line for the address

post_code: str

Address post code

ShippingOption

This object represents one shipping option.

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

id: str

Shipping option identifier

title: str

Option title

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

 $model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

prices: List[LabeledPrice]

List of price portions

ShippingQuery

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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 AnswerShippingQuery aiogram.methods.answer_shipping_query.

SuccessfulPayment

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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

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, business connection:* BusinessConnection | None = None, business message: Message | None = None, edited_business_message: Message | None = None, *deleted_business_messages:* BusinessMessagesDeleted | *None = None*, *message_reaction:* MessageReactionUpdated | *None = None*, message_reaction_count: MessageReactionCountUpdated | 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*, *chat boost*: ChatBoostUpdated | *None* = None, removed chat boost: ChatBoostRemoved | 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 identifier 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. This update may at times be triggered by changes to message fields that are either unavailable or not actively used by your bot.

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. This update may at times be triggered by changes to message fields that are either unavailable or not actively used by your bot.

business_connection: BusinessConnection | None

Optional. The bot was connected to or disconnected from a business account, or a user edited an existing connection with the bot

business_message: Message | None

Optional. New non-service message from a connected business account

edited_business_message: Message | None

Optional. New version of a message from a connected business account

deleted_business_messages: BusinessMessagesDeleted | None

Optional. Messages were deleted from a connected business account

message_reaction: MessageReactionUpdated | None

Optional. A reaction to a message was changed by a user. The bot must be an administrator in the chat and must explicitly specify "message_reaction" in the list of *allowed_updates* to receive these updates. The update isn't received for reactions set by bots.

message_reaction_count: MessageReactionCountUpdated | None

Optional. Reactions to a message with anonymous reactions were changed. The bot must be an administrator in the chat and must explicitly specify "message_reaction_count" in the list of *allowed_updates* to receive these updates. The updates are grouped and can be sent with delay up to a few minutes.

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init($ModelMetaclass_context: Any$) \rightarrow None

We need to both initialize private attributes and call the user-defined model_post_init method.

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 manually 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.

chat_boost: ChatBoostUpdated | None

Optional. A chat boost was added or changed. The bot must be an administrator in the chat to receive these updates.

removed_chat_boost: ChatBoostRemoved | None

Optional. A boost was removed from a chat. The bot must be an administrator 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

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_post_init($ModelMetaclass_context: Any$) \rightarrow None

We need to both initialize private attributes and call the user-defined model_post_init method.

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

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

Game

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.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

```
model_post_init(ModelMetaclass\_context: Any) \rightarrow None
```

We need to both initialize private attributes and call the user-defined model_post_init method.

score: int Score

2.3.4 Methods

Here is list of all available API methods:

Stickers

addStickerToSet

Returns: bool

Use this method to add a new sticker to a set created by the bot. Emoji sticker sets can have up to 200 stickers. Other 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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init($ModelMetaclass_context: Any$) \rightarrow None

We need to both initialize private attributes and call the user-defined model_post_init method.

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

sticker_format: str | None

Format of stickers in the set, must be one of 'static', 'animated', 'video'

Deprecated since version API:7.2: https://core.telegram.org/bots/api-changelog#march-31-2024

Usage

As bot method

result: bool = await bot.create_new_sticker_set(...)

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

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

Usage

As bot method

result: bool = await bot.delete_sticker_from_set(...)

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

Usage

As bot method

result: bool = await bot.delete_sticker_set(...)

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]

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]

A JSON-serialized list of custom emoji identifiers. At most 200 custom emoji identifiers can be specified.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(_*ModelMetaclass_context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

Usage

As bot method

result: List[Sticker] = await bot.get_custom_emoji_stickers(...)

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

```
model_post_init(ModelMetaclass\_context: Any) \rightarrow None
```

We need to both initialize private attributes and call the user-defined model_post_init method.

Usage

As bot method

result: StickerSet = await bot.get_sticker_set(...)

Method as object

- from aiogram.methods.get_sticker_set import GetStickerSet
- alias: from aiogram.methods import GetStickerSet

result: StickerSet = await bot(GetStickerSet(...))

replaceStickerInSet

Returns: bool

Use this method to replace an existing sticker in a sticker set with a new one. The method is equivalent to calling aiogram.methods.delete_sticker_from_set.DeleteStickerFromSet, then aiogram.methods.add_sticker_to_set.AddStickerToSet, then aiogram.methods.set_sticker_position_in_set. SetStickerPositionInSet. Returns True on success.

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

user_id: int

User identifier of the sticker set owner

name: str

Sticker set name

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

old_sticker: str

File identifier of the replaced sticker

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 remains unchanged.

Usage

As bot method

result: bool = await bot.replace_sticker_in_set(...)

Imports:

- from aiogram.methods.replace_sticker_in_set import ReplaceStickerInSet
- alias: from aiogram.methods import ReplaceStickerInSet

With specific bot

result: bool = await bot(ReplaceStickerInSet(...))

As reply into Webhook in handler

return ReplaceStickerInSet(...)

sendSticker

Returns: Message

class aiogram.methods.send_sticker.SendSticker(*, chat_id: int | str, sticker:

~aiogram.types.input_file.InputFile | str, business_connection_id: str | None = None, message thread id: int | None = None, emoji: str | *None* = *None*, *disable notification: bool* | *None* = *None*, *protect_content: bool* | ~aiogram.client.default.Default | None = *<Default('protect_content')>, reply_parameters:* ~aiogram.types.reply_parameters.ReplyParameters *None* = *None*, *reply_markup*: ~aiogram.types.inline_keyboard_markup.InlineKeyboardMarkup ~aiogram.types.reply_keyboard_markup.ReplyKeyboardMarkup ~aiogram.types.reply_keyboard_remove.ReplyKeyboardRemove | ~aiogram.types.force_reply.ForceReply | None = *None*, *allow_sending_without_reply: bool* | *None* = *None*, *reply_to_message_id: int* | *None* = *None*, **extra_data: ~typing.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, .TGS, or .WEBM sticker using multipart/form-data. *More information on Sending Files* ». Video and animated stickers can't be sent via an HTTP URL.

business_connection_id: str | None

Unique identifier of the business connection on behalf of which the message will be sent

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.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

 $model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

protect_content: bool | Default | None

Protects the contents of the sent message from forwarding and saving

reply_parameters: ReplyParameters | None

Description of the message to reply to

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. Not supported for messages sent on behalf of a business account.

allow_sending_without_reply: bool | None

Pass True if the message should be sent even if the specified replied-to message is not found

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

reply_to_message_id: int | None

If the message is a reply, ID of the original message

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

Usage

As bot method

result: Message = await bot.send_sticker(...)

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_join_request.ChatJoinRequest.answer_sticker()
- aiogram.types.chat_join_request.ChatJoinRequest.answer_sticker_pm()
- aiogram.types.chat_member_updated.ChatMemberUpdated.answer_sticker()

setCustomEmojiStickerSetThumbnail

Returns: bool

class aiogram.methods.set_custom_emoji_sticker_set_thumbnail.SetCustomEmojiStickerSetThumbnail(*,

• /
name:
str,
cus-
tom_emoj
str
I
None
=
None,
** <i>ex</i> -
tra_data:
Any)
2 -

Use this method to set the thumbnail of a custom emoji sticker set. Returns True on success.

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

name: str

Sticker set name

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

```
model_post_init(_ModelMetaclass__context: Any) → None
```

We need to both initialize private attributes and call the user-defined model_post_init method.

```
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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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

**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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

format: str

Format of the thumbnail, must be one of 'static' for a **.WEBP** or **.PNG** image, 'animated' for a **.TGS** animation, or 'video' for a **WEBM** video

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#animatedsticker-requirements <https://core.telegram.org/stickers#animated-stickerrequirements>`_`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 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
```

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

```
model_post_init(\_ModelMetaclass\_context: Any) \rightarrow None
```

We need to both initialize private attributes and call the user-defined model_post_init method.

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

Use this method to upload a file with a sticker for later use in the *aiogram.methods. create_new_sticker_set.CreateNewStickerSet, aiogram.methods.add_sticker_to_set. AddStickerToSet,* or *aiogram.methods.replace_sticker_in_set.ReplaceStickerInSet* 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 »*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

$model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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

- from aiogram.methods.upload_sticker_file import UploadStickerFile
- alias: from aiogram.methods import UploadStickerFile

result: File = await bot(UploadStickerFile(...))

Available methods

answerCallbackQuery

Returns: bool

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*.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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)

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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

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)

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

 $model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

Usage

As bot method

result: bool = await bot.close(...)

Method as object

- from aiogram.methods.close import Close
- alias: from aiogram.methods import Close

```
result: bool = await bot(Close(...))
```

As reply into Webhook in handler

return Close(...)

closeForumTopic

Returns: bool

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)

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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

- from aiogram.methods.close_forum_topic import CloseForumTopic
- alias: from aiogram.methods import CloseForumTopic

result: bool = await bot(CloseForumTopic(...))

As reply into Webhook in handler

return CloseForumTopic(...)

closeGeneralForumTopic

Returns: bool

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)

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

Usage

As bot method

result: bool = await bot.close_general_forum_topic(...)

Method as object

- from aiogram.methods.close_general_forum_topic import CloseGeneralForumTopic
- alias: from aiogram.methods import CloseGeneralForumTopic

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* | ~aiogram.client.default.Default | None = <Default('parse_mode')>, caption_entities: ~typing.List[~aiogram.types.message_entity.MessageEntity] | None = None, disable_notification: bool | None = *None*, *protect_content: bool* | ~aiogram.client.default.Default | None = <*Default('protect_content')*>, reply_parameters: ~aiogram.types.reply_parameters.ReplyParameters | None = None, reply_markup: ~aiogram.types.inline_keyboard_markup.InlineKeyboardMarkup ~aiogram.types.reply_keyboard_markup.ReplyKeyboardMarkup ~aiogram.types.reply_keyboard_remove.ReplyKeyboardRemove |~aiogram.types.force reply.ForceReply | None = *None*, *allow_sending_without_reply: bool* | *None* = *None*, *reply_to_message_id: int* | *None* = *None*, ***extra_data:* ~*typing.Any*)

Use this method to copy messages of any kind. Service messages, giveaway messages, giveaway winners 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 | Default | 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*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

disable_notification: bool | None

Sends the message silently. Users will receive a notification with no sound.

protect_content: bool | Default | None

Protects the contents of the sent message from forwarding and saving

reply_parameters: ReplyParameters | None

Description of the message to reply to

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.

allow_sending_without_reply: bool | None

Pass True if the message should be sent even if the specified replied-to message is not found

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

reply_to_message_id: int | None

If the message is a reply, ID of the original message

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

Usage

As bot method

result: MessageId = await bot.copy_message(...)

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()

copyMessages

Returns: List[MessageId]

Use this method to copy messages of any kind. If some of the specified messages can't be found or copied, they are skipped. Service messages, giveaway messages, giveaway winners 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_messages*. *ForwardMessages*, but the copied messages don't have a link to the original message. Album grouping is kept for copied messages. On success, an array of *aiogram.types.message_id.MessageId* of the sent messages is returned.

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

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 messages were sent (or channel username in the format @channelusername)

message_ids: List[int]

A JSON-serialized list of 1-100 identifiers of messages in the chat *from_chat_id* to copy. The identifiers must be specified in a strictly increasing order.

message_thread_id: int | None

Unique identifier for the target message thread (topic) of the forum; for forum supergroups only

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

disable_notification: bool | None

Sends the 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

remove_caption: bool | None

Pass True to copy the messages without their captions

Usage

As bot method

result: List[MessageId] = await bot.copy_messages(...)

Method as object

Imports:

- from aiogram.methods.copy_messages import CopyMessages
- alias: from aiogram.methods import CopyMessages

With specific bot

result: List[MessageId] = await bot(CopyMessages(...))

As reply into Webhook in handler

return CopyMessages(...)

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_post_init(_*ModelMetaclass_context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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

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)

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

$model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

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(DeclineChatJoinRequest(...))

As reply into Webhook in handler

return DeclineChatJoinRequest(...)

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)

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

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)

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

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

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)

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

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 | 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.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init($ModelMetaclass_context: Any$) \rightarrow None

We need to both initialize private attributes and call the user-defined model_post_init method.

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

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

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)

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

```
model_post_init(_ModelMetaclass__context: Any) \rightarrow None
```

We need to both initialize private attributes and call the user-defined model_post_init method.

name: str

New topic name, 1-128 characters

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

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#exportchatinvitelink

chat_id: int | str

Unique identifier for the target chat or username of the target channel (in the format @channelusername)

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

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 |

~aiogram.client.default.Default | None =

<Default('protect_content')>, **extra_data:

~typing.Any)
```

Use this method to forward messages of any kind. Service messages and messages with protected content 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*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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 | Default | 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()

forwardMessages

Returns: List[MessageId]

Use this method to forward multiple messages of any kind. If some of the specified messages can't be found or forwarded, they are skipped. Service messages and messages with protected content can't be forwarded. Album grouping is kept for forwarded messages. On success, an array of *aiogram.types.message_id.MessageId* of the sent messages is returned.

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

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 messages were sent (or channel username in the format @channelusername)

message_ids: List[int]

A JSON-serialized list of 1-100 identifiers of messages in the chat *from_chat_id* to forward. The identifiers must be specified in a strictly increasing order.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(_*ModelMetaclass_context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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 messages silently. Users will receive a notification with no sound.

protect_content: bool | None

Protects the contents of the forwarded messages from forwarding and saving

As bot method

result: List[MessageId] = await bot.forward_messages(...)

Method as object

Imports:

- from aiogram.methods.forward_messages import ForwardMessages
- alias: from aiogram.methods import ForwardMessages

With specific bot

result: List[MessageId] = await bot(ForwardMessages(...))

As reply into Webhook in handler

return ForwardMessages(...)

getBusinessConnection

Returns: BusinessConnection

Use this method to get information about the connection of the bot with a business account. Returns a *aiogram*. *types.business_connection.BusinessConnection* object on success.

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

business_connection_id: str

Unique identifier of the business connection

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

As bot method

result: BusinessConnection = await bot.get_business_connection(...)

Method as object

Imports:

- from aiogram.methods.get_business_connection import GetBusinessConnection
- alias: from aiogram.methods import GetBusinessConnection

With specific bot

result: BusinessConnection = await bot(GetBusinessConnection(...))

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. 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)

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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]]

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)

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

```
model_post_init(_ModelMetaclass__context: Any) → None
```

We need to both initialize private attributes and call the user-defined model_post_init method.

Usage

As bot method

Method as object

Imports:

- from aiogram.methods.get_chat_administrators import GetChatAdministrators
- alias: from aiogram.methods import GetChatAdministrators

With specific bot

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)

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

 $model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

user_id: int

Unique identifier of the target user

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

As shortcut from received object

aiogram.types.chat.Chat.get_member()

getChatMemberCount

Returns: int

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)

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

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]

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

 $model_post_init(ModelMetaclass_context: Any) \rightarrow None$

As bot method

Method as object

Imports:

- from aiogram.methods.get_chat_menu_button import GetChatMenuButton
- alias: from aiogram.methods import GetChatMenuButton

With specific bot

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>, 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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

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]

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

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

```
model_post_init(\_ModelMetaclass\_context: Any) \rightarrow None
```

We need to both initialize private attributes and call the user-defined model_post_init method.

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

We need to both initialize private attributes and call the user-defined model_post_init method.

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 (*, <i>scope:</i> BotCommandScopeDefault			
BotCommandScopeA	AllPrivateChats		
BotCommandScopeA	AllGroupChats		
BotCommandScopeA	AllChatAdministrators		
BotCommandScopeC	Chat		
BotCommandScopeC	ChatAdministrators		
BotCommandScopeC	ChatMember $None = None$,		
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.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

```
model_post_init(ModelMetaclass\_context: Any) \rightarrow None
```

We need to both initialize private attributes and call the user-defined model_post_init method.

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(*,

	5 - 7 -	_	- 0 ,		• /
					for_channels:
					bool
					1
					None
					=
					None,
					** <i>ex</i> -
					tra_data:
					Any)
s method to	get the current	default administr	ator rights of the bot	t Returns aiogram type	c

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.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

```
model_post_init(ModelMetaclass\_context: Any) \rightarrow None
```

We need to both initialize private attributes and call the user-defined model_post_init method.

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

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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(...))

getUserChatBoosts

Returns: UserChatBoosts

Use this method to get the list of boosts added to a chat by a user. Requires administrator rights in the chat. Returns a *aiogram.types.user_chat_boosts.UserChatBoosts* object.

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

chat_id: int | str

Unique identifier for the chat or username of the channel (in the format @channelusername)

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

user_id: int

Unique identifier of the target user

Usage

As bot method

result: UserChatBoosts = await bot.get_user_chat_boosts(...)

Method as object

Imports:

- from aiogram.methods.get_user_chat_boosts import GetUserChatBoosts
- alias: from aiogram.methods import GetUserChatBoosts

With specific bot

result: UserChatBoosts = await bot(GetUserChatBoosts(...))

getUserProfilePhotos

Returns: UserProfilePhotos

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.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass_context: Any*) \rightarrow None

We need to both initialize private attributes and call the user-defined model_post_init method.

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

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)

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

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)

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

 $model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

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

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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.

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.chat.Chat.pin_message()
- aiogram.types.message.Message.pin()

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_delete_messages: bool | None = None,can manage video chats: bool *None* = *None*, *can restrict members*: $bool \mid None = None,$ can promote members: bool | None = None, can_change_info: bool | None = None, can_invite_users: bool | None = None, can_post_stories: bool | None = None, can_edit_stories: bool | None = None, can_delete_stories: bool | None = None, can_post_messages: $bool \mid None = None,$ can_edit_messages: 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, get boost list, see hidden supergroup and channel members, report spam messages and ignore slow mode. Implied by any other administrator privilege.

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, or access supergroup statistics

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(<u>ModelMetaclass_context</u>: Any) \rightarrow None We need to both initialize private attributes and call the user-defined model_post_init method.

can_invite_users: bool | None

Pass True if the administrator can invite new users to the chat

can_post_stories: bool | None

Pass True if the administrator can post stories to the chat

can_edit_stories: bool | None

Pass True if the administrator can edit stories posted by other users

can_delete_stories: bool | None

Pass True if the administrator can delete stories posted by other users

can_post_messages: bool | None

Pass True if the administrator can post messages in the channel, or access channel statistics; for channels only

can_edit_messages: bool | None

Pass True if the administrator can edit messages of other users and can pin messages; for channels only

can_pin_messages: bool | None

Pass True if the administrator can pin messages; for supergroups only

can_manage_topics: bool | None

Pass True if the user is allowed to create, rename, close, and reopen forum topics; for 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

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)

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(_*ModelMetaclass_context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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

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)

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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

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)

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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	(*, cl	hat_id.	: int str,	animation:
-------	---------	----------	-------	------------	---------------	--------	---------	--------------	------------

~aiogram.types.input_file.InputFile | str, business_connection_id: str | None = None, *message_thread_id: int | None = None, duration: int* | *None* = *None*, *width: int* | *None* = *None*, *height: int | None = None, thumbnail:* ~aiogram.types.input_file.InputFile | None = *None*, *caption: str* | *None* = *None*, *parse mode:* str | ~aiogram.client.default.Default | None = <Default('parse mode')>, caption entities: ~typing.List[~aiogram.types.message_entity.MessageEntity] | None = None, has_spoiler: bool | None = None, *disable_notification: bool* | *None* = *None*, protect_content: bool | ~aiogram.client.default.Default | None = <*Default('protect_content')*>, *reply_parameters:* ~aiogram.types.reply_parameters.ReplyParameters | *None* = *None*, *reply_markup*: ~aiogram.types.inline_keyboard_markup.InlineKeyboardMarkup ~aiogram.types.reply_keyboard_markup.ReplyKeyboardMarkup ~aiogram.types.reply keyboard remove.ReplyKeyboardRemove | ~aiogram.types.force_reply.ForceReply | None = None, allow sending without reply: bool | None = None, reply_to_message_id: int | None = None, **extra data: ~typing.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* »

business_connection_id: str | None

Unique identifier of the business connection on behalf of which the message will be sent

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 | 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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

parse_mode: str | Default | 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 | Default | None

Protects the contents of the sent message from forwarding and saving

reply_parameters: ReplyParameters | None

Description of the message to reply to

reply_markup: InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply | None

Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account

allow_sending_without_reply: bool | None

Pass True if the message should be sent even if the specified replied-to message is not found

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

reply_to_message_id: int | None

If the message is a reply, ID of the original message

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

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_join_request.ChatJoinRequest.answer_animation()
- aiogram.types.chat_join_request.ChatJoinRequest.answer_animation_pm()
- aiogram.types.chat_member_updated.ChatMemberUpdated.answer_animation()

sendAudio

Returns: Message

class aiogram.methods.send_audio.**SendAudio**(*, *chat_id: int* | *str*, *audio:*

~aiogram.types.input_file.InputFile | str, business connection id: $str \mid None = None$, message_thread_id: int | None = None, caption: str | None = *None*, *parse mode: str* | ~*aiogram.client.default.Default* | *None* = <*Default('parse mode')*>, *caption entities:* ~typing.List[~aiogram.types.message_entity] | None = None, duration: int | None = None, performer: str | *None* = *None*, *title: str* | *None* = *None*, *thumbnail:* ~aiogram.types.input_file.InputFile | None = None, *disable_notification: bool | None = None, protect_content:* bool | ~aiogram.client.default.Default | None = <*Default('protect_content')*>, *reply_parameters:* ~aiogram.types.reply_parameters.ReplyParameters | None = *None*, *reply_markup*: ~aiogram.types.inline_keyboard_markup.InlineKeyboardMarkup ~aiogram.types.reply_keyboard_markup.ReplyKeyboardMarkup ~aiogram.types.reply keyboard remove.ReplyKeyboardRemove | ~aiogram.types.force_reply.ForceReply | None = None, allow_sending_without_reply: bool | None = None, reply_to_message_id: int | None = None, **extra_data: ~typing.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* »

business_connection_id: str | None

Unique identifier of the business connection on behalf of which the message will be sent

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 | Default | 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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass_context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

title: str | None

Track name

thumbnail: InputFile | 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 | Default | None

Protects the contents of the sent message from forwarding and saving

reply_parameters: ReplyParameters | None

Description of the message to reply to

reply_markup: InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply | None

Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account

allow_sending_without_reply: bool | None

Pass True if the message should be sent even if the specified replied-to message is not found

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

reply_to_message_id: int | None

If the message is a reply, ID of the original message

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

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_join_request.ChatJoinRequest.answer_audio()
- aiogram.types.chat_join_request.ChatJoinRequest.answer_audio_pm()
- aiogram.types.chat_member_updated.ChatMemberUpdated.answer_audio()

sendChatAction

Returns: bool

class aiogram.methods.send_chat_action.**SendChatAction**(*, *chat_id: int | str, action: str,*

business_connection_id: str | None = None, 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.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

```
model_post_init(_ModelMetaclass__context: Any) → None
```

We need to both initialize private attributes and call the user-defined model_post_init method.

business_connection_id: str | None

Unique identifier of the business connection on behalf of which the action will be sent

message_thread_id: int | None

Unique identifier for the target message thread; for 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, business_connection_id: str | None = None,
                                                         message thread id: int | None = None, last name: str |
                                                         None = None, vcard: str | None = None,
                                                         disable notification: bool | None = None,
                                                         protect content: bool | ~aiogram.client.default.Default
                                                         | None = <Default('protect_content')>,
                                                         reply_parameters:
                                                         ~aiogram.types.reply_parameters.ReplyParameters |
                                                         None = None, reply_markup:
                                                         ~aiogram.types.inline_keyboard_markup.InlineKeyboardMarkup
                                                         ~aiogram.types.reply_keyboard_markup.ReplyKeyboardMarkup
                                                         ~aiogram.types.reply_keyboard_remove.ReplyKeyboardRemove
                                                         | ~aiogram.types.force_reply.ForceReply | None =
                                                         None, allow_sending_without_reply: bool | None =
                                                         None, reply to message id: int | None = None,
                                                         **extra_data: ~typing.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

business_connection_id: str | None

Unique identifier of the business connection on behalf of which the message will be sent

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

disable_notification: bool | None

Sends the message silently. Users will receive a notification with no sound.

protect_content: bool | Default | None

Protects the contents of the sent message from forwarding and saving

reply_parameters: ReplyParameters | None

Description of the message to reply to

reply_markup: InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply | None

Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account

allow_sending_without_reply: bool | None

Pass True if the message should be sent even if the specified replied-to message is not found

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

reply_to_message_id: int | None

If the message is a reply, ID of the original message

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

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_join_request.ChatJoinRequest.answer_contact()
- aiogram.types.chat_join_request.ChatJoinRequest.answer_contact_pm()
- aiogram.types.chat_member_updated.ChatMemberUpdated.answer_contact()

sendDice

Returns: Message

= None, disable_notification: bool | None = None, protect_content: bool | ~aiogram.client.default.Default | None = <Default('protect_content')>, reply_parameters: ~aiogram.types.reply_parameters.ReplyParameters | None = None, reply_markup: ~aiogram.types.inline_keyboard_markup.InlineKeyboardMarkup | ~aiogram.types.reply_keyboard_markup.ReplyKeyboardMarkup | ~aiogram.types.reply_keyboard_remove.ReplyKeyboardRemove | ~aiogram.types.force_reply.ForceReply | None = None, allow_sending_without_reply: bool | None = None, reply_to_message_id: int | None = None, **extra_data: ~typing.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)

business_connection_id: str | None

Unique identifier of the business connection on behalf of which the message will be sent

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.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

protect_content: bool | Default | None

Protects the contents of the sent message from forwarding

reply_parameters: ReplyParameters | None

Description of the message to reply to

reply_markup: InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply | None

Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account

allow_sending_without_reply: bool | None

Pass True if the message should be sent even if the specified replied-to message is not found

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

reply_to_message_id: int | None

If the message is a reply, ID of the original message

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

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_join_request.ChatJoinRequest.answer_dice()
- aiogram.types.chat_join_request.ChatJoinRequest.answer_dice_pm()
- aiogram.types.chat_member_updated.ChatMemberUpdated.answer_dice()

sendDocument

Returns: Message

class aiogram.methods.send_document.SendDocument(*, chat_id: int | str, document:

~aiogram.types.input_file.InputFile | str, business connection id: $str \mid None = None$, *message_thread_id: int | None = None, thumbnail:* ~aiogram.types.input file.InputFile | None = None, *caption: str* | *None* = *None*, *parse_mode: str* | ~aiogram.client.default.Default | None = <Default('parse_mode')>, caption_entities: ~typing.List[~aiogram.types.message_entity.MessageEntity] | *None* = *None*, *disable_content_type_detection*: *bool* | *None* = *None*, *disable_notification: bool* | *None* = *None*, *protect_content: bool* | ~aiogram.client.default.Default | None = <*Default('protect_content')*>, reply_parameters: ~aiogram.types.reply_parameters.ReplyParameters | *None* = *None*, *reply_markup*: ~aiogram.types.inline_keyboard_markup.InlineKeyboardMarkup Т ~aiogram.types.reply_keyboard_markup.ReplyKeyboardMarkup ~aiogram.types.reply_keyboard_remove.ReplyKeyboardRemove ~aiogram.types.force reply.ForceReply | None = *None*, *allow_sending_without_reply: bool* | *None* = *None*, *reply_to_message_id: int* | *None* = *None*, **extra_data: ~typing.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* »

business_connection_id: str | None

Unique identifier of the business connection on behalf of which the message will be sent

message_thread_id: int | None

Unique identifier for the target message thread (topic) of the forum; for forum supergroups only

thumbnail: InputFile | 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 | Default | 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*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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 | Default | None

Protects the contents of the sent message from forwarding and saving

reply_parameters: ReplyParameters | None

Description of the message to reply to

reply_markup: InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply | None

Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account

allow_sending_without_reply: bool | None

Pass True if the message should be sent even if the specified replied-to message is not found

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

reply_to_message_id: int | None

If the message is a reply, ID of the original message

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

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_join_request.ChatJoinRequest.answer_document()
- aiogram.types.chat_join_request.ChatJoinRequest.answer_document_pm()
- aiogram.types.chat_member_updated.ChatMemberUpdated.answer_document()

sendLocation

Returns: Message

class aiogram.methods.send_location.**SendLocation**(*, *chat_id: int | str, latitude: float, longitude: float, husings_compaction_id_str_l_Nong__Nong__*

business_connection_id: str | None = None, 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 ~aiogram.client.default.Default | None = *<Default('protect content')>, reply parameters:* ~aiogram.types.reply_parameters.ReplyParameters | *None* = *None*, *reply_markup*: ~aiogram.types.inline_keyboard_markup.InlineKeyboardMarkup T ~aiogram.types.reply_keyboard_markup.ReplyKeyboardMarkup ~aiogram.types.reply_keyboard_remove.ReplyKeyboardRemove | ~aiogram.types.force_reply.ForceReply | None = *None*, *allow_sending_without_reply: bool* | *None* = *None*, *reply_to_message_id: int* | *None* = *None*, **extra_data: ~typing.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

business_connection_id: str | None

Unique identifier of the business connection on behalf of which the message will be sent

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.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(_*ModelMetaclass_context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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 | Default | None

Protects the contents of the sent message from forwarding and saving

reply_parameters: ReplyParameters | None

Description of the message to reply to

reply_markup: InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply | None

Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account

allow_sending_without_reply: bool | None

Pass True if the message should be sent even if the specified replied-to message is not found

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

reply_to_message_id: int | None

If the message is a reply, ID of the original message

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

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_join_request.ChatJoinRequest.answer_location()
- aiogram.types.chat_join_request.ChatJoinRequest.answer_location_pm()
- aiogram.types.chat_member_updated.ChatMemberUpdated.answer_location()

sendMediaGroup

Returns: List[Message]

class aiogram.methods.send_media_group.SendMediaGroup(*, chat_id: int | str, media: ~typ-

 ing.List[~aiogram.types.input_media_audio.InputMediaAudio
~aiogram.types.input_media_document.InputMediaDocument
~aiogram.types.input_media_photo.InputMediaPhoto
 ~aiogram.types.input_media_video.InputMediaVideo],
business_connection_id: str None = None,
message_thread_id: int None = None,
disable_notification: bool None = None,
protect_content: bool
~aiogram.client.default.Default None =
<default('protect_content')>,</default('protect_content')>
reply_parameters: ~aiogram.types.reply_parameters.ReplyParameters
None = None, allow_sending_without_reply:
bool None = None, reply_to_message_id: int
None = None, **extra_data: ~typing.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

business_connection_id: str | None

Unique identifier of the business connection on behalf of which the message will be sent

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.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

protect_content: bool | Default | None

Protects the contents of the sent messages from forwarding and saving

reply_parameters: ReplyParameters | None

Description of the message to reply to

allow_sending_without_reply: bool | None

Pass True if the message should be sent even if the specified replied-to message is not found

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

reply_to_message_id: int | None

If the messages are a reply, ID of the original message

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

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_join_request.ChatJoinRequest.answer_media_group()
- aiogram.types.chat_join_request.ChatJoinRequest.answer_media_group_pm()
- aiogram.types.chat_member_updated.ChatMemberUpdated.answer_media_group()

sendMessage

Returns: Message

class aiogram.methods.send_message.SendMessage(*, chat_id: int | str, text: str, business_connection_id:

```
str | None = None, message_thread_id: int | None =
None, parse mode: str |
~aiogram.client.default.Default | None =
<Default('parse mode')>, entities: ~typ-
ing.List[~aiogram.types.message_entity.MessageEntity]
| None = None, link preview options:
~aiogram.types.link_preview_options.LinkPreviewOptions
| ~aiogram.client.default.Default | None =
<Default('link_preview')>, disable_notification: bool |
None = None, protect_content: bool |
~aiogram.client.default.Default | None =
<Default('protect_content')>, reply_parameters:
~aiogram.types.reply_parameters.ReplyParameters
None = None, reply markup:
~aiogram.types.inline_keyboard_markup.InlineKeyboardMarkup
~aiogram.types.reply_keyboard_markup.ReplyKeyboardMarkup
Ι
~aiogram.types.reply_keyboard_remove.ReplyKeyboardRemove
| ~aiogram.types.force_reply.ForceReply | None =
None, allow sending without reply: bool | None =
None, disable_web_page_preview: bool |
~aiogram.client.default.Default | None =
<Default('link_preview_is_disabled')>,
reply_to_message_id: int | None = None,
**extra_data: ~typing.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

business_connection_id: str | None

Unique identifier of the business connection on behalf of which the message will be sent

message_thread_id: int | None

Unique identifier for the target message thread (topic) of the forum; for forum supergroups only

parse_mode: str | Default | 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*

link_preview_options: LinkPreviewOptions | Default | None

Link preview generation options for the message

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

disable_notification: bool | None

Sends the message silently. Users will receive a notification with no sound.

protect_content: bool | Default | None

Protects the contents of the sent message from forwarding and saving

reply_parameters: ReplyParameters | None

Description of the message to reply to

reply_markup: InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply | None

Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account

allow_sending_without_reply: bool | None

Pass True if the message should be sent even if the specified replied-to message is not found

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

disable_web_page_preview: bool | Default | None

Disables link previews for links in this message

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

reply_to_message_id: int | None

If the message is a reply, ID of the original message

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

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_join_request.ChatJoinRequest.answer()
- aiogram.types.chat_join_request.ChatJoinRequest.answer_pm()
- aiogram.types.chat_member_updated.ChatMemberUpdated.answer()

sendPhoto

Returns: Message

class aiogram.methods.send_photo.**SendPhoto**(*, *chat_id: int | str, photo:*

~aiogram.types.input_file.InputFile | str, business connection id: $str \mid None = None$, message_thread_id: int | None = None, caption: str | None = *None*, *parse mode: str* | ~*aiogram.client.default.Default* | *None* = *<Default('parse mode')>, caption entities:* ~typing.List[~aiogram.types.message_entity] | None = None, has spoiler: bool | None = None, *disable notification: bool* | *None* = *None, protect content:* bool | ~aiogram.client.default.Default | None = *<Default('protect_content')>, reply_parameters:* ~aiogram.types.reply_parameters.ReplyParameters | None = None, reply markup: ~aiogram.types.inline_keyboard_markup.InlineKeyboardMarkup ~aiogram.types.reply_keyboard_markup.ReplyKeyboardMarkup ~aiogram.types.reply keyboard remove.ReplyKeyboardRemove | ~aiogram.types.force_reply.ForceReply | None = None, allow sending without reply: bool | None = None, reply_to_message_id: int | None = None, **extra_data: ~typing.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* »

business_connection_id: str | None

Unique identifier of the business connection on behalf of which the message will be sent

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 | Default | 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*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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 | Default | None

Protects the contents of the sent message from forwarding and saving

reply_parameters: ReplyParameters | None

Description of the message to reply to

reply_markup: InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply | None

Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account

allow_sending_without_reply: bool | None

Pass True if the message should be sent even if the specified replied-to message is not found

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

reply_to_message_id: int | None

If the message is a reply, ID of the original message

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

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_join_request.ChatJoinRequest.answer_photo()
- aiogram.types.chat_join_request.ChatJoinRequest.answer_photo_pm()
- aiogram.types.chat_member_updated.ChatMemberUpdated.answer_photo()

sendPoll

Returns: Message

class aiogram.methods.send_poll.**SendPoll**(*, chat_id: int | str, question: str, options: ~typing.List[str], business connection id: $str \mid None = None$, 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 | ~aiogram.client.default.Default | None = <*Default('parse_mode')*>, *explanation_entities:* ~typing.List[~aiogram.types.message_entity.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* | ~aiogram.client.default.Default | None = <Default('protect_content')>, reply_parameters: ~aiogram.types.reply_parameters.ReplyParameters | None = *None, reply_markup:* ~aiogram.types.inline_keyboard_markup.InlineKeyboardMarkup ~aiogram.types.reply_keyboard_markup.ReplyKeyboardMarkup I ~aiogram.types.reply_keyboard_remove.ReplyKeyboardRemove | ~aiogram.types.force_reply.ForceReply | None = None, allow sending without reply: bool | None = None, *reply_to_message_id: int | None = None, **extra_data:* ~typing.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

business_connection_id: str | None

Unique identifier of the business connection on behalf of which the message will be sent

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 | Default | None

Mode for parsing entities in the explanation. See formatting options for more details.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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 | Default | None

Protects the contents of the sent message from forwarding and saving

reply_parameters: ReplyParameters | None

Description of the message to reply to

reply_markup: InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply | None

Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account

allow_sending_without_reply: bool | None

Pass True if the message should be sent even if the specified replied-to message is not found

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

reply_to_message_id: int | None

If the message is a reply, ID of the original message

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

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_join_request.ChatJoinRequest.answer_poll()
- aiogram.types.chat_join_request.ChatJoinRequest.answer_poll_pm()
- aiogram.types.chat_member_updated.ChatMemberUpdated.answer_poll()

sendVenue

Returns: Message

class aiogram.methods.send_venue.**SendVenue**(*, chat_id: int | str, latitude: float, longitude: float, title: str,

address: str, business_connection_id: str | None = None, 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 | ~aiogram.client.default.Default | *None* = *<Default('protect_content')>*, *reply_parameters:* ~aiogram.types.reply_parameters.ReplyParameters | None = None, reply_markup: ~aiogram.types.inline_keyboard_markup.InlineKeyboardMarkup ~aiogram.types.reply_keyboard_markup.ReplyKeyboardMarkup ~aiogram.types.reply_keyboard_remove.ReplyKeyboardRemove | ~aiogram.types.force_reply.ForceReply | None = None, allow_sending_without_reply: bool | None = None, reply to message id: int | None = None, **extra data: ~typing.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

business_connection_id: str | None

Unique identifier of the business connection on behalf of which the message will be sent

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'.)

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

We need to both initialize private attributes and call the user-defined model_post_init method.

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 | Default | None

Protects the contents of the sent message from forwarding and saving

reply_parameters: ReplyParameters | None

Description of the message to reply to

reply_markup: InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply | None

Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account

allow_sending_without_reply: bool | None

Pass True if the message should be sent even if the specified replied-to message is not found

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

reply_to_message_id: int | None

If the message is a reply, ID of the original message

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

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_join_request.ChatJoinRequest.answer_venue()
- aiogram.types.chat_join_request.ChatJoinRequest.answer_venue_pm()
- aiogram.types.chat_member_updated.ChatMemberUpdated.answer_venue()

sendVideo

Returns: Message

class aiogram.methods.send_video.**SendVideo**(*, *chat_id: int | str, video:*

~aiogram.types.input_file.InputFile | str, business_connection_id: str | None = None, *message_thread_id: int | None = None, duration: int | None* = None, width: int | None = None, height: int | None = *None, thumbnail: ~aiogram.types.input file.InputFile | None* = None, caption: str | None = None, parse_mode: str | ~aiogram.client.default.Default | None = <*Default('parse_mode')*>, *caption_entities:* ~typing.List[~aiogram.types.message_entity.MessageEntity] | None = None, has spoiler: bool | None = None, supports_streaming: bool | None = None, *disable_notification: bool | None = None, protect_content:* bool | ~aiogram.client.default.Default | None = <*Default('protect_content')*>, *reply_parameters:* ~aiogram.types.reply_parameters.ReplyParameters | None = *None*, *reply_markup*: ~aiogram.types.inline_keyboard_markup.InlineKeyboardMarkup ~aiogram.types.reply_keyboard_markup.ReplyKeyboardMarkup Ι ~aiogram.types.reply_keyboard_remove.ReplyKeyboardRemove |~aiogram.types.force reply.ForceReply | None = None, *allow_sending_without_reply: bool | None = None,* reply_to_message_id: int | None = None, **extra_data: ~typing.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* »

business_connection_id: str | None

Unique identifier of the business connection on behalf of which the message will be sent

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 | 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 | Default | None

Mode for parsing entities in the video caption. See formatting options for more details.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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 | Default | None

Protects the contents of the sent message from forwarding and saving

reply_parameters: ReplyParameters | None

Description of the message to reply to

reply_markup: InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply | None

Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account

allow_sending_without_reply: bool | None

Pass True if the message should be sent even if the specified replied-to message is not found

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

reply_to_message_id: int | None

If the message is a reply, ID of the original message

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

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_join_request.ChatJoinRequest.answer_video()
- aiogram.types.chat_join_request.ChatJoinRequest.answer_video_pm()
- aiogram.types.chat_member_updated.ChatMemberUpdated.answer_video()

sendVideoNote

Returns: Message

class aiogram.methods.send_video_note.**SendVideoNote**(*, *chat_id: int | str, video_note:*

~aiogram.types.input_file.InputFile | str, business connection id: $str \mid None = None$, *message_thread_id: int | None = None, duration: int* | *None* = *None*, *length: int* | *None* = *None*, thumbnail: ~aiogram.types.input file.InputFile | *None* = *None*, *disable notification: bool* | *None* = None, protect content: bool ~aiogram.client.default.Default | None = <Default('protect_content')>, reply_parameters: ~aiogram.types.reply_parameters.ReplyParameters | *None* = *None*, *reply_markup*: ~aiogram.types.inline_keyboard_markup.InlineKeyboardMarkup ~aiogram.types.reply_keyboard_markup.ReplyKeyboardMarkup ~aiogram.types.reply_keyboard_remove.ReplyKeyboardRemove |~aiogram.types.force reply.ForceReply | None = None, allow_sending_without_reply: bool | *None* = *None*, *reply to message id: int* | *None* = None, **extra_data: ~typing.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

business_connection_id: str | None

Unique identifier of the business connection on behalf of which the message will be sent

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 | 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* »

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

disable_notification: bool | None

Sends the message silently. Users will receive a notification with no sound.

protect_content: bool | Default | None

Protects the contents of the sent message from forwarding and saving

reply_parameters: ReplyParameters | None

Description of the message to reply to

reply_markup: InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply | None

Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account

allow_sending_without_reply: bool | None

Pass True if the message should be sent even if the specified replied-to message is not found

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

reply_to_message_id: int | None

If the message is a reply, ID of the original message

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

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_join_request.ChatJoinRequest.answer_video_note()
- aiogram.types.chat_join_request.ChatJoinRequest.answer_video_note_pm()
- aiogram.types.chat_member_updated.ChatMemberUpdated.answer_video_note()

sendVoice

Returns: Message

class aiogram.methods.send_voice.SendVoice(*, chat_id: int | str, voice:

~aiogram.types.input_file.InputFile | str, business_connection_id: str | None = None, message_thread_id: int | None = None, caption: str | None = *None*, *parse_mode: str* | ~*aiogram.client.default.Default* | *None* = *<Default('parse_mode')>, caption_entities:* ~typing.List[~aiogram.types.message_entity.MessageEntity] | None = None, duration: int | None = None, *disable_notification: bool | None = None, protect_content:* bool | ~aiogram.client.default.Default | None = <*Default('protect_content')*>, *reply_parameters:* ~aiogram.types.reply_parameters.ReplyParameters | None = None, reply_markup: ~aiogram.types.inline_keyboard_markup.InlineKeyboardMarkup ~aiogram.types.reply_keyboard_markup.ReplyKeyboardMarkup ~aiogram.types.reply_keyboard_remove.ReplyKeyboardRemove | ~aiogram.types.force_reply.ForceReply | None = None, *allow_sending_without_reply: bool | None = None, reply_to_message_id: int | None = None, **extra_data:* ~typing.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* »

business_connection_id: str | None

Unique identifier of the business connection on behalf of which the message will be sent

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 | Default | 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*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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 | Default | None

Protects the contents of the sent message from forwarding and saving

reply_parameters: ReplyParameters | None

Description of the message to reply to

reply_markup: InlineKeyboardMarkup | ReplyKeyboardMarkup | ReplyKeyboardRemove | ForceReply | None

Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove a reply keyboard or to force a reply from the user. Not supported for messages sent on behalf of a business account

allow_sending_without_reply: bool | None

Pass True if the message should be sent even if the specified replied-to message is not found

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

reply_to_message_id: int | None

If the message is a reply, ID of the original message

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

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_join_request.ChatJoinRequest.answer_voice()
- aiogram.types.chat_join_request.ChatJoinRequest.answer_voice_pm()
- aiogram.types.chat_member_updated.ChatMemberUpdated.answer_voice()

setChatAdministratorCustomTitle

Returns: bool

class aiogram.methods.set_chat_administrator_custom_title.SetChatAdministratorCustomTitle(*,

chat_id: int | str, user_id: int, custom_title: str, **extra_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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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

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)

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

```
model_post_init(\_ModelMetaclass\_context: Any) \rightarrow None
```

We need to both initialize private attributes and call the user-defined model_post_init method.

```
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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

 $model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_post_init(_*ModelMetaclass_context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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.

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)

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

photo: InputFile

New chat photo, uploaded using multipart/form-data

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

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)

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

```
model_post_init(ModelMetaclass_context: Any) \rightarrow None
```

We need to both initialize private attributes and call the user-defined model_post_init method.

sticker_set_name: str

Name of the sticker set to be set as the group sticker set

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)

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

title: str

New chat title, 1-128 characters

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()

setMessageReaction

Returns: bool

class aiogram.methods.set_message_reaction.SetMessageReaction(*, chat_id: int | str, message_id:

int, reaction: List[ReactionTypeEmoji | ReactionTypeCustomEmoji] | None = None, is_big: bool | None = None, **extra_data: Any)

Use this method to change the chosen reactions on a message. Service messages can't be reacted to. Automatically forwarded messages from a channel to its discussion group have the same available reactions as messages in the channel. Returns **True** on success.

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

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 target message. If the message belongs to a media group, the reaction is set to the first non-deleted message in the group instead.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

reaction: List[ReactionTypeEmoji | ReactionTypeCustomEmoji] | None

A JSON-serialized list of reaction types to set on the message. Currently, as non-premium users, bots can set up to one reaction per message. A custom emoji reaction can be used if it is either already present on the message or explicitly allowed by chat administrators.

is_big: bool | None

Pass True to set the reaction with a big animation

Usage

As bot method

result: bool = await bot.set_message_reaction(...)

Method as object

Imports:

- from aiogram.methods.set_message_reaction import SetMessageReaction
- alias: from aiogram.methods import SetMessageReaction

With specific bot

result: bool = await bot(SetMessageReaction(...))

As reply into Webhook in handler

return SetMessageReaction(...)

As shortcut from received object

aiogram.types.message.Message.react()

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.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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: ChatAdministrator-Rights L None _ None. for_channels: bool None = None, **extra_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.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass_context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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

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.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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

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.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

```
model_post_init(_ModelMetaclass_context: Any) → None
```

We need to both initialize private attributes and call the user-defined model_post_init method.

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

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.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init($ModelMetaclass_context: Any$) \rightarrow None

We need to both initialize private attributes and call the user-defined model_post_init method.

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

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

```
model_post_init(ModelMetaclass\_context: Any) \rightarrow None
```

We need to both initialize private attributes and call the user-defined model_post_init method.

```
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

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)

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

```
model_post_init(_ModelMetaclass__context: Any) → None
```

We need to both initialize private attributes and call the user-defined model_post_init method.

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

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)

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(_*ModelMetaclass_context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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

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)

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

We need to both initialize private attributes and call the user-defined model_post_init method.

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, message_thread_id: int, **extra_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)

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

message_thread_id: int

Unique identifier for the target message thread of the forum topic

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, **extra_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)

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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)

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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.chat.Chat.unpin_message()
- aiogram.types.message.Message.unpin()

Updating messages

deleteMessage

Returns: bool

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)

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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.chat.Chat.delete_message()
- aiogram.types.message.Message.delete()

deleteMessages

Returns: bool

Use this method to delete multiple messages simultaneously. If some of the specified messages can't be found, they are skipped. Returns True on success.

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

chat_id: int | str

Unique identifier for the target chat or username of the target channel (in the format @channelusername)

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

 $model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

message_ids: List[int]

A JSON-serialized list of 1-100 identifiers of messages to delete. See *aiogram.methods*. *delete_message.DeleteMessage* for limitations on which messages can be deleted

Usage

As bot method

result: bool = await bot.delete_messages(...)

Method as object

Imports:

- from aiogram.methods.delete_messages import DeleteMessages
- alias: from aiogram.methods import DeleteMessages

With specific bot

result: bool = await bot(DeleteMessages(...))

As reply into Webhook in handler

return DeleteMessages(...)

editMessageCaption

Returns: Union[Message, bool]

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

parse_mode: str | Default | 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: $str \mid None = None,$ horizontal accuracy: $float \mid None = None,$ heading: int | None = None, proximity_alert_radius: int | None = None,reply_markup: InlineKeyboardMarkup | 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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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]

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

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

```
model_post_init(_ModelMetaclass__context: Any) → None
```

We need to both initialize private attributes and call the user-defined model_post_init method.

```
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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

```
model_post_init(_ModelMetaclass__context: Any) → None
```

We need to both initialize private attributes and call the user-defined model_post_init method.

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 | ~aiogram.client.default.Default | None = <Default('parse_mode')>, entities: ~typing.List[~aiogram.types.message_entity.MessageEntity] | None = None, link_preview_options: ~aiogram.types.link_preview_options.LinkPreviewOptions | None = None, reply_markup: ~aiogram.types.inline_keyboard_markup.InlineKeyboardMark | None = None, disable_web_page_preview: bool | ~aiogram.client.default.Default | None = <Default('link_preview_is_disabled')>, **extra_data: ~typing.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 | Default | None

Mode for parsing entities in the message text. See formatting options for more details.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

entities: List[MessageEntity] | None

A JSON-serialized list of special entities that appear in message text, which can be specified instead of *parse_mode*

link_preview_options: LinkPreviewOptions | None

Link preview generation options for the message

reply_markup: InlineKeyboardMarkup | None

A JSON-serialized object for an inline keyboard.

disable_web_page_preview: bool | Default | None

Disables link previews for links in this message

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

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]

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 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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

 $model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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 | InlineOuervResultCachedSticker | InlineQueryResultCachedVideo InlineQueryResultCachedVoice InlineQueryResultArticle InlineQueryResultAudio InlineQueryResultContact InlineOueryResultGame InlineQueryResultDocument InlineQueryResultGif | InlineQueryResultLocation | InlineQueryResultMpeg4Gif InlineOueryResultPhoto | 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.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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 InlineOueryResultCachedVideo 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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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(...)

Games

getGameHighScores

Returns: List[GameHighScore]

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

 $model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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,

business_connection_id: str | None = None, message_thread_id: int | None = None, disable_notification: bool | None = None, protect_content: bool | ~aiogram.client.default.Default | None = <Default('protect_content')>, reply_parameters: ~aiogram.types.reply_parameters.ReplyParameters | None = None, reply_markup: ~aiogram.types.inline_keyboard_markup.InlineKeyboardMarkup | None = None, allow_sending_without_reply: bool | None = None, reply_to_message_id: int | None = None, **extra_data: ~typing.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.

business_connection_id: str | None

Unique identifier of the business connection on behalf of which the message will be sent

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.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

We need to both initialize private attributes and call the user-defined model_post_init method.

protect_content: bool | Default | None

Protects the contents of the sent message from forwarding and saving

reply_parameters: ReplyParameters | None

Description of the message to reply to

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. Not supported for messages sent on behalf of a business account.

allow_sending_without_reply: bool | None

Pass True if the message should be sent even if the specified replied-to message is not found

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

reply_to_message_id: int | None

If the message is a reply, ID of the original message

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

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_join_request.ChatJoinRequest.answer_game()
- aiogram.types.chat_join_request.ChatJoinRequest.answer_game_pm()
- aiogram.types.chat_member_updated.ChatMemberUpdated.answer_game()

setGameScore

Returns: Union[Message, bool]

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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(...)

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.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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

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)

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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: $int \mid None = None$, 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: $bool \mid None = None,$ 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.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

```
model_post_init(_ModelMetaclass__context: Any) → None
```

We need to both initialize private attributes and call the user-defined model_post_init method.

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

	<pre>str, provider_token: str, currency: str, prices: ~typ- ing.List[~aiogram.types.labeled_price.LabeledPrice], message_thread_id: int None = None, max_tip_amount: int None = None, suggested_tip_amounts: ~typing.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, is_flexible: bool None = None, disable_notification: bool None = None, protect_content: bool ~aiogram.client.default.Default None = <default('protect_content')>, reply_parameters: ~aiogram.types.reply_parameters.ReplyParameters None = None, reply_markup:</default('protect_content')></pre>
	· · · · · · · · · · · · · · · · · · ·
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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(_ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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 | Default | None

Protects the contents of the sent message from forwarding and saving

reply_parameters: ReplyParameters | None

Description of the message to reply to

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.

allow_sending_without_reply: bool | None

Pass True if the message should be sent even if the specified replied-to message is not found

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

reply_to_message_id: int | None

If the message is a reply, ID of the original message

Deprecated since version API:7.0: https://core.telegram.org/bots/api-changelog#december-29-2023

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_join_request.ChatJoinRequest.answer_invoice()
- aiogram.types.chat_join_request.ChatJoinRequest.answer_invoice_pm()
- aiogram.types.chat_member_updated.ChatMemberUpdated.answer_invoice()

Getting updates

deleteWebhook

Returns: bool

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(<u>ModelMetaclass_context</u>: Any) \rightarrow None We need to both initialize private attributes and call the user-defined model_post_init method.

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]

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.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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, message_reaction, and message_reaction_count (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

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init($ModelMetaclass_context: Any$) \rightarrow None

We need to both initialize private attributes and call the user-defined model_post_init method.

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.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

$model_post_init(ModelMetaclass_context: Any) \rightarrow None$

We need to both initialize private attributes and call the user-defined model_post_init method.

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*, *message_reaction*, and *message_reaction_count* (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
                                                                                  | PassportElementError-
                                                                                  FrontSide |
                                                                                  PassportElementErrorRe-
                                                                                  verseSide
                                                                                  PassportElementError-
                                                                                  Selfie
                                                                                  PassportElementErrorFile
                                                                                  | PassportElementError-
                                                                                  Files
                                                                                  PassportElementError-
                                                                                  TranslationFile
                                                                                  PassportElementError-
                                                                                  TranslationFiles |
                                                                                  PassportElementErrorUn-
                                                                                  specified], **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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

```
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(...)

2.3.5 Enums

Here is list of all available enums:

BotCommandScopeType

```
class aiogram.enums.bot_command_scope_type.BotCommandScopeType(value, names=None, *,
```

module=None, qualname=None, type=None, start=1, boundary=None)

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

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_VIDE0 = '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'
```

ChatBoostSourceType

```
class aiogram.enums.chat_boost_source_type.ChatBoostSourceType(value, names=None, *,
```

module=None, qualname=None, type=None, start=1, boundary=None)

This object represents a type of chat boost source. Source: https://core.telegram.org/bots/api#chatboostsource PREMIUM = 'premium'

GIFT_CODE = 'gift_code'

GIVEAWAY = 'giveaway'

ChatMemberStatus

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

ContentType

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'
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'
- USERS_SHARED = 'users_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'
- BOOST_ADDED = 'boost_added'
- 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'
- GIVEAWAY_CREATED = 'giveaway_created'
- **GIVEAWAY** = 'giveaway'
- GIVEAWAY_WINNERS = 'giveaway_winners'

GIVEAWAY_COMPLETED = 'giveaway_completed'
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'
USER_SHARED = 'user_shared'

Currency

```
class aiogram.enums.currency.Currency(value, names=None, *, module=None, qualname=None, type=None, start=1, boundary=None)
```

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

EncryptedPassportElement

class aiogram.enums.encrypted_passport_element.EncryptedPassportElement(value, names=None,

*, module=None, qualname=None, type=None, start=1, boundary=None)

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, names=None, *,

module=None, qualname=None, type=None, start=1, boundary=None)

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

ANIMATION = 'animation'

AUDIO = 'audio' DOCUMENT = 'document' PHOTO = 'photo' VIDEO = 'video'

KeyboardButtonPollTypeType

class aiogram.enums.keyboard_button_poll_type_type.KeyboardButtonPollTypeType(value,

names=None, *, module=None, qualname=None, type=None, start=1, boundary=None)

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

QUIZ = 'quiz'

REGULAR = 'regular'

MaskPositionPoint

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

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

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'
BLOCKQUOTE = 'blockquote'
CODE = 'code'
PRE = 'pre'
TEXT_LINK = 'text_link'
TEXT_MENTION = 'text_mention'
CUSTOM_EMOJI = 'custom_emoji'
```

MessageOriginType

This object represents origin of a message.

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

USER = 'user'

HIDDEN_USER = 'hidden_user'

CHAT = 'chat'

CHANNEL = 'channel'

ParseMode

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, names=None,

*, module=None, qualname=None, type=None, start=1, boundary=None)

```
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

This object represents poll type

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

REGULAR = 'regular'

QUIZ = 'quiz'

ReactionTypeType

This object represents reaction type.

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

EMOJI = 'emoji'

CUSTOM_EMOJI = 'custom_emoji'

StickerFormat

Format of the sticker

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

STATIC = 'static'

ANIMATED = 'animated'

VIDEO = 'video'

StickerType

REGULAR = 'regular'

MASK = 'mask' CUSTOM_EMOJI = 'custom_emoji'

TopicIconColor

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

BLUE = 7322096 YELLOW = 16766590 VIOLET = 13338331 GREEN = 9367192 ROSE = 16749490 RED = 16478047

UpdateType

```
class aiogram.enums.update_type.UpdateType(value, names=None, *, module=None, qualname=None,
                                             type=None, start=1, boundary=None)
     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'
     BUSINESS_CONNECTION = 'business_connection'
     BUSINESS_MESSAGE = 'business_message'
     EDITED_BUSINESS_MESSAGE = 'edited_business_message'
     DELETED_BUSINESS_MESSAGES = 'deleted_business_messages'
     MESSAGE_REACTION = 'message_reaction'
     MESSAGE_REACTION_COUNT = 'message_reaction_count'
     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' CHAT_BOOST = 'chat_boost' REMOVED_CHAT_BOOST = 'removed_chat_boost'

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.

async Bot.download_file(*file_path: str, destination: BinaryIO* | *Path* | *str* | *None* = *None, timeout: int* = 30, *chunk_size: int* = 65536, *seek: bool* = *True*) \rightarrow 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.

async Bot.download(file: str | Downloadable, destination: BinaryIO | Path | str | None = None, timeout: int = 30, chunk_size: int = 65536, seek: bool = True) \rightarrow 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"
)
```

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*) \rightarrow None

Parameters

name – Optional router name, can be useful for debugging

include_router(router: Router) \rightarrow Router

Attach another router.

Parameters router –

Returns

include_routers: Router) \rightarrow None

Attach multiple routers.

Parameters routers –

Returns

resolve_used_update_types(*skip_events: Set[str]* | *None* = *None*) \rightarrow 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

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

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
```

My chat member

```
@router.my_chat_member()
async def my_chat_member_handler(my_chat_member: types.ChatMemberUpdated) -> Any: pass
```

Chat member

```
@router.chat_member()
async def chat_member_handler(chat_member: types.ChatMemberUpdated) -> Any: pass
```

Chat join request

Message reaction

```
@router.message_reaction()
async def message_reaction_handler(message_reaction: types.MessageReactionUpdated) ->__
→Any: pass
```

Message reaction count

Chat boost

```
@router.chat_boost()
async def chat_boost_handler(chat_boost: types.ChatBoostUpdated) -> Any: pass
```

Remove chat boost

Errors

```
@router.errors()
async def error_handler(exception: types.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

name

module_1

router2 = Router()

@router2.message() ...

Listing 2: module_2.py

name

module_2

from module_2 import router2

router1 = Router() router1.include_router(router2)

Update

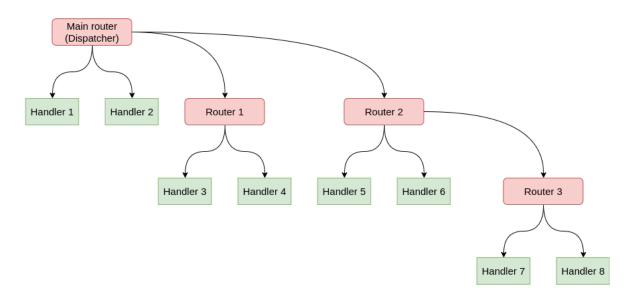
```
@dispatcher.update()
async def message_handler(update: types.Update) -> Any: pass
```

Warning: The only root Router (Dispatcher) can handle this type of event.

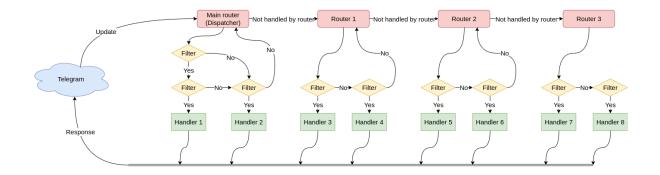
Note: Dispatcher already has default handler for this event type, so you can use it for handling all updates that are not handled by any other handlers.

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

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 -

 $\begin{aligned} \textbf{run_polling(*bots: Bot, polling_timeout: int = 10, handle_as_tasks: bool = True, backoff_config: \\ BackoffConfig = BackoffConfig(min_delay=1.0, max_delay=5.0, factor=1.3, jitter=0.1), \\ allowed_updates: List[str] | _SentinelObject | None = sentinel.UNSET, handle_signals: bool \\ = True, close_bot_session: bool = True, **kwargs: Any) \rightarrow None \end{aligned}$

Run many bots with polling

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
- 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() \rightarrow 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 = Dispathcher()
...
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
```

(continues on next page)

(continued from previous page)

) -> 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.aiohttp_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
```

(continues on next page)

(continued from previous page)

```
# 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. By default, all handlers has empty set of filters, so all updates will be passed to first handler that has empty set of filters.

aiogram has some builtin useful filters or you can write own 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")

Warning: 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

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

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
F.text != 'spam' # lambda message: message.text != 'spam'
```

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.startswith('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 # lambda message: not message.text
~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.regexp(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.regexp(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() \rightarrow str

Generate callback data string

Returns

valid callback data for Telegram Bot API

classmethod unpack(value: str) \rightarrow T

Parse callback data string

Parameters value – value from Telegram

Returns

instance of CallbackData

classmethod filter(*rule: MagicFilter* | *None* = *None*) \rightarrow CallbackQueryFilter

Generates a filter for callback query with rule

Parameters rule – magic rule

Returns

instance of filter

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

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
)
...
```

... 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,
        . . .
   )
```

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]) \rightarrow 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

Grouter.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.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, html
from aiogram.client.default import DefaultBotProperties
from aiogram.enums import ParseMode
from aiogram.filters import CommandStart
from aiogram.types import Message
# 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'
\hookrightarrow context
    # For example if you want to answer to incoming message you can use `message.answer(.
\leftrightarrow..)` alias
    # and the target chat will be passed to :ref:`aiogram.methods.send_message.
\leftrightarrow SendMessage`
    # method automatically or call API method directly via
    # Bot instance: `bot.send_message(chat_id=message.chat.id, ...)`
    await message.answer(f"Hello, {html.bold(message.from_user.full_name)}!")
@dp.message()
async def echo_handler(message: 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,
\rightarrow sticker etc.)
    ......
    trv:
        # 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)

```
async def main() -> None:
    # Initialize Bot instance with default bot properties which will be passed to all_
    API calls
    bot = Bot(token=TOKEN, default=DefaultBotProperties(parse_mode=ParseMode.HTML))
    # And the run events dispatching
    await dp.start_polling(bot)
if __name__ == "__main__":
    logging.basicConfig(level=logging.INF0, 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. By 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.aiohttp_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.aiohttp_server.BaseRequestHandler(dispatcher: Dispatcher,

handle_in_background: bool = False,
**data: Any)

__init__(*dispatcher:* Dispatcher, *handle_in_background:* bool = False, **data: Any) \rightarrow 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*) \rightarrow None

Register route and shutdown callback

Parameters

- **app** instance of aiohttp Application
- **path** route path
- kwargs -

abstract async resolve_bot(request: Request) \rightarrow 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.SimpleRequestHandler(dispatcher: Dispatcher, bot: Bot,

handle_in_background: bool = True, secret_token: str | None = None, **data: Any)

 $__init__(dispatcher: Dispatcher, bot: Bot, handle_in_background: bool = True, secret_token: str | None = None, **data: Any) <math>\rightarrow$ 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

 $\texttt{async close()} \rightarrow None$

Close bot session

register(*app: None*, /, *path: str*, ***kwargs: Any*) \rightarrow None

Register route and shutdown callback

Parameters

- **app** instance of aiohttp Application
- path route path
- kwargs -

async resolve_bot(request: Request) \rightarrow 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)

 $_$ ____(*dispatcher:* Dispatcher, handle_in_background: bool = True, bot_settings: Dict[str, Any] | None = None, **data: Any) \rightarrow 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*) \rightarrow None

Validate path, register route and shutdown callback

Parameters

- app instance of aiohttp Application
- **path** route path
- kwargs -

async resolve_bot(request: Request) \rightarrow 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) \rightarrow 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.

```
# 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'
\rightarrow 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.
\leftrightarrow 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.
\rightarrowetc.)
    ......
    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)
```

```
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_
\rightarrow 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.
\leftrightarrow 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;

```
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 without using 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)
```

```
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'
\rightarrow context
    # For example if you want to answer to incoming message you can use `message.answer(.
\leftrightarrow..)` 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.
\rightarrowetc.)
    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.
\hookrightarrow 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()
```

```
(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.
\hookrightarrow 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.INF0, 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.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.

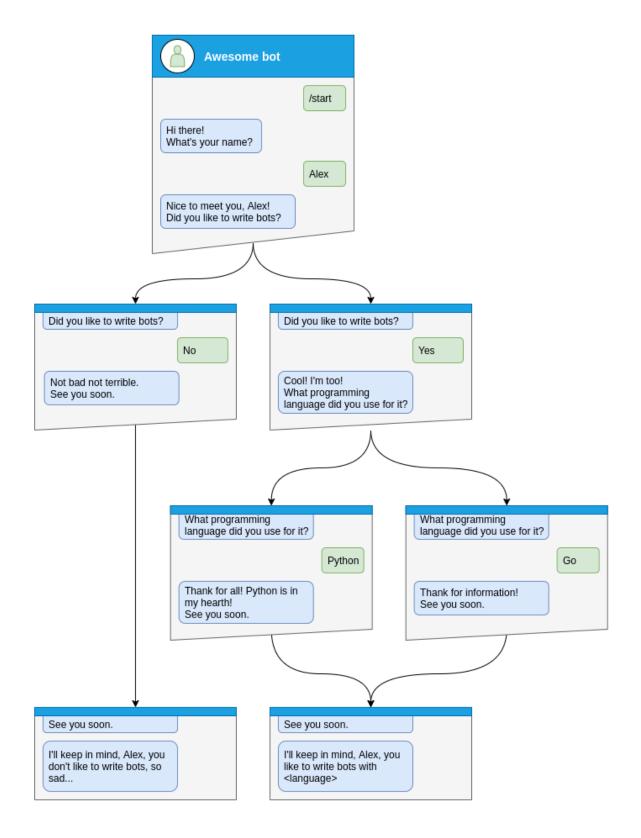
It is an abstract machine that can be in exactly one of a finite number of states at any given time. The FSM can change from one state to another in response to some inputs; the change from one state to another is called a transition.

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=[
                Ε
                    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:
```

```
return
logging.info("Cancelling state %r", current_state)
await state.clear()
await message.answer(
    "Cancelled.",
    reply_markup=ReplyKeyboardRemove(),
)
```

Complete example

```
import asyncio
1
   import logging
2
   import sys
3
   from os import getenv
4
   from typing import Any, Dict
5
6
   from aiogram import Bot, Dispatcher, F, Router, html
7
   from aiogram.enums import ParseMode
8
   from aiogram.filters import Command, CommandStart
9
   from aiogram.fsm.context import FSMContext
10
   from aiogram.fsm.state import State, StatesGroup
11
   from aiogram.types import (
12
       KeyboardButton,
13
       Message,
14
       ReplyKeyboardMarkup,
15
       ReplyKeyboardRemove,
16
   )
17
18
   TOKEN = getenv("BOT_TOKEN")
19
20
   form_router = Router()
21
22
23
   class Form(StatesGroup):
24
       name = State()
25
       like_bots = State()
26
       language = State()
27
28
29
   @form_router.message(CommandStart())
30
   async def command_start(message: Message, state: FSMContext) -> None:
31
       await state.set_state(Form.name)
32
       await message.answer(
33
            "Hi there! What's your name?",
34
            reply_markup=ReplyKeyboardRemove(),
35
       )
36
37
38
   @form_router.message(Command("cancel"))
39
```

```
(continued from previous page)
```

```
@form_router.message(F.text.casefold() == "cancel")
40
   async def cancel_handler(message: Message, state: FSMContext) -> None:
41
        .....
42
       Allow user to cancel any action
43
        .....
44
       current_state = await state.get_state()
45
       if current state is None:
46
            return
47
48
       logging.info("Cancelling state %r", current_state)
49
       await state.clear()
50
       await message.answer(
51
            "Cancelled.",
52
            reply_markup=ReplyKeyboardRemove(),
53
       )
54
55
56
   @form_router.message(Form.name)
57
   async def process_name(message: Message, state: FSMContext) -> None:
58
       await state update_data(name=message.text)
59
       await state.set_state(Form.like_bots)
60
       await message.answer(
61
            f"Nice to meet you, {html.quote(message.text)}!\nDid you like to write bots?",
62
            reply_markup=ReplyKeyboardMarkup(
63
                keyboard=[
64
                     Γ
65
                         KeyboardButton(text="Yes"),
66
                         KeyboardButton(text="No"),
67
                    1
68
                ],
69
                resize_keyboard=True,
70
            ),
71
       )
72
73
74
   @form_router.message(Form.like_bots, F.text.casefold() == "no")
75
   async def process_dont_like_write_bots(message: Message, state: FSMContext) -> None:
76
       data = await state.get_data()
77
       await state.clear()
78
       await message.answer(
79
            "Not bad not terrible.\nSee you soon.",
80
            reply_markup=ReplyKeyboardRemove(),
81
       )
82
       await show_summary(message=message, data=data, positive=False)
83
84
85
   @form_router.message(Form.like_bots, F.text.casefold() == "yes")
86
   async def process_like_write_bots(message: Message, state: FSMContext) -> None:
87
       await state set_state(Form language)
88
89
       await message.reply(
90
            "Cool! I'm too!\nWhat programming language did you use for it?",
91
```

```
reply_markup=ReplyKeyboardRemove(),
92
        )
93
94
95
    @form_router.message(Form.like_bots)
96
    async def process_unknown_write_bots(message: Message) -> None:
97
        await message.reply("I don't understand you :(")
98
99
100
    @form_router.message(Form.language)
101
    async def process_language(message: Message, state: FSMContext) -> None:
102
        data = await state.update_data(language=message.text)
103
        await state.clear()
104
105
        if message.text.casefold() == "python":
106
             await message.reply(
107
                 "Python, you say? That's the language that makes my circuits light up! "
108
             )
109
110
        await show_summary(message=message, data=data)
111
112
113
114
    async def show_summary(message: Message, data: Dict[str, Any], positive: bool = True) ->_
    \rightarrow None:
        name = data["name"]
115
        language = data.get("language", "<something unexpected>")
116
        text = f"I'll keep in mind that, {html.quote(name)}, "
117
        text += (
118
             f"you like to write bots with {html.quote(language)}."
119
             if positive
120
             else "you don't like to write bots, so sad..."
121
        )
122
        await message.answer(text=text, reply_markup=ReplyKeyboardRemove())
123
124
125
    async def main():
126
        bot = Bot(token=TOKEN, parse_mode=ParseMode.HTML)
127
        dp = Dispatcher()
128
        dp.include_router(form_router)
129
130
        await dp.start_polling(bot)
131
132
133
       <u>__name__</u> == "__main__":
    if
134
        logging basicConfig(level=logging INFO, stream=sys stdout)
135
        asyncio.run(main())
136
```

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

 $__init_() \rightarrow None$

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*) \rightarrow *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_business_connection_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, business_connection_id and destiny.

Format:

```
<prefix>:<bot_id?>:<business_connection_id?>:<chat_id>:<user_id>:<destiny?
>:<field>
```

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*) \rightarrow None

Set state for specified key

Parameters

- **key** storage key
- state new state

abstract async get_state(*key: StorageKey*) \rightarrow str | None

Get key state

Parameters key – storage key

Returns

current state

abstract async set_data(*key: StorageKey*, *data: Dict*[*str*, *Any*]) \rightarrow None

Write data (replace)

Parameters

- key storage key
- **data** new data

abstract async get_data(*key: StorageKey*) \rightarrow 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 date in the storage for key (like dict.update)

Parameters

- key storage key
- data partial data

Returns

new data

abstract async close() \rightarrow None

Close storage (database connection, file or etc.)

Scenes Wizard

New in version 3.2.

Warning: This feature is experimental and may be changed in future versions.

aiogram's basics API is easy to use and powerful, allowing the implementation of simple interactions such as triggering a command or message for a response. However, certain tasks require a dialogue between the user and the bot. This is where Scenes come into play.

Understanding Scenes

A Scene in **aiogram** is like an abstract, isolated namespace or room that a user can be ushered into via the code. When a user is within a Scene, most other global commands or message handlers are bypassed, unless they are specifically designed to function outside of the Scenes. This helps in creating an experience of focused interactions. Scenes provide a structure for more complex interactions, effectively isolating and managing contexts for different stages of the conversation. They allow you to control and manage the flow of the conversation in a more organized manner.

Scene Lifecycle

Each Scene can be "entered", "left" of "exited", allowing for clear transitions between different stages of the conversation. For instance, in a multi-step form filling interaction, each step could be a Scene - the bot guides the user from one Scene to the next as they provide the required information.

Scene Listeners

Scenes have their own hooks which are command or message listeners that only act while the user is within the Scene. These hooks react to user actions while the user is 'inside' the Scene, providing the responses or actions appropriate for that context. When the user is ushered from one Scene to another, the actions and responses change accordingly as the user is now interacting with the set of listeners inside the new Scene. These 'Scene-specific' hooks or listeners, detached from the global listening context, allow for more streamlined and organized bot-user interactions.

Scene Interactions

Each Scene is like a self-contained world, with interactions defined within the scope of that Scene. As such, only the handlers defined within the specific Scene will react to user's input during the lifecycle of that Scene.

Scene Benefits

Scenes can help manage more complex interaction workflows and enable more interactive and dynamic dialogs between the user and the bot. This offers great flexibility in handling multi-step interactions or conversations with the users.

How to use Scenes

For example we have a quiz bot, which asks the user a series of questions and then displays the results.

Lets start with the data models, in this example simple data models are used to represent the questions and answers, in real life you would probably use a database to store the data.

```
Listing 3: Questions list
```

```
@dataclass
class Answer:
    """
    Represents an answer to a question.
    """
    text: str
    """The answer text"""
    is_correct: bool = False
    """Indicates if the answer is correct"""

@dataclass
class Question:
    """
    Class representing a quiz with a question and a list of answers.
    """
```

```
text: str
    """The question text"""
    answers: list[Answer]
    """List of answers"""
    correct_answer: str = field(init=False)
    def __post_init__(self):
        self.correct_answer = next(answer.text for answer in self.answers if answer.is_
\rightarrowcorrect)
# Fake data, in real application you should use a database or something else
QUESTIONS = [
    Question(
        text="What is the capital of France?",
        answers=[
            Answer("Paris", is_correct=True),
            Answer("London"),
            Answer("Berlin"),
            Answer("Madrid"),
       ],
    ),
    Question(
        text="What is the capital of Spain?",
        answers=[
            Answer("Paris"),
            Answer("London"),
            Answer("Berlin"),
            Answer("Madrid", is_correct=True),
        ],
    ),
    Question(
        text="What is the capital of Germany?",
        answers=[
            Answer("Paris"),
            Answer("London"),
            Answer("Berlin", is_correct=True),
            Answer("Madrid"),
        ],
    ),
    Question(
        text="What is the capital of England?",
        answers=[
            Answer("Paris"),
            Answer("London", is_correct=True),
            Answer("Berlin"),
            Answer("Madrid"),
        ],
    ),
    Question(
```

```
text="What is the capital of Italy?",
answers=[
        Answer("Paris"),
        Answer("London"),
        Answer("Berlin"),
        Answer("Rome", is_correct=True),
    ],
    ],
]
```

Then, we need to create a Scene class that will represent the quiz game scene:

Note: Keyword argument passed into class definition describes the scene name - is the same as state of the scene.

Listing 4: Quiz Scene

```
class QuizScene(Scene, state="quiz"):
    """
    This class represents a scene for a quiz game.
    It inherits from Scene class and is associated with the state "quiz".
    It handles the logic and flow of the quiz game.
    """
```

Also we need to define a handler that helps to start the quiz game:

Listing 5: Start command handler

```
quiz_router = Router(name=__name__)
# Add handler that initializes the scene
quiz_router.message.register(QuizScene.as_handler(), Command("quiz"))
```

Once the scene is defined, we need to register it in the SceneRegistry:

Listing 6: Registering the scene

```
def create_dispatcher():
    # Event isolation is needed to correctly handle fast user responses
    dispatcher = Dispatcher(
        events_isolation=SimpleEventIsolation(),
    )
    dispatcher.include_router(quiz_router)

    # To use scenes, you should create a SceneRegistry and register your scenes there
    scene_registry = SceneRegistry(dispatcher)
    # ... and then register a scene in the registry
    # by default, Scene will be mounted to the router that passed to the SceneRegistry,
    # but you can specify the router explicitly using the `router` argument
    scene_registry.add(QuizScene)
    return dispatcher
```

So, now we can implement the quiz game logic, each question is sent to the user one by one, and the user's answer is checked at the end of all questions.

Now we need to write an entry point for the question handler:

```
Listing 7: Question handler entry point
```

```
@on.message.enter()
   async def on_enter(self, message: Message, state: FSMContext, step: int | None = 0) -
\rightarrow Any:
       Method triggered when the user enters the quiz scene.
       It displays the current question and answer options to the user.
        :param message:
        :param state:
        :param step: Scene argument, can be passed to the scene using the wizard
        :return:
        ......
       if not step:
            # This is the first step, so we should greet the user
            await message.answer("Welcome to the quiz!")
       trv:
            quiz = QUESTIONS[step]
       except IndexError:
            # This error means that the question's list is over
           return await self.wizard.exit()
       markup = ReplyKeyboardBuilder()
       markup.add(*[KeyboardButton(text=answer.text) for answer in quiz.answers])
       if step > 0:
            markup.button(text=" Back")
       markup.button(text=" Exit")
       await state.update_data(step=step)
       return await message answer(
            text=QUESTIONS[step].text,
            reply_markup=markup.adjust(2).as_markup(resize_keyboard=True),
       )
```

Once scene is entered, we should expect the user's answer, so we need to write a handler for it, this handler should expect the text message, save the answer and retake the question handler for the next question:

Listing 8:	Answer	handler
------------	--------	---------

```
@on.message(F.text)
async def answer(self, message: Message, state: FSMContext) -> None:
    """
    Method triggered when the user selects an answer.
    It stores the answer and proceeds to the next question.
```

```
:param message:
:param state:
:return:
"""
data = await state.get_data()
step = data["step"]
answers = data.get("answers", {})
answers[step] = message.text
await state.update_data(answers=answers)
await self.wizard.retake(step=step + 1)
```

When user answer with unknown message, we should expect the text message again:

Listing 9: Unknown message handler

```
@on.message()
async def unknown_message(self, message: Message) -> None:
    """
    Method triggered when the user sends a message that is not a command or an_
    answer.
    It asks the user to select an answer.
    :param message: The message received from the user.
    :return: None
    """
    await message.answer("Please select an answer.")
```

When all questions are answered, we should show the results to the user, as you can see in the code below, we use *await self.wizard.exit()* to exit from the scene when questions list is over in the *QuizScene.on_enter* handler.

Thats means that we need to write an exit handler to show the results to the user:

Listing 10: Show results handler

```
@on.message.exit()
async def on_exit(self, message: Message, state: FSMContext) -> None:
    """
    Method triggered when the user exits the quiz scene.
    It calculates the user's answers, displays the summary, and clears the stored_
    →answers.
    :param message:
    :param state:
    :return:
    """
    data = await state.get_data()
    answers = data.get("answers", {})
    correct = 0
```

```
incorrect = 0
user_answers = []
for step, quiz in enumerate(QUESTIONS):
    answer = answers.get(step)
    is_correct = answer == guiz.correct_answer
    if is_correct:
        correct += 1
        icon = ""
    else:
        incorrect += 1
        icon = ""
    if answer is None:
        answer = "no answer"
    user_answers.append(f"{quiz.text} ({icon} {html.quote(answer)})")
content = as_list(
    as_section(
        Bold("Your answers:"),
        as_numbered_list(*user_answers),
    ),
    ....
    as_section(
        Bold("Summary:"),
        as_list(
            as_key_value("Correct", correct),
            as_key_value("Incorrect", incorrect),
        ),
    ),
)
await message.answer(**content.as_kwargs(), reply_markup=ReplyKeyboardRemove())
await state.set_data({})
```

Also we can implement a actions to exit from the quiz game or go back to the previous question:

Listing 11: Exit handler

```
@on.message(F.text == " Exit")
async def exit(self, message: Message) -> None:
    """
    Method triggered when the user selects the "Exit" button.
    It exits the quiz.
    :param message:
    :return:
    """
    await self.wizard.exit()
```

Listing 12: Back handler

```
@on.message(F.text == " Back")
```

```
(continued from previous page)
```

```
async def back(self, message: Message, state: FSMContext) -> None:
    """
    Method triggered when the user selects the "Back" button.
    It allows the user to go back to the previous question.
    :param message:
    :param state:
    :return:
    """
    data = await state.get_data()
    step = data["step"]
    previous_step = step - 1
    if previous_step < 0:
        # In case when the user tries to go back from the first question,
        # we just exit the quiz
        return await self.wizard.exit()
    return await self.wizard.back(step=previous_step)</pre>
```

Now we can run the bot and test the quiz game:

Listing 13: Run the bot

```
async def main():
    dispatcher = create_dispatcher()
    bot = Bot(TOKEN)
    await dispatcher.start_polling(bot)
if ___name__ == "__main__":
    logging.basicConfig(level=logging.INFO)
    asyncio.run(main())
    # Alternatively, you can use aiogram-cli:
    # `aiogram run polling quiz_scene:create_dispatcher --log-level info --token BOT_
    -_TOKEN`
```

Complete them all

```
Listing 14: Quiz Example
```

```
import asyncio
import logging
from dataclasses import dataclass, field
from os import getenv
from typing import Any
from aiogram import Bot, Dispatcher, F, Router, html
from aiogram.filters import Command
from aiogram.fsm.context import FSMContext
from aiogram.fsm.scene import Scene, SceneRegistry, ScenesManager, on
from aiogram.fsm.storage.memory import SimpleEventIsolation
from aiogram.types import KeyboardButton, Message, ReplyKeyboardRemove
```

```
from aiogram.utils.formatting import (
    Bold,
    as_key_value,
    as_list,
    as_numbered_list,
    as_section,
)
from aiogram.utils.keyboard import ReplyKeyboardBuilder
TOKEN = getenv("BOT_TOKEN")
@dataclass
class Answer:
    ......
   Represents an answer to a question.
    ......
   text: str
    """The answer text"""
    is_correct: bool = False
    """Indicates if the answer is correct"""
@dataclass
class Question:
    ......
    Class representing a quiz with a question and a list of answers.
    .....
   text: str
   """The question text"""
    answers: list[Answer]
    """List of answers"""
    correct_answer: str = field(init=False)
    def __post_init__(self):
        self.correct_answer = next(answer.text for answer in self.answers if answer.is_
\rightarrow correct)
# Fake data, in real application you should use a database or something else
QUESTIONS = [
    Question(
        text="What is the capital of France?",
        answers=[
            Answer("Paris", is_correct=True),
            Answer("London"),
            Answer("Berlin"),
            Answer("Madrid"),
        ],
```

),

(continued from previous page)

```
Question(
        text="What is the capital of Spain?",
        answers=[
            Answer("Paris"),
            Answer("London"),
            Answer("Berlin"),
            Answer("Madrid", is_correct=True),
        ],
    ),
    Question(
        text="What is the capital of Germany?",
        answers=[
            Answer("Paris"),
            Answer("London"),
            Answer("Berlin", is_correct=True),
            Answer("Madrid"),
        ],
    ),
    Question(
        text="What is the capital of England?",
        answers=[
            Answer("Paris"),
            Answer("London", is_correct=True),
            Answer("Berlin"),
            Answer("Madrid"),
        ],
    ),
    Question(
        text="What is the capital of Italy?",
        answers=[
            Answer("Paris"),
            Answer("London"),
            Answer("Berlin"),
            Answer("Rome", is_correct=True),
        ],
    ),
]
class QuizScene(Scene, state="quiz"):
    ......
    This class represents a scene for a quiz game.
    It inherits from Scene class and is associated with the state "quiz".
    It handles the logic and flow of the quiz game.
    .....
    @on.message.enter()
    async def on_enter(self, message: Message, state: FSMContext, step: int | None = 0) -
\rightarrow Any:
        ......
```

```
(continued from previous page)
       Method triggered when the user enters the quiz scene.
       It displays the current question and answer options to the user.
        :param message:
        :param state:
        :param step: Scene argument, can be passed to the scene using the wizard
        :return:
        .....
       if not step:
            # This is the first step, so we should greet the user
            await message.answer("Welcome to the quiz!")
       try:
            quiz = QUESTIONS[step]
       except IndexError:
            # This error means that the question's list is over
           return await self.wizard.exit()
       markup = ReplyKeyboardBuilder()
       markup.add(*[KeyboardButton(text=answer.text) for answer in quiz.answers])
       if step > 0:
           markup.button(text=" Back")
       markup.button(text=" Exit")
       await state.update_data(step=step)
       return await message.answer(
            text=QUESTIONS[step].text,
            reply_markup=markup.adjust(2).as_markup(resize_keyboard=True),
       )
   @on.message.exit()
   async def on_exit(self, message: Message, state: FSMContext) -> None:
       Method triggered when the user exits the quiz scene.
       It calculates the user's answers, displays the summary, and clears the stored.
\rightarrow answers.
       :param message:
       :param state:
       :return:
        ......
       data = await state.get_data()
       answers = data.get("answers", {})
       correct = 0
       incorrect = 0
       user_answers = []
       for step, quiz in enumerate(QUESTIONS):
            answer = answers.get(step)
```

```
(continued from previous page)
```

```
is_correct = answer == quiz.correct_answer
        if is_correct:
            correct += 1
            icon = ""
        else:
            incorrect += 1
            icon = ""
        if answer is None:
            answer = "no answer"
        user_answers.append(f"{quiz.text} ({icon} {html.quote(answer)})")
    content = as_list(
        as_section(
            Bold("Your answers:"),
            as_numbered_list(*user_answers),
        ),
        .....
        as_section(
            Bold("Summary:"),
            as_list(
                as_key_value("Correct", correct),
                as_key_value("Incorrect", incorrect),
            ),
        ),
    )
    await message.answer(**content.as_kwargs(), reply_markup=ReplyKeyboardRemove())
    await state.set_data({})
@on.message(F.text == " Back")
async def back(self, message: Message, state: FSMContext) -> None:
    ......
    Method triggered when the user selects the "Back" button.
    It allows the user to go back to the previous question.
    :param message:
    :param state:
    :return:
    ......
    data = await state.get_data()
    step = data["step"]
    previous_step = step - 1
    if previous_step < 0:
        # In case when the user tries to go back from the first question,
        # we just exit the quiz
        return await self.wizard.exit()
    return await self.wizard.back(step=previous_step)
@on.message(F.text == " Exit")
async def exit(self, message: Message) -> None:
```

```
Method triggered when the user selects the "Exit" button.
        It exits the quiz.
        :param message:
        :return:
        ......
        await self.wizard.exit()
    @on.message(F.text)
    async def answer(self, message: Message, state: FSMContext) -> None:
        ......
        Method triggered when the user selects an answer.
        It stores the answer and proceeds to the next question.
        :param message:
        :param state:
        :return:
        .....
        data = await state.get_data()
        step = data["step"]
        answers = data.get("answers", {})
        answers[step] = message.text
        await state.update_data(answers=answers)
        await self.wizard.retake(step=step + 1)
    @on.message()
    async def unknown_message(self, message: Message) -> None:
        Method triggered when the user sends a message that is not a command or an.
\rightarrow answer.
        It asks the user to select an answer.
        :param message: The message received from the user.
        :return: None
        ......
        await message.answer("Please select an answer.")
quiz_router = Router(name=__name__)
# Add handler that initializes the scene
quiz_router.message.register(QuizScene.as_handler(), Command("quiz"))
@quiz_router.message(Command("start"))
async def command_start(message: Message, scenes: ScenesManager):
    await scenes.close()
    await message.answer(
                                                                              (continues on next page)
```

.....

```
"Hi! This is a quiz bot. To start the quiz, use the /quiz command.",
        reply_markup=ReplyKeyboardRemove(),
   )
def create_dispatcher():
    # Event isolation is needed to correctly handle fast user responses
   dispatcher = Dispatcher(
        events_isolation=SimpleEventIsolation(),
   )
   dispatcher.include_router(quiz_router)
   # To use scenes, you should create a SceneRegistry and register your scenes there
    scene_registry = SceneRegistry(dispatcher)
    # ... and then register a scene in the registry
    # by default. Scene will be mounted to the router that passed to the SceneRegistry.
    # but you can specify the router explicitly using the `router` argument
   scene_registry.add(QuizScene)
   return dispatcher
async def main():
   dispatcher = create_dispatcher()
   bot = Bot(TOKEN)
   await dispatcher.start_polling(bot)
if __name__ == "__main__":
   logging.basicConfig(level=logging.INFO)
   asyncio.run(main())
    # Alternatively, you can use aiogram-cli:
    # `aiogram run polling guiz_scene:create_dispatcher --log-level info --token BOT_
\rightarrow TOKEN
```

Components

- aiogram.fsm.scene.Scene represents a scene, contains handlers
- *aiogram.fsm.scene.SceneRegistry* container for all scenes in the bot, used to register scenes and resolve them by name
- *aiogram.fsm.scene.ScenesManager* manages scenes for each user, used to enter, leave and resolve current scene for user
- aiogram.fsm.scene.SceneConfig scene configuration, used to configure scene
- *aiogram.fsm.scene.SceneWizard* scene wizard, used to interact with user in scene from active scene handler
- Markers marker for scene handlers, used to mark scene handlers

class aiogram.fsm.scene.Scene(wizard: SceneWizard)

Represents a scene in a conversation flow.

A scene is a specific state in a conversation where certain actions can take place.

Each scene has a set of filters that determine when it should be triggered, and a set of handlers that define the actions to be executed when the scene is active.

Note: This class is not meant to be used directly. Instead, it should be subclassed to define custom scenes.

classmethod add_to_router(router: Router) \rightarrow None

Adds the scene to the given router.

Parameters router –

Returns

classmethod as_handler(**kwargs: Any) \rightarrow Callable[[...], Any]

Create an entry point handler for the scene, can be used to simplify the handler that starts the scene.

>>> router.message.register(MyScene.as_handler(), Command("start"))

classmethod as_router(name: str | None = None) \rightarrow Router

Returns the scene as a router.

Returns

new router

class aiogram.fsm.scene.**SceneRegistry**(*router*: Router, *register_on_add: bool = True*)

A class that represents a registry for scenes in a Telegram bot.

add(*scenes: Type[Scene], router: Router | None = None) \rightarrow None

This method adds the specified scenes to the registry and optionally registers it to the router.

If a scene with the same state already exists in the registry, a SceneException is raised.

Warning: If the router is not specified, the scenes will not be registered to the router. You will need to include the scenes manually to the router or use the register method.

Parameters

- **scenes** A variable length parameter that accepts one or more types of scenes. These scenes are instances of the Scene class.
- **router** An optional parameter that specifies the router to which the scenes should be added.

Returns

None

get(scene: Type[Scene] | str | None) \rightarrow Type[Scene]

This method returns the registered Scene object for the specified scene. The scene parameter can be either a Scene object or a string representing the name of the scene. If a Scene object is provided, the state attribute of the SceneConfig object associated with the Scene object will be used as the scene name. If None or an invalid type is provided, a SceneException will be raised.

If the specified scene is not registered in the SceneRegistry object, a SceneException will be raised.

Parameters

scene – A Scene object or a string representing the name of the scene.

Returns

The registered Scene object corresponding to the given scene parameter.

register(**scenes: Type*[Scene]) \rightarrow None

Registers one or more scenes to the SceneRegistry.

Parameters

scenes - One or more scene classes to register.

Returns

None

The ScenesManager class is responsible for managing scenes in an application. It provides methods for entering and exiting scenes, as well as retrieving the active scene.

async close(***kwargs:* Any) \rightarrow None

Close method is used to exit the currently active scene in the ScenesManager.

Parameters

kwargs – Additional keyword arguments passed to the scene's exit method.

Returns

None

async enter(*scene_type: Type*[Scene] | *str* | *None*, _*check_active: bool* = *True*, ***kwargs: Any*) \rightarrow None Enters the specified scene.

Parameters

- scene_type Optional Type[Scene] or str representing the scene type to enter.
- **_check_active** Optional bool indicating whether to check if there is an active scene to exit before entering the new scene. Defaults to True.
- kwargs Additional keyword arguments to pass to the scene's wizard.enter() method.

Returns

None

actions: Dict[SceneAction, Dict[str, CallableObject]]

Scene actions

callback_query_without_state: bool | None = None

Allow callback query without state

handlers: List[HandlerContainer]

Scene handlers

reset_data_on_enter: bool | None = None

Reset scene data on enter

reset_history_on_enter: bool | None = None

Reset scene history on enter

state: str | None

Scene state

class aiogram.fsm.scene.SceneWizard(scene_config: SceneConfig, manager: ScenesManager, state:

FSMContext, update_type: str, event: TelegramObject, data: Dict[str, Anv])

A class that represents a wizard for managing scenes in a Telegram bot.

Instance of this class is passed to each scene as a parameter. So, you can use it to transition between scenes, get and set data, etc.

Note: This class is not meant to be used directly. Instead, it should be used as a parameter in the scene constructor.

async back(***kwargs:* Any) \rightarrow None

This method is used to go back to the previous scene.

Parameters

kwargs – Keyword arguments that can be passed to the method.

Returns None

async clear_data() \rightarrow None

Clears the data.

Returns

None

async enter(***kwargs:* Any) \rightarrow None

Enter method is used to transition into a scene in the SceneWizard class. It sets the state, clears data and history if specified, and triggers entering event of the scene.

Parameters

kwargs – Additional keyword arguments.

Returns

None

async exit(***kwargs:* Any) \rightarrow None

Exit the current scene and enter the default scene/state.

Parameters

kwargs – Additional keyword arguments.

Returns None

async get_data() \rightarrow Dict[str, Any]

This method returns the data stored in the current state.

Returns

A dictionary containing the data stored in the scene state.

async goto(*scene: Type*[Scene] | *str*, ***kwargs: Any*) \rightarrow None

The *goto* method transitions to a new scene. It first calls the *leave* method to perform any necessary cleanup in the current scene, then calls the *enter* event to enter the specified scene.

Parameters

- **scene** The scene to transition to. Can be either a *Scene* instance or a string representing the scene.
- kwargs Additional keyword arguments to pass to the *enter* method of the scene manager.

Returns

None

async leave(*_with_history: bool* = *True*, ***kwargs: Any*) \rightarrow None

Leaves the current scene. This method is used to exit a scene and transition to the next scene.

Parameters

- _with_history Whether to include history in the snapshot. Defaults to True.
- kwargs Additional keyword arguments.

Returns

None

async retake(***kwargs:* Any) \rightarrow None

This method allows to re-enter the current scene.

Parameters

kwargs – Additional keyword arguments to pass to the scene.

Returns

None

async set_data(*data: Dict[str, Any]*) \rightarrow None

Sets custom data in the current state.

Parameters

data – A dictionary containing the custom data to be set in the current state.

Returns

None

async update_data(data: Dict[str, Any] | None = None, **kwargs: Any) \rightarrow Dict[str, Any]

This method updates the data stored in the current state

Parameters

- data Optional dictionary of data to update.
- kwargs Additional key-value pairs of data to update.

Returns

Dictionary of updated data

Markers

Markers are similar to the Router event registering mechanism, but they are used to mark scene handlers in the Scene class.

It can be imported from from aiogram.fsm.scene import on and should be used as decorator.

Allowed event types:

- 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

Each event type can be filtered in the same way as in the Router.

Also each event type can be marked as scene entry point, exit point or leave point.

If you want to mark the scene can be entered from message or inline query, you should use on.message or on. inline_query marker:

```
class MyScene(Scene, name="my_scene"):
    @on.message.enter()
    async def on_enter(self, message: types.Message):
        pass
    @on.callback_query.enter()
    async def on_enter(self, callback_query: types.CallbackQuery):
        pass
```

Scene has only tree points for transitions:

- enter point when user enters to the scene
- leave point when user leaves the scene and the enter another scene
- · exit point when user exits from the scene

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:

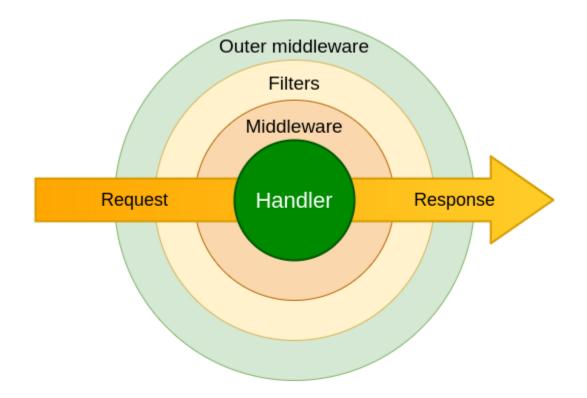
Middleware is reusable software that leverages patterns and frameworks to bridge the gap between the functional requirements of applications and the underlying operating systems, network protocol stacks, and databases.

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(...))



Attention: Middleware should be subclass of BaseMiddleware (from aiogram import BaseMiddleware) 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]) \rightarrow 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. If you want to stop processing event in middleware you should not call await handler(event, data).

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

```
model_post_init(ModelMetaclass\_context: Any) \rightarrow None
```

We need to both initialize private attributes and call the user-defined model_post_init method.

exception: Exception

Exception

Error types

exception aiogram.exceptions.AiogramError
Base exception for all aiogram errors.
<pre>exception aiogram.exceptions.DetailedAiogramError(message: str)</pre>
Base exception for all aiogram errors with detailed message.
exception aiogram.exceptions.CallbackAnswerException
Exception for callback answer.
exception aiogram.exceptions.SceneException
Exception for scenes.
<pre>exception aiogram.exceptions.UnsupportedKeywordArgument(message: str)</pre>
Exception raised when a keyword argument is passed as filter.
exception aiogram.exceptions. TelegramAPIError (<i>method: TelegramMethod</i> , <i>message: str</i>)
Base exception for all Telegram API errors.
exception aiogram.exceptions. TelegramNetworkError (<i>method: TelegramMethod</i> , <i>message: str</i>)
Base exception for all Telegram network errors.
exception aiogram.exceptions. TelegramRetryAfter (<i>method: TelegramMethod, message: str, retry_after: int</i>)
Exception raised when flood control exceeds.
<pre>exception aiogram.exceptions.TelegramMigrateToChat(method: TelegramMethod, message: str,</pre>
Exception raised when chat has been migrated to a supergroup.
exception aiogram.exceptions. TelegramBadRequest (<i>method: TelegramMethod, message: str</i>)
Exception raised when request is malformed.
exception aiogram.exceptions. TelegramNotFound (<i>method: TelegramMethod, message: str</i>)
Exception raised when chat, message, user, etc. not found.
exception aiogram.exceptions. TelegramConflictError (<i>method: TelegramMethod, message: str</i>)
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

```
@flags.chat_action
async def my_handler(message: Message)
```

Or just for rate-limit or something else

```
from aiogram import flags
```

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

Use in middlewares

aiogram.dispatcher.flags.check_flags(handler: HandlerObject | Dict[str, Any], magic: MagicFilter) \rightarrow Any

Check flags via magic filter

Parameters

- handler handler object or data
- magic instance of the magic

Returns

the result of magic filter check

 $\texttt{aiogram.dispatcher.flags.extract_flags}(\textit{handler: HandlerObject} \mid \textit{Dict[str, Any]}) \rightarrow \textit{Dict[str, Any]}$

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) \rightarrow 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.handlers import BaseHandler

By default you will need to override only method async def handle(self) -> Any: ...

This class also has a default initializer and you don't need to change it. The initializer accepts the incoming event and all contextual data, which can be accessed from the handler through attributes: event: TelegramEvent and data: Dict[Any, str]

If an 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: MaybeInaccessibleMessage | 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* 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() \rightarrow *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

 $copy() \rightarrow InlineKeyboardBuilder$

Make full copy of current builder with markup

Returns

```
export() \rightarrow 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) \rightarrow InlineKeyboardBuilder

Create builder from existing markup

Parameters markup –

Returns

row(**buttons: ButtonType, width: int* | *None* = *None*) \rightarrow 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

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*) \rightarrow *aiogram.utils.keyboard.ReplyKeyboardBuilder*

Add new button to markup

 $as_markup() \rightarrow aiogram.types.reply_keyboard_markup.ReplyKeyboardMarkup$

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*) \rightarrow 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() \rightarrow ReplyKeyboardBuilder
```

Make full copy of current builder with markup

Returns

export() \rightarrow 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) \rightarrow ReplyKeyboardBuilder

Create builder from existing markup

Parameters markup –

Returns

row(**buttons: ButtonType, width: int* | *None* = *None*) \rightarrow 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.)

Working with plural forms

The *gettext* from *aiogram.utils.i18n* is the one alias for two functions _gettext_ and _ngettext_ of GNU gettext Python module. Therefore, the wrapper for message strings is the same _(). You need to pass three parameters to the function: a singular string, a plural string, and a value.

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:

Simplel18nMiddleware

Simple I18n middleware.

Chooses language code from the User object received in event

__init__(*i18n: 118n, i18n_key: str* | *None* = '*i18n', middleware_key: str* = '*i18n_middleware'*) \rightarrow 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

Constl18nMiddleware

Const middleware chooses statically defined locale

 $_$ __init__(locale: str, i18n: I18n, i18n_key: str | None = 'i18n', middleware_key: str = 'i18n_middleware') \rightarrow 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

This middleware stores locale in the FSM storage

 $__iiit__(i18n: I18n, key: str = 'locale', i18n_key: str | None = 'i18n', middleware_key: str = 'i18n_middleware') <math>\rightarrow$ 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*) \rightarrow 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:

Abstract I18n middleware.

 $_$ _init__(*i18n: I18n, i18n_key: str* | *None* = '*i18n', middleware_key: str* = '*i18n_middleware'*) \rightarrow 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*]) \rightarrow 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.

Working with plural forms

Extracting with Pybabel all strings options:

- -k _:1,1t -k _:1,2 for both singular and plural
- -k ___ for lazy strings

pybabel extract -k _:1,1t -k _:1,2 -k __ --input-dirs=. -o locales/messages.pot

Note: Some useful options:

- Add comments for translators, you can use another tag if you want (TR) --add-comments=NOTE
- Contact email for bugreport --msgid-bugs-address=EMAIL
- Disable comments with string location in code --no-location
- Copyrights --copyright-holder=AUTHOR
- 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
- -1 en language. Can be changed to any other valid language code (For example -1 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

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) \rightarrow None

Parameters

- **bot** instance of the bot
- **chat_id** target chat id
- message_thread_id unique identifier for the target message thread; supergroups only
- action chat action type
- interval interval between iterations
- initial_sleep sleep before first sending of the action

classmethod choose_sticker(*chat_id: int* | *str*, *bot: Bot*, *message_thread_id: int* | *None* = *None*, *interval:* float = 5.0, *initial_sleep:* float = 0.0) \rightarrow *ChatActionSender*

Create instance of the sender with *choose_sticker* action

classmethod find_location(*chat_id: int* | *str, bot: Bot, message_thread_id: int* | *None* = *None, interval: float* = 5.0, *initial_sleep: float* = 0.0) \rightarrow *ChatActionSender*

Create instance of the sender with *find_location* action

classmethod record_video(*chat_id: int* | *str, bot: Bot, message_thread_id: int* | *None* = *None, interval:* $float = 5.0, initial_sleep: float = 0.0$) \rightarrow *ChatActionSender*

Create instance of the sender with record_video action

classmethod record_video_note(*chat_id: int* | *str, bot: Bot, message_thread_id: int* | *None* = *None, interval: float* = 5.0, *initial_sleep: float* = 0.0) \rightarrow *ChatActionSender*

Create instance of the sender with record_video_note action

classmethod record_voice(*chat_id: int* | *str, bot: Bot, message_thread_id: int* | *None* = *None, interval: float* = 5.0, *initial_sleep: float* = 0.0) \rightarrow *ChatActionSender*

Create instance of the sender with record_voice action

classmethod typing(*chat_id: int* | *str, bot: Bot, message_thread_id: int* | *None* = *None, interval: float* = 5.0, *initial_sleep: float* = 0.0) \rightarrow *ChatActionSender*

Create instance of the sender with typing action

classmethod upload_document (*chat_id: int* | *str, bot: Bot, message_thread_id: int* | *None* = *None, interval: float* = 5.0, *initial_sleep: float* = 0.0) \rightarrow *ChatActionSender*

Create instance of the sender with *upload_document* action

classmethod upload_photo(*chat_id: int* | *str, bot: Bot, message_thread_id: int* | *None* = *None, interval:* $float = 5.0, initial_sleep: float = 0.0$) \rightarrow *ChatActionSender*

Create instance of the sender with upload_photo action

classmethod upload_video(*chat_id: int* | *str, bot: Bot, message_thread_id: int* | *None* = *None, interval:* $float = 5.0, initial_sleep: float = 0.0$) \rightarrow *ChatActionSender*

Create instance of the sender with upload_video action

classmethod upload_video_note(*chat_id: int* | *str, bot: Bot, message_thread_id: int* | *None* = *None, interval: float* = 5.0, *initial_sleep: float* = 0.0) \rightarrow *ChatActionSender*

Create instance of the sender with upload_video_note action

classmethod upload_voice(*chat_id: int* | *str, bot: Bot, message_thread_id: int* | *None* = *None, interval: float* = 5.0, *initial_sleep: float* = 0.0) \rightarrow *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 usa 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:
```

(continues on next page)

(continued from previous page)

```
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) \rightarrow 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

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

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

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

```
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 [*Config-Dict*][pydantic.config.ConfigDict].

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'auth_date':
FieldInfo(annotation=datetime, required=True), 'can_send_after':
FieldInfo(annotation=Union[int, NoneType], required=False, default=None), 'chat':
FieldInfo(annotation=Union[WebAppChat, NoneType], required=False, default=None),
'chat_instance': FieldInfo(annotation=Union[str, NoneType], required=False,
default=None), 'chat_type': FieldInfo(annotation=Union[str, NoneType],
required=False, default=None), 'hash': FieldInfo(annotation=str, required=True),
'query_id': FieldInfo(annotation=Union[str, NoneType], required=False,
default=None), 'receiver': FieldInfo(annotation=Union[WebAppUser, NoneType],
required=False, default=None), 'start_param': FieldInfo(annotation=Union[str,
NoneType], required=False, default=None), 'user':
FieldInfo(annotation=Union[WebAppUser, NoneType], required=False, default=None), 'user':
```

Metadata about the fields defined on the model, mapping of field names to [*Field-Info*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

model_post_init($ModelMetaclass_context: Any$) \rightarrow None

We need to both initialize private attributes and call the user-defined model_post_init method.

query_id: str | None

A unique identifier for the Web App session, required for sending messages via the answerWebAppQuery method.

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.

chat: WebAppChat | None

An object containing data about the chat where the bot was launched via the attachment menu. Returned for supergroups, channels, and group chats – only for Web Apps launched via the attachment menu.

chat_type: str | None

Type of the chat from which the Web App was opened. Can be either "sender" for a private chat with the user opening the link, "private", "group", "supergroup", or "channel". Returned only for Web Apps launched from direct links.

chat_instance: str | None

Global identifier, uniquely corresponding to the chat from which the Web App was opened. Returned only for Web Apps launched from a direct link.

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.

can_send_after: int | None

Time in seconds, after which a message can be sent via the answerWebAppQuery method.

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.

is_premium: bool | None

True, if this user is a Telegram Premium user.

added_to_attachment_menu: bool | None

True, if this user added the bot to the attachment menu.

allows_write_to_pm: bool | None

True, if this user allowed the bot to message them.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

```
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 [*Config-Dict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'added_to_attachment_menu': FieldInfo(annotation=Union[bool, NoneType], required=False, default=None), 'allows_write_to_pm': FieldInfo(annotation=Union[bool, NoneType], required=False, default=None), 'first_name': FieldInfo(annotation=str, required=True), 'id': FieldInfo(annotation=int, required=True), 'is_bot': FieldInfo(annotation=Union[bool, NoneType], required=False, default=None), 'is_premium': FieldInfo(annotation=Union[bool, NoneType], required=False, default=None), 'language_code': FieldInfo(annotation=Union[str, NoneType], required=False, default=None), 'last_name': FieldInfo(annotation=Union[str, NoneType], required=False, default=None), 'photo_url': FieldInfo(annotation=Union[str, NoneType], required=False, default=None), 'username': FieldInfo(annotation=Union[str, NoneType], required=False, default=None),

default=None)}

Metadata about the fields defined on the model, mapping of field names to [*Field-Info*][pydantic.fields.FieldInfo].

This replaces *Model*.__*fields*__ from Pydantic V1.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

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.

class aiogram.utils.web_app.WebAppChat(**extra_data: Any)

This object represents a chat.

Source: https://core.telegram.org/bots/webapps#webappchat

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 "group", "supergroup" or "channel"

title: str

Title of the chat

username: str | None

Username of the chat

photo_url: str | None

URL of the chat's photo. The photo can be in .jpeg or .svg formats. Only returned for Web Apps launched from the attachment menu.

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

```
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 [*Config-Dict*][pydantic.config.ConfigDict].

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'id': FieldInfo(annotation=int,
required=True), 'photo_url': FieldInfo(annotation=Union[str, NoneType],
required=False, default=None), 'title': FieldInfo(annotation=str, required=True),
'type': FieldInfo(annotation=str, required=True), 'username':
FieldInfo(annotation=Union[str, NoneType], required=False, default=None)}
```

Metadata about the fields defined on the model, mapping of field names to [*Field-Info*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

model_post_init($ModelMetaclass_context: Any$) \rightarrow None

We need to both initialize private attributes and call the user-defined model_post_init method.

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

Bases: BaseMiddleware

 $__init__(pre: bool = False, text: str | None = None, show_alert: bool | None = None, url: str | None = None, cache_time: int | None = None) \rightarrow None$

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

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) \rightarrow 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

 $\texttt{disable()} \rightarrow \text{None}$

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', $sep: str = n' \rightarrow Text$

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') \rightarrow Text

Wrap each element to separated lines

Parameters

- items –
- sep –

Returns

aiogram.utils.formatting.as_marked_list(*items: Any, marker: str = '- ') \rightarrow 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 = '{}. ') \rightarrow 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) \rightarrow 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: $str = - \to Text$ 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 = '{}. ') \rightarrow

Text

Wrap elements as section with numbered list

Parameters

- title -
- body –
- start –
- fmt -

Returns

aiogram.utils.formatting.as_key_value(key: Any, value: Any) \rightarrow Text

Wrap elements pair as key-value line. ({key}: {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="\langle n \rangle 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
<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

 $\texttt{as_html()} \rightarrow str$

Render elements tree as HTML markup

as_markdown() \rightarrow 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

2.5.7 Media group builder

This module provides a builder for media groups, it can be used to build media groups for aiogram.types. input_media_photo.InputMediaPhoto, aiogram.types.input_media_video.InputMediaVideo, aiogram.types.input_media_document.InputMediaDocument and aiogram.types.input_media_audio. InputMediaAudio.

Warning: *aiogram.types.input_media_animation.InputMediaAnimation* is not supported yet in the Bot API to send as media group.

Usage

```
media_group = MediaGroupBuilder(caption="Media group caption")
# Add photo
media_group.add_photo(media="https://picsum.photos/200/300")
# Dynamically add photo with known type without using separate method
media_group.add(type="photo", media="https://picsum.photos/200/300")
# ... or video
media_group.add(type="video", media=FSInputFile("media/video.mp4"))
```

To send media group use *aiogram.methods.send_media_group.SendMediaGroup()* method, but when you use *aiogram.utils.media_group.MediaGroupBuilder* you should pass media argument as media_group.build().

If you specify caption in *aiogram.utils.media_group.MediaGroupBuilder* it will be used as caption for first media in group.

await bot.send_media_group(chat_id=chat_id, media=media_group.build())

References

- add(*, type: Literal[InputMediaType.AUDIO], media: str | InputFile, caption: str | None = None, parse_mode: str | None = UNSET_PARSE_MODE, caption_entities: List[MessageEntity] | None = None, duration: int | None = None, performer: str | None = None, title: str | None = None, **kwargs: Any) → None
- **add**(*, type: Literal[InputMediaType.PHOTO], media: str | InputFile, caption: str | None = None, parse_mode: str | None = UNSET_PARSE_MODE, caption_entities: List[MessageEntity] | None = None, has_spoiler: bool | None = None, **kwargs: Any) → None
- add(*, type: Literal[InputMediaType.VIDEO], media: str | InputFile, thumbnail: InputFile | str | None =
 None, caption: str | None = None, parse_mode: str | None = UNSET_PARSE_MODE, caption_entities:
 List[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, **kwargs:
 Any) → None
- add(*, type: Literal[InputMediaType.DOCUMENT], media: str | InputFile, thumbnail: InputFile | str | None = None, caption: str | None = None, parse_mode: str | None = UNSET_PARSE_MODE, caption_entities: List[MessageEntity] | None = None, disable_content_type_detection: bool | None = None, **kwargs: Any) → None

Add a media object to the media group.

Parameters

kwargs – Keyword arguments for the media object. The available keyword arguments depend on the media type.

Returns

None

add_audio(media: str | ~aiogram.types.input_file.InputFile, thumbnail: ~aiogram.types.input_file.InputFile |
 None = None, caption: str | None = None, parse_mode: str | None = <Default('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, **kwargs:
 ~typing.Any) → None

Add an audio file to the media group.

Parameters

• **media** – 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** *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.
- **caption** *Optional*. Caption of the audio to be sent, 0-1024 characters after entities parsing
- **parse_mode** *Optional*. Mode for parsing entities in the audio caption. See formatting options for more details.
- **caption_entities** *Optional*. List of special entities that appear in the caption, which can be specified instead of *parse_mode*
- duration Optional. Duration of the audio in seconds
- **performer** Optional. Performer of the audio
- **title** *Optional*. Title of the audio

Returns

None

```
add_document(media: str | ~aiogram.types.input_file.InputFile, thumbnail:
```

~aiogram.types.input_file.InputFile | None = None, caption: str | None = None, parse_mode: str | None = <Default('parse_mode')>, caption_entities:

~typing.List[~aiogram.types.message_entity.MessageEntity] | None = None,

```
disable_content_type_detection: bool | None = None, **kwargs: ~typing.Any) \rightarrow None
```

Add a document to the media group.

Parameters

- **media** 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** *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** *Optional*. Caption of the document to be sent, 0-1024 characters after entities parsing
- **parse_mode** *Optional*. Mode for parsing entities in the document caption. See formatting options for more details.
- **caption_entities** *Optional*. List of special entities that appear in the caption, which can be specified instead of *parse_mode*
- disable_content_type_detection 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.

Returns

None

- add_photo(media: str | ~aiogram.types.input_file.InputFile, caption: str | None = None, parse_mode: str | None = <Default('parse_mode')>, caption_entities:
 - ~typing.List[~aiogram.types.message_entity.MessageEntity] | None = None, has_spoiler: bool | None = None, **kwargs: ~typing.Any) \rightarrow None
 - Add a photo to the media group.

Parameters

• **media** – 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** *Optional*. Caption of the photo to be sent, 0-1024 characters after entities parsing
- **parse_mode** *Optional*. Mode for parsing entities in the photo caption. See formatting options for more details.
- **caption_entities** *Optional*. List of special entities that appear in the caption, which can be specified instead of *parse_mode*
- **has_spoiler** *Optional*. Pass True if the photo needs to be covered with a spoiler animation

Returns

None

- add_video(media: str | ~aiogram.types.input_file.InputFile, thumbnail: ~aiogram.types.input_file.InputFile | None = None, caption: str | None = None, parse_mode: str | None = <Default('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, **kwargs: ~typing.Any) → None
 - Add a video to the media group.

Parameters

• **media** – 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** *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 Optional. Caption of the video to be sent, 0-1024 characters after entities parsing
- **parse_mode** *Optional*. Mode for parsing entities in the video caption. See formatting options for more details.
- **caption_entities** *Optional*. List of special entities that appear in the caption, which can be specified instead of *parse_mode*
- width Optional. Video width
- height Optional. Video height
- duration Optional. Video duration in seconds
- **supports_streaming** *Optional*. Pass True if the uploaded video is suitable for streaming
- **has_spoiler** *Optional*. Pass True if the video needs to be covered with a spoiler animation

Returns

None

build() \rightarrow List[InputMediaAudio | InputMediaPhoto | InputMediaVideo | InputMediaDocument]

Builds a list of media objects for a media group.

Adds the caption to the first media object if it is present.

Returns

List of media objects.

2.5.8 Deep Linking

Telegram bots have a deep linking mechanism, that allows for passing additional parameters to the bot on startup. It could be a command that launches the bot — or an auth token to connect the user's Telegram account to their account on some external service.

You can read detailed description in the source: https://core.telegram.org/bots/features#deep-linking

We have added some utils to get deep links more handy.

Examples

Basic link example

```
from aiogram.utils.deep_linking import create_start_link
```

link = await create_start_link(bot, 'foo')

```
# result: 'https://t.me/MyBot?start=foo'
```

Encoded link

```
from aiogram.utils.deep_linking import create_start_link
```

```
link = await create_start_link(bot, 'foo', encode=True)
# result: 'https://t.me/MyBot?start=Zm9v'
```

Decode it back

```
from aiogram.utils.deep_linking import decode_payload
from aiogram.filters import CommandStart, CommandObject
from aiogram.types import Message
@router.message(CommandStart(deep_link=True))
async def handler(message: Message, command: CommandObject):
    args = command.args
    payload = decode_payload(args)
    await message.answer(f"Your payload: {payload}")
```

References

```
async aiogram.utils.deep_linking.create_start_link(bot: Bot, payload: str, encode: bool = False,
```

encoder: Callable[[bytes], bytes] | None = None) \rightarrow str

Create 'start' deep link with your payload.

If you need to encode payload or pass special characters set encode as True

Parameters

- **bot** bot instance
- payload args passed with /start
- encode encode payload with base64url or custom encoder
- encoder custom encoder callable

Returns

link

aiogram.utils.deep_linking.decode_payload(payload: str, decoder: Callable[[bytes], bytes] | None = None) \rightarrow str

Decode URL-safe base64url payload with decoder.

2.6 Changelog

2.6.1 3.5.0 (2024-04-23)

Features

- Added message_thread_id parameter to ChatActionSender class methods. #1437
- Added context manager interface to Bot instance, from now you can use:

```
async with Bot(...) as bot:
...
```

instead of

. . .

```
async with Bot(...).context() as bot:
```

#1468

Bugfixes

- WebAppUser Class Fields: Added missing *is_premium*, *added_to_attachment_menu*, and *allows_write_to_pm* fields to WebAppUser class to align with the Telegram API.
 - WebAppChat Class Implementation: Introduced the WebAppChat class with all its fields (*id*, *type*, *title*, *username*, and *photo_url*) as specified in the Telegram API, which was previously missing from the library.
 - WebAppInitData Class Fields: Included previously omitted fields in the WebAppInitData class: chat, chat_type, chat_instance, to match the official documentation for a complete Telegram Web Apps support.

#1424

- Fixed poll answer FSM context by handling voter_chat for poll_answer event #1436
- Added missing error handling to _background_feed_update (when in handle_in_background=True webhook mode) #1458

Improved Documentation

• Added WebAppChat class to WebApp docs, updated uk_UA localisation of WebApp docs. #1433

Misc

- Added full support of Bot API 7.2 #1444
- Loosened pydantic version upper restriction from <2.7 to <2.8 #1460

2.6.2 3.4.1 (2024-02-17)

Bugfixes

• Fixed JSON serialization of the LinkPreviewOptions class while it is passed as bot-wide default options. #1418

2.6.3 3.4.0 (2024-02-16)

Features

• Reworked bot-wide globals like parse_mode, disable_web_page_preview, and others to be more flexible.

Warning: Note that the old way of setting these global bot properties is now deprecated and will be removed in the next major release.

#1392

- A new enum KeyboardButtonPollTypeType for KeyboardButtonPollTypeType.type field has bed added. #1398
- Added full support of Bot API 7.1
 - Added support for the administrator rights can_post_stories, can_edit_stories, can_delete_stories in supergroups.
 - Added the class ChatBoostAdded and the field boost_added to the class Message for service messages about a user boosting a chat.
 - Added the field sender_boost_count to the class Message.
 - Added the field reply_to_story to the class Message.
 - Added the fields chat and id to the class Story.
 - Added the field unrestrict_boost_count to the class Chat.
 - Added the field custom_emoji_sticker_set_name to the class Chat.

#1417

Bugfixes

- Update KeyboardBuilder utility, fixed type-hints for button method, adjusted limits of the different markup types to real world values. #1399
- Added new reply_parameters param to message.send_copy because it hasn't been added there #1403

Improved Documentation

• Add notion "Working with plural forms" in documentation Utils -> Translation #1395

2.6.4 3.3.0 (2023-12-31)

Features

- Added full support of Bot API 7.0
 - Reactions
 - Replies 2.0
 - Link Preview Customization
 - Block Quotation
 - Multiple Message Actions
 - Requests for multiple users
 - Chat Boosts
 - Giveaway
 - Other changes

#1387

2.6.5 3.2.0 (2023-11-24)

Features

- Introduced Scenes feature that helps you to simplify user interactions using Finite State Machine. Read more about *Scenes*. #1280
- Added the new FSM strategy CHAT_TOPIC, which sets the state for the entire topic in the chat, also works in private messages and regular groups without topics. #1343

Bugfixes

- Fixed parse_mode argument in the in Message.send_copy shortcut. Disable by default. #1332
- Added ability to get handler flags from filters. #1360
- Fixed a situation where a CallbackData could not be parsed without a default value. #1368

Improved Documentation

- Corrected grammatical errors, improved sentence structures, translation for migration 2.x-3.x #1302
- Minor typo correction, specifically in module naming + some grammar. #1340
- Added *CITATION.cff* file for automatic academic citation generation. Now you can copy citation from the GitHub page and paste it into your paper. #1351
- Minor typo correction in middleware docs. #1353

Misc

- Fixed ResourceWarning in the tests, reworked RedisEventsIsolation fixture to use Redis connection from RedisStorage #1320
- Updated dependencies, bumped minimum required version:
 - magic-filter fixed .resolve operation
 - pydantic fixed compatibility (broken in 2.4)
 - aiodns added new dependency to the fast extras (pip install aiogram[fast])
 - others...

#1327

- Prevent update handling task pointers from being garbage collected, backport from 2.x #1331
- Updated typing-extensions package version range in dependencies to fix compatibility with FastAPI #1347
- Introduce Python 3.12 support #1354
- Speeded up CallableMixin processing by caching references to nested objects and simplifying kwargs assembly. #1357
- Added pydantic v2.5 support. #1361
- Updated thumbnail fields type to InputFile only #1372

2.6.6 3.1.1 (2023-09-25)

Bugfixes

• Fixed *pydantic* version <2.4, since 2.4 has breaking changes. #1322

2.6.7 3.1.0 (2023-09-22)

Features

- Added support for custom encoders/decoders for payload (and also for deep-linking). #1262
- Added aiogram.utils.input_media.MediaGroupBuilder for media group construction. #1293
- Added full support of Bot API 6.9 #1319

Bugfixes

- Added actual param hints for InlineKeyboardBuilder and ReplyKeyboardBuilder. #1303
- Fixed priority of events isolation, now user state will be loaded only after lock is acquired #1317

2.6.8 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

2.6.9 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 alogram 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

2.6.10 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

2.6.11 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

2.6.12 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 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
```

2.6.13 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. aiogram.enums.dice_emoji.DiceEmoji, aiogram.enums. content_type.ContentType, inline_query_result_type.InlineQueryResultType, aiogram.enums.input_media_type. aiogram.enums.mask_position_point.MaskPositionPoint, InputMediaType, 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.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

2.6.14 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 cacertificates the requests to Bot API fails with reason [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: self signed certificate in certificate chain #1066

2.6.15 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 describilized. 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

2.6.16 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

2.6.17 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

2.6.18 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

2.6.19 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

2.6.20 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

2.6.21 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

2.6.22 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

2.6.23 3.0.0a15 (2021-09-10)

Features

• Ability to iterate over all states in StatesGroup. Alogram 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

2.6.24 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

2.6.25 2.14.3 (2021-07-21)

• Fixed ChatMember type detection via adding customizable object serialization mechanism (#624, #623)

2.6.26 2.14.2 (2021-07-26)

- Fixed MemoryStorage cleaner (#619)
- Fixed unused default locale in I18nMiddleware (#562, #563)

2.6.27 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)

2.6.28 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)

2.6.29 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.__setitem__ when Telegram adds a new field (#532)
- Fixed examples/chat_type_filter.py (#533)
- Removed redundant definitions in framework code (#531)

2.6.30 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)

2.6.31 2.11.2 (2021-11-10)

- Fixed default parse mode
- Added missing "supports_streaming" argument to answer_video method #462

2.6.32 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)

2.6.33 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)

2.6.34 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'

2.6.35 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)

2.6.36 2.9.2 (2021-06-13)

- Fixed Message.get_full_command() #352
- Fixed markdown util #353

2.6.37 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)

2.6.38 2.8 (2021-04-26)

- Added full support of Bot API 4.8
- Added Message.answer_dice and Message.reply_dice methods (#306)

2.6.39 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

2.6.40 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)

2.6.41 2.6 (2021-01-23)

- Full support of Telegram Bot API v4.6 (Polls 2.0) #265
- Aded new filter IsContactSender (commit)
- Fixed proxy extra dependencies version #262

2.6.42 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

2.6.43 2.5.2 (2021-01-01)

• Get back quote_html and escape_md functions

2.6.44 2.5.1 (2021-01-01)

• Hot-fix of CommandStart filter

2.6.45 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...)

2.6.46 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

2.6.47 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

2.6.48 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)

2.6.49 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)
- Maked 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

2.6.50 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)

2.6.51 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 I18n Middleware; - Errors handlers now should accept only two arguments (current update and exception); - Used aiohttp_socks instead of aiosocksy for Socks4/5 proxy; - types.ContentType was divided to types.ContentType and types.ContentTypes; - Allowed to use rapidjson instead of ujson/json; - .current() method in bot and dispatcher objects was renamed to get_current();

Full changelog - You can read more details about this release in migration FAQ: https://aiogram.readthedocs.io/en/latest/migration_1_to_2.html

2.6.52 1.4 (2021-08-03)

• Bot API 4.0 (#57)

2.6.53 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.

2.6.54 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

2.6.55 1.3.1 (2018-05-27)

2.6.56 1.3 (2021-04-22)

- Allow to use Socks5 proxy (need manually install aiosocksy).
- Refactored aiogram.utils.executor module.
- [Warning] Updated requirements list.

2.6.57 1.2.3 (2018-04-14)

- Fixed API errors detection
- Fixed compability of setup.py with pip 10.0.0

2.6.58 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.

2.6.59 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.

2.6.60 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.

2.6.61 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)

2.6.62 1.0.4 (2018-01-10)

2.6.63 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.

2.6.64 1.0.2 (2017-11-29)

2.6.65 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 polling. Check it in you'r code before updating!
- Fixed helper for chat actions (types.ChatActions)
- Added example for media group.

2.6.66 1.0 (2017-11-19)

- Remaked data types serialozation/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.

2.6.67 0.4.1 (2017-08-03)

2.6.68 0.4 (2017-08-05)

2.6.69 0.3.4 (2017-08-04)

2.6.70 0.3.3 (2017-07-05)

2.6.71 0.3.2 (2017-07-04)

2.6.72 0.3.1 (2017-07-04)

2.6.73 0.2b1 (2017-06-00)

2.6.74 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

а

aiogram.dispatcher.flags, 542 aiogram.enums.bot_command_scope_type, 457 aiogram.enums.chat_action,457 aiogram.enums.chat_boost_source_type, 458 aiogram.enums.chat_member_status,458 aiogram.enums.chat_type, 459 aiogram.enums.content_type, 459 aiogram.enums.currency, 461 aiogram.enums.dice_emoji,464 aiogram.enums.encrypted_passport_element, 464 aiogram.enums.inline_query_result_type, 465 aiogram.enums.input_media_type, 465 aiogram.enums.keyboard_button_poll_type_type, 466 aiogram.enums.mask_position_point, 466 aiogram.enums.menu_button_type, 467 aiogram.enums.message_entity_type, 467 aiogram.enums.message_origin_type, 468 aiogram.enums.parse_mode, 468 aiogram.enums.passport_element_error_type, 468 aiogram.enums.poll_type, 469 aiogram.enums.reaction_type_type, 469 aiogram.enums.sticker_format, 469 aiogram.enums.sticker_type,469 aiogram.enums.topic_icon_color,470 aiogram.enums.update_type,470 aiogram.exceptions, 540 aiogram.handlers.callback_query, 544 aiogram.methods.add_sticker_to_set, 288 aiogram.methods.answer_callback_query, 307 aiogram.methods.answer_inline_query, 433 aiogram.methods.answer_pre_checkout_query, 442 aiogram.methods.answer_shipping_query, 443 aiogram.methods.answer_web_app_query, 436 aiogram.methods.approve_chat_join_request, 308 aiogram.methods.ban_chat_member, 309 aiogram.methods.ban_chat_sender_chat, 311 aiogram.methods.close, 312

aiogram.methods.close_forum_topic, 313 aiogram.methods.close_general_forum_topic, 314 aiogram.methods.copy_message, 315 aiogram.methods.copy_messages, 317 aiogram.methods.create_chat_invite_link, 319 aiogram.methods.create_forum_topic, 320 aiogram.methods.create_invoice_link,444 aiogram.methods.create_new_sticker_set, 289 aiogram.methods.decline_chat_join_request, 321 aiogram.methods.delete_chat_photo, 322 aiogram.methods.delete_chat_sticker_set, 323 aiogram.methods.delete_forum_topic, 324 aiogram.methods.delete_message, 420 aiogram.methods.delete_messages, 422 aiogram.methods.delete_my_commands, 325 aiogram.methods.delete_sticker_from_set, 291 aiogram.methods.delete_sticker_set, 292 aiogram.methods.delete_webhook, 450 aiogram.methods.edit_chat_invite_link, 326 aiogram.methods.edit_forum_topic, 328 aiogram.methods.edit_general_forum_topic, 329 aiogram.methods.edit_message_caption, 423 aiogram.methods.edit_message_live_location, 424 aiogram.methods.edit_message_media, 426 aiogram.methods.edit_message_reply_markup, 428 aiogram.methods.edit_message_text, 429 aiogram.methods.export_chat_invite_link, 330 aiogram.methods.forward_message, 331 aiogram.methods.forward_messages, 333 aiogram.methods.get_business_connection, 334 aiogram.methods.get_chat, 335 aiogram.methods.get_chat_administrators,336 aiogram.methods.get_chat_member, 337 aiogram.methods.get_chat_member_count, 338 aiogram.methods.get_chat_menu_button, 339 aiogram.methods.get_custom_emoji_stickers, 293 aiogram.methods.get_file,340

```
aiogram.methods.get_forum_topic_icon_stickers,aiogram.methods.set_game_score,440
        341
                                               aiogram.methods.set_message_reaction, 405
                                               aiogram.methods.set_my_commands, 407
aiogram.methods.get_game_high_scores, 437
aiogram.methods.get_me, 342
                                               aiogram.methods.set_my_default_administrator_rights,
aiogram.methods.get_my_commands, 343
                                                       408
aiogram.methods.get_my_default_administrator_raiphqsam.methods.set_my_description, 409
        344
                                               aiogram.methods.set_my_name, 410
aiogram.methods.get_my_description, 345
                                               aiogram.methods.set_my_short_description, 411
aiogram.methods.get_my_name, 346
                                               aiogram.methods.set_passport_data_errors,455
aiogram.methods.get_my_short_description, 347
                                               aiogram.methods.set_sticker_emoji_list, 299
aiogram.methods.get_sticker_set, 294
                                               aiogram.methods.set_sticker_keywords, 300
aiogram.methods.get_updates, 451
                                               aiogram.methods.set_sticker_mask_position,
aiogram.methods.get_user_chat_boosts, 347
                                                       301
aiogram.methods.get_user_profile_photos, 348
                                               aiogram.methods.set_sticker_position_in_set,
aiogram.methods.get_webhook_info,453
                                                       302
aiogram.methods.hide_general_forum_topic, 349
                                               aiogram.methods.set_sticker_set_thumbnail,
aiogram.methods.leave_chat, 350
                                                       303
aiogram.methods.log_out, 351
                                               aiogram.methods.set_sticker_set_title,305
aiogram.methods.pin_chat_message, 352
                                               aiogram.methods.set_webhook,453
aiogram.methods.promote_chat_member, 353
                                               aiogram.methods.stop_message_live_location,
aiogram.methods.reopen_forum_topic, 356
                                                       431
aiogram.methods.reopen_general_forum_topic,
                                               aiogram.methods.stop_poll,432
        357
                                               aiogram.methods.unban_chat_member, 412
aiogram.methods.replace_sticker_in_set, 295
                                               aiogram.methods.unban_chat_sender_chat, 414
aiogram.methods.restrict_chat_member, 358
                                               aiogram.methods.unhide_general_forum_topic,
aiogram.methods.revoke_chat_invite_link, 359
                                                       415
aiogram.methods.send_animation, 361
                                               aiogram.methods.unpin_all_chat_messages, 416
aiogram.methods.send_audio, 363
                                               aiogram.methods.unpin_all_forum_topic_messages,
aiogram.methods.send_chat_action, 366
                                                       417
aiogram.methods.send_contact, 368
                                               aiogram.methods.unpin_all_general_forum_topic_messages,
aiogram.methods.send_dice, 370
                                                       418
aiogram.methods.send_document, 372
                                               aiogram.methods.unpin_chat_message, 419
aiogram.methods.send_game,438
                                               aiogram.methods.upload_sticker_file, 306
aiogram.methods.send_invoice,447
                                               aiogram.types.animation, 17
aiogram.methods.send_location, 375
                                               aiogram.types.audio, 18
aiogram.methods.send_media_group, 377
                                               aiogram.types.birthdate, 19
aiogram.methods.send_message, 379
                                               aiogram.types.bot_command, 19
aiogram.methods.send_photo, 381
                                               aiogram.types.bot_command_scope, 20
aiogram.methods.send_poll, 384
                                               aiogram.types.bot_command_scope_all_chat_administrators,
aiogram.methods.send_sticker,296
                                                       20
aiogram.methods.send_venue, 387
                                               aiogram.types.bot_command_scope_all_group_chats,
aiogram.methods.send_video,390
                                                       21
aiogram.methods.send_video_note, 392
                                               aiogram.types.bot_command_scope_all_private_chats,
aiogram.methods.send_voice, 395
                                                       21
aiogram.methods.set_chat_administrator_custom_atidgeam.types.bot_command_scope_chat, 22
        397
                                               aiogram.types.bot_command_scope_chat_administrators,
aiogram.methods.set_chat_description, 399
                                                       22
aiogram.methods.set_chat_menu_button, 400
                                               aiogram.types.bot_command_scope_chat_member,
aiogram.methods.set_chat_permissions,401
                                                       23
aiogram.methods.set_chat_photo,402
                                               aiogram.types.bot_command_scope_default, 23
aiogram.methods.set_chat_sticker_set, 403
                                               aiogram.types.bot_description, 24
aiogram.methods.set_chat_title,404
                                               aiogram.types.bot_name, 24
aiogram.methods.set_custom_emoji_sticker_set_thimmbraniltypes.bot_short_description, 24
        298
                                               aiogram.types.business_connection, 25
```

aiogram.types.business_intro, 25 aiogram.types.business_location, 26 aiogram.types.business_messages_deleted, 26 aiogram.types.business_opening_hours, 27 aiogram.types.business_opening_hours_interval,aiogram.types.inline_keyboard_markup, 127 27 aiogram.types.callback_game, 287 aiogram.types.callback_query, 28 aiogram.types.chat, 29 aiogram.types.chat_administrator_rights,43 aiogram.types.chat_boost,45 aiogram.types.chat_boost_added, 46 aiogram.types.chat_boost_removed, 46 aiogram.types.chat_boost_source, 47 aiogram.types.chat_boost_source_gift_code,47 aiogram.types.chat_boost_source_giveaway,47 aiogram.types.chat_boost_source_premium, 48 aiogram.types.chat_boost_updated,49 aiogram.types.chat_invite_link,49 aiogram.types.chat_join_request, 50 aiogram.types.chat_location, 87 aiogram.types.chat_member,87 aiogram.types.chat_member_administrator,88 aiogram.types.chat_member_banned,90 aiogram.types.chat_member_left,90 aiogram.types.chat_member_member,91 aiogram.types.chat_member_owner,91 aiogram.types.chat_member_restricted,92 aiogram.types.chat_member_updated,93 aiogram.types.chat_permissions, 112 aiogram.types.chat_photo,113 aiogram.types.chat_shared, 114 aiogram.types.chosen_inline_result, 215 aiogram.types.contact, 114 aiogram.types.dice, 115 aiogram.types.document, 116 aiogram.types.encrypted_credentials, 267 aiogram.types.encrypted_passport_element, 268 aiogram.types.error_event, 539 aiogram.types.external_reply_info, 117 aiogram.types.file, 119 aiogram.types.force_reply, 119 aiogram.types.forum_topic, 120 aiogram.types.forum_topic_closed, 120 aiogram.types.forum_topic_created, 121 aiogram.types.forum_topic_edited, 121 aiogram.types.forum_topic_reopened, 122 aiogram.types.game, 287 aiogram.types.game_high_score, 288 aiogram.types.general_forum_topic_hidden, 122 aiogram.types.general_forum_topic_unhidden, 122 aiogram.types.giveaway, 122 aiogram.types.giveaway_completed, 123

aiogram.types.giveaway_created, 124 aiogram.types.giveaway_winners, 124 aiogram.types.inaccessible_message, 125 aiogram.types.inline_keyboard_button, 125 aiogram.types.inline_query, 216 aiogram.types.inline_query_result, 218 aiogram.types.inline_query_result_article, 219 aiogram.types.inline_query_result_audio, 220 aiogram.types.inline_query_result_cached_audio, 221 aiogram.types.inline_query_result_cached_document, 223 aiogram.types.inline_query_result_cached_gif, 225 aiogram.types.inline_query_result_cached_mpeg4_gif, 227 aiogram.types.inline_query_result_cached_photo, 229 aiogram.types.inline_query_result_cached_sticker, 231 aiogram.types.inline_query_result_cached_video, 233 aiogram.types.inline_query_result_cached_voice, 236 aiogram.types.inline_query_result_contact, 238 aiogram.types.inline_query_result_document, 239 aiogram.types.inline_query_result_game, 241 aiogram.types.inline_query_result_gif, 242 aiogram.types.inline_query_result_location, 244 aiogram.types.inline_guery_result_mpeg4_gif, 246 aiogram.types.inline_query_result_photo, 249 aiogram.types.inline_query_result_venue, 251 aiogram.types.inline_query_result_video, 252 aiogram.types.inline_query_result_voice, 254 aiogram.types.inline_query_results_button, 256 aiogram.types.input_contact_message_content, 257 aiogram.types.input_file, 127 aiogram.types.input_invoice_message_content, 257 aiogram.types.input_location_message_content, 260 aiogram.types.input_media, 128 aiogram.types.input_media_animation, 128 aiogram.types.input_media_audio, 129 aiogram.types.input_media_document, 130 aiogram.types.input_media_photo, 132

aiogram.types.input_media_video, 132 aiogram.types.passport_element_error_translation_file, aiogram.types.input_message_content, 261 275 aiogram.types.input_sticker, 263 aiogram.types.passport_element_error_translation_files, aiogram.types.input_text_message_content, 261 276 aiogram.types.input_venue_message_content, aiogram.types.passport_element_error_unspecified, 262 277 aiogram.types.invoice, 278 aiogram.types.passport_file, 278 aiogram.types.keyboard_button, 134 aiogram.types.photo_size, 197 aiogram.types.keyboard_button_poll_type, 135 aiogram.types.poll, 198 aiogram.types.keyboard_button_request_chat, aiogram.types.poll_answer, 199 135 aiogram.types.poll_option, 199 aiogram.types.keyboard_button_request_user, aiogram.types.pre_checkout_query, 280 137 aiogram.types.proximity_alert_triggered, 199 aiogram.types.keyboard_button_request_users, aiogram.types.reaction_count, 200 138 aiogram.types.reaction_type, 200 aiogram.types.labeled_price, 279 aiogram.types.reaction_type_custom_emoji, 201 aiogram.types.link_preview_options, 139 aiogram.types.reaction_type_emoji, 201 aiogram.types.location, 140 aiogram.types.reply_keyboard_markup, 202 aiogram.types.login_url,140 aiogram.types.reply_keyboard_remove, 203 aiogram.types.mask_position, 264 aiogram.types.reply_parameters, 203 aiogram.types.maybe_inaccessible_message, 141 aiogram.types.response_parameters, 204 aiogram.types.menu_button, 141 aiogram.types.sent_web_app_message, 263 aiogram.types.menu_button_commands, 142 aiogram.types.shared_user, 205 aiogram.types.menu_button_default, 142 aiogram.types.shipping_address, 281 aiogram.types.menu_button_web_app, 143 aiogram.types.shipping_option, 281 aiogram.types.message, 143 aiogram.types.shipping_query, 282 aiogram.types.message_auto_delete_timer_changeadiogram.types.sticker, 265 aiogram.types.sticker_set,266 191 aiogram.types.message_entity, 192 aiogram.types.story, 205 aiogram.types.message_id, 193 aiogram.types.successful_payment, 283 aiogram.types.message_origin, 193 aiogram.types.switch_inline_query_chosen_chat, aiogram.types.message_origin_channel, 193 206 aiogram.types.text_quote, 207 aiogram.types.message_origin_chat, 194 aiogram.types.message_origin_hidden_user, 195 aiogram.types.update, 284 aiogram.types.message_origin_user, 195 aiogram.types.user, 207 aiogram.types.message_reaction_count_updated, aiogram.types.user_chat_boosts, 209 196 aiogram.types.user_profile_photos, 209 aiogram.types.message_reaction_updated, 196 aiogram.types.user_shared, 209 aiogram.types.order_info, 279 aiogram.types.users_shared,210 aiogram.types.passport_data, 269 aiogram.types.venue, 210 aiogram.types.passport_element_error, 269 aiogram.types.video, 211 aiogram.types.passport_element_error_data_fieladiogram.types.video_chat_ended, 212 aiogram.types.video_chat_participants_invited, 270 aiogram.types.passport_element_error_file, 212 aiogram.types.video_chat_scheduled, 212 271 aiogram.types.passport_element_error_files, aiogram.types.video_chat_started, 213 272 aiogram.types.video_note, 213 aiogram.types.passport_element_error_front_sidaejogram.types.voice, 214 273 aiogram.types.web_app_data,214 aiogram.types.passport_element_error_reverse_sailegram.types.web_app_info, 215 aiogram.types.webhook_info, 286 273 aiogram.types.passport_element_error_selfie, aiogram.types.write_access_allowed, 215 274

INDEX

Symb	ools		<i>method</i>), 501
call_	_() (aiogram.dispatcher.middlewares.base.BaseM	1iddiewar	e-() (aiogram.webhook.security.IPFilter
	<i>method</i>), 537		<i>method</i>), 502
	_() (aiogram.filters.base.Filter method), 496	А	
<pre>init_</pre>	_() (aiogram.dispatcher.dispatcher.Dispatcher		
	<i>method</i>), 481	accent_	_color_id (aiogram.types.chat.Chat attribute),
init_	_() (aiogram.dispatcher.router.Router method),		30
	475	action	(aiogram.methods.send_chat_action.SendChatAction
init_			attribute), 366
	<i>method</i>), 486		s (aiogram.fsm.scene.SceneConfig attribute), 532
init_	() (aiogram.fsm.storage.memory.MemoryStorag	eactive_	
	<i>method</i>), 515		30
init_		actor_c	<pre>chat (aiogram.types.message_reaction_updated.MessageReactionU</pre>
	<i>method</i>), 515		attribute), 197
init_	() (aiogram.types.input_file.BufferedInputFile		iiogram.fsm.scene.SceneRegistry method), 531
	<i>method</i>), 474	add()	(a i ogram. utils. keyboard. In line Keyboard Builder
init_			<i>method</i>), 549
	<i>method</i>), 473	add()	(a iogram.utils.keyboard.ReplyKeyboardBuilder)
init_	() (aiogram.utils.callback_answer.CallbackAnsw		<i>method</i>), 551
	<i>method</i>), 565		(aiogram.utils.media_group.MediaGroupBuilder
init_	() (aiogram.utils.callback_answer.CallbackAnsw	verMiddle	ewnethod), 573
	<i>method</i>), 565	add_aud	lio() (aiogram.utils.media_group.MediaGroupBuilder
init_	() (aiogram.utils.chat_action.ChatActionSender		<i>method</i>), 573
	<i>method</i>), 556	add_dat	e (aiogram.types.chat_boost.ChatBoost at-
init_	() (aiogram.utils.formatting.Text method), 569		tribute), 45
init_	() (aiogram.utils.i18n.middleware.ConstI18nMi	d alled vale	<pre>cument() (aiogram.utils.media_group.MediaGroupBuilder</pre>
	<i>method</i>), 553		<i>method</i>), 574
init_	() (aiogram.utils.i18n.middleware.FSMI18nMid	deddapho	oto() (aiogram.utils.media_group.MediaGroupBuilder
	<i>method</i>), 554		<i>method</i>), 575
init_	() (aiogram.utils.i18n.middleware.I18nMiddlew	aadd_to_	router() (aiogram.fsm.scene.Scene class
	<i>method</i>), 554		<i>method</i>), 531
init_	() (aiogram.utils.i18n.middleware.SimpleI18nM	iaddewrine	leo() (aiogram.utils.media_group.MediaGroupBuilder
	<i>method</i>), 553		<i>method</i>), 575
init_	() (aiogram.utils.keyboard.InlineKeyboardBuild	<i>e</i> added_t	o_attachment_menu (<i>aiogram.types.user.User</i>
	<i>method</i>), 549		attribute), 208
init_	() (aiogram.utils.keyboard.ReplyKeyboardBuild	eadded_t	o_attachment_menu
	<i>method</i>), 550		(aiogram.utils.web_app.WebAppUser at-
init_	() (aiogram.webhook.aiohttp_server.BaseReque	stHandler	tribute), 561
	<i>method</i>), 499	additio	onal_chat_count
init_	() (aiogram.webhook.aiohttp_server.SimpleRequ	uestHandl	ekaiogram.types.giveaway_winners.GiveawayWinners
	<i>method</i>), 500		attribute), 124
init_	() (aiogram.webhook.aiohttp_server.TokenBased	d RDDRESSE	Haidwam.enums.encrypted_passport_element.EncryptedPassport

attribute), 464 address (aiogram.methods.send_venue.SendVenue at- tribute), 387	aiogram.enums.message_origin_type module,468 aiogram.enums.parse_mode
address (aiogram.types.business_location.BusinessLocation	on module, 468
attribute), 26 address (aiogram.types.chat_location.ChatLocation at- tribute), 87	<pre>aiogram.enums.passport_element_error_type module, 468 aiogram.enums.poll_type</pre>
address (aiogram.types.inline_query_result_venue.InlineQ attribute), 251	
address (aiogram.types.input_venue_message_content.Inp	
attribute), 262	aiogram.enums.sticker_format
address (aiogram.types.venue.Venue attribute), 210	module, 469
	aiogram.enums.sticker_type
aiogram.methods.add_sticker_to_set), 288	module, 469
adjust() (aiogram.utils.keyboard.InlineKeyboardBuilder method), 549	module, 470
adjust() (aiogram.utils.keyboard.ReplyKeyboardBuilder	
method), 551	module, 470
ADMINISTRATOR (aiogram.enums.chat_member_status.Cha attribute), 458	module, 540
AED (aiogram.enums.currency.Currency attribute), 461	aiogram.handlers.callback_query
AFN (<i>aiogram.enums.currency.Currency attribute</i>), 461	module, 544
aiogram.dispatcher.flags	aiogram.methods.add_sticker_to_set
module, 542	module, 288
<pre>aiogram.enums.bot_command_scope_type</pre>	aiogram.methods.answer_callback_query
module, 457	module, 307
aiogram.enums.chat_action module,457	<pre>aiogram.methods.answer_inline_query module, 433</pre>
<pre>aiogram.enums.chat_boost_source_type module, 458</pre>	<pre>aiogram.methods.answer_pre_checkout_query module, 442</pre>
<pre>aiogram.enums.chat_member_status module, 458</pre>	<pre>aiogram.methods.answer_shipping_query module, 443</pre>
aiogram.enums.chat_type	aiogram.methods.answer_web_app_query
module, 459	module, 436
aiogram.enums.content_type module,459	<pre>aiogram.methods.approve_chat_join_request module, 308</pre>
aiogram.enums.currency module,461	<pre>aiogram.methods.ban_chat_member module, 309</pre>
aiogram.enums.dice_emoji module,464	<pre>aiogram.methods.ban_chat_sender_chat module, 311</pre>
aiogram.enums.encrypted_passport_element	aiogram.methods.close
module, 464	module, 312
aiogram.enums.inline_query_result_type	aiogram.methods.close_forum_topic
module, 465	module, 313
<pre>aiogram.enums.input_media_type module, 465</pre>	<pre>aiogram.methods.close_general_forum_topic module, 314</pre>
<pre>aiogram.enums.keyboard_button_poll_type_type module, 466</pre>	<pre>aiogram.methods.copy_message module, 315</pre>
<pre>aiogram.enums.mask_position_point module, 466</pre>	<pre>aiogram.methods.copy_messages module, 317</pre>
aiogram.enums.menu_button_type	<pre>aiogram.methods.create_chat_invite_link</pre>
module, 467	module, 319
<pre>aiogram.enums.message_entity_type module, 467</pre>	<pre>aiogram.methods.create_forum_topic module, 320</pre>

aiogram.methods.create_invoice_link module.444 aiogram.methods.create_new_sticker_set module, 289 aiogram.methods.decline_chat_join_request module, 321 aiogram.methods.delete_chat_photo module, 322 aiogram.methods.delete_chat_sticker_set module, 323 aiogram.methods.delete_forum_topic module, 324 aiogram.methods.delete_message module, 420 aiogram.methods.delete_messages module, 422 aiogram.methods.delete_my_commands module, 325 aiogram.methods.delete_sticker_from_set module, 291 aiogram.methods.delete_sticker_set module, 292 aiogram.methods.delete_webhook module. 450 aiogram.methods.edit_chat_invite_link module. 326 aiogram.methods.edit_forum_topic module, 328 aiogram.methods.edit_general_forum_topic module, 329 aiogram.methods.edit_message_caption module, 423 aiogram.methods.edit_message_live_location module, 424 aiogram.methods.edit_message_media module. 426 aiogram.methods.edit_message_reply_markup module, 428 aiogram.methods.edit_message_text module, 429 aiogram.methods.export_chat_invite_link module, 330 aiogram.methods.forward_message module, 331 aiogram.methods.forward_messages module, 333 aiogram.methods.get_business_connection module, 334 aiogram.methods.get_chat module, 335 aiogram.methods.get_chat_administrators module, 336 aiogram.methods.get_chat_member module, 337

aiogram.methods.get_chat_member_count module.338 aiogram.methods.get_chat_menu_button module, 339 aiogram.methods.get_custom_emoji_stickers module, 293 aiogram.methods.get_file module, 340 aiogram.methods.get_forum_topic_icon_stickers module, 341 aiogram.methods.get_game_high_scores module, 437 aiogram.methods.get_me module, 342 aiogram.methods.get_my_commands module, 343 aiogram.methods.get_my_default_administrator_rights module, 344 aiogram.methods.get_my_description module, 345 aiogram.methods.get_my_name module, 346 aiogram.methods.get_my_short_description module.347 aiogram.methods.get_sticker_set module.294 aiogram.methods.get_updates module, 451 aiogram.methods.get_user_chat_boosts module, 347 aiogram.methods.get_user_profile_photos module, 348 aiogram.methods.get_webhook_info module, 453 aiogram.methods.hide_general_forum_topic module. 349 aiogram.methods.leave_chat module, 350 aiogram.methods.log_out module, 351 aiogram.methods.pin_chat_message module, 352 aiogram.methods.promote_chat_member module, 353 aiogram.methods.reopen_forum_topic module, 356 aiogram.methods.reopen_general_forum_topic module, 357 aiogram.methods.replace_sticker_in_set module, 295 aiogram.methods.restrict_chat_member module, 358 aiogram.methods.revoke_chat_invite_link module, 359

aiogram.methods.send_animation module. 361 aiogram.methods.send_audio module, 363 aiogram.methods.send_chat_action module, 366 aiogram.methods.send_contact module, 368 aiogram.methods.send_dice module, 370 aiogram.methods.send_document module, 372 aiogram.methods.send_game module, 438 aiogram.methods.send_invoice module, 447 aiogram.methods.send_location module, 375 aiogram.methods.send_media_group module, 377 aiogram.methods.send_message module, 379 aiogram.methods.send_photo module.381 aiogram.methods.send_poll module. 384 aiogram.methods.send_sticker module, 296 aiogram.methods.send_venue module, 387 aiogram.methods.send_video module, 390 aiogram.methods.send_video_note module, 392 aiogram.methods.send_voice module.395 aiogram.methods.set_chat_administrator_custom_aiodream.methods.unhide_general_forum_topic module, 397 aiogram.methods.set_chat_description module, 399 aiogram.methods.set_chat_menu_button module, 400aiogram.methods.set_chat_permissions module, 401 aiogram.methods.set_chat_photo module, 402 aiogram.methods.set_chat_sticker_set module, 403 aiogram.methods.set_chat_title module, 404 aiogram.methods.set_custom_emoji_sticker_set_thiumburaniltypes.audio module, 298 aiogram.methods.set_game_score module, 440

aiogram.methods.set_message_reaction module, 405 aiogram.methods.set_my_commands module, 407 aiogram.methods.set_my_default_administrator_rights module, 408aiogram.methods.set_my_description module, 409 aiogram.methods.set_my_name module, 410 aiogram.methods.set_my_short_description module, 411 aiogram.methods.set_passport_data_errors module, 455 aiogram.methods.set_sticker_emoji_list module, 299 aiogram.methods.set_sticker_keywords module, 300 aiogram.methods.set_sticker_mask_position module, 301 aiogram.methods.set_sticker_position_in_set module, 302 aiogram.methods.set_sticker_set_thumbnail module. 303 aiogram.methods.set_sticker_set_title module.305 aiogram.methods.set_webhook module, 453 aiogram.methods.stop_message_live_location module, 431 aiogram.methods.stop_poll module, 432 aiogram.methods.unban_chat_member module, 412 aiogram.methods.unban_chat_sender_chat module. 414 module, 415 aiogram.methods.unpin_all_chat_messages module, 416 aiogram.methods.unpin_all_forum_topic_messages module, 417 aiogram.methods.unpin_all_general_forum_topic_messages module, 418 aiogram.methods.unpin_chat_message module, 419 aiogram.methods.upload_sticker_file module, 306 aiogram.types.animation module, 17 module, 18 aiogram.types.birthdate module, 19

aiogram.types.bot_command aiogram.types.chat_boost_source_giveaway module. 19 module, 47 aiogram.types.bot_command_scope aiogram.types.chat_boost_source_premium module, 20 module, 48 aiogram.types.bot_command_scope_all_chat_adminaistgramorsypes.chat_boost_updated module, 49 module, 20 aiogram.types.bot_command_scope_all_group_chatasiogram.types.chat_invite_link module, 21 module, 49 aiogram.types.bot_command_scope_all_private_chamicsgram.types.chat_join_request module, 21 module, 50 aiogram.types.chat_location aiogram.types.bot_command_scope_chat module, 87 module, 22 aiogram.types.bot_command_scope_chat_administratiognsam.types.chat_member module, 87 module, 22 aiogram.types.bot_command_scope_chat_member aiogram.types.chat_member_administrator module, 23 module, 88 aiogram.types.chat_member_banned aiogram.types.bot_command_scope_default module, 23 module, 90 aiogram.types.chat_member_left aiogram.types.bot_description module, 24 module, 90 aiogram.types.bot_name aiogram.types.chat_member_member module, 24 module, 91 aiogram.types.bot_short_description aiogram.types.chat_member_owner module.24 module.91 aiogram.types.business_connection aiogram.types.chat_member_restricted module.25 module.92 aiogram.types.business_intro aiogram.types.chat_member_updated module, 25 module, 93 aiogram.types.chat_permissions aiogram.types.business_location module, 26 module, 112 aiogram.types.business_messages_deleted aiogram.types.chat_photo module, 26 module, 113 aiogram.types.business_opening_hours aiogram.types.chat_shared module, 27 module, 114 aiogram.types.business_opening_hours_interval aiogram.types.chosen_inline_result module, 27 module. 215 aiogram.types.callback_game aiogram.types.contact module, 287 module, 114 aiogram.types.callback_query aiogram.types.dice module, 115 module, 28 aiogram.types.chat aiogram.types.document module, 29 module, 116 aiogram.types.chat_administrator_rights aiogram.types.encrypted_credentials module, 43 module, 267 aiogram.types.chat_boost aiogram.types.encrypted_passport_element module, 45 module, 268 aiogram.types.chat_boost_added aiogram.types.error_event module, 539 module, 46 aiogram.types.chat_boost_removed aiogram.types.external_reply_info module, 46 module, 117 aiogram.types.chat_boost_source aiogram.types.file module, 47 module, 119 aiogram.types.chat_boost_source_gift_code aiogram.types.force_reply module, 47 module, 119

aiogram.types.forum_topic module, 120 aiogram.types.forum_topic_closed module, 120 aiogram.types.forum_topic_created module, 121 aiogram.types.forum_topic_edited module, 121 module, 241 aiogram.types.forum_topic_reopened module, 122 aiogram.types.game module, 287 aiogram.types.game_high_score module, 288 aiogram.types.general_forum_topic_hidden module, 122 aiogram.types.general_forum_topic_unhidden module, 122 module, 251 aiogram.types.giveaway module, 122 aiogram.types.giveaway_completed module, 123 aiogram.types.giveaway_created module, 124 aiogram.types.giveaway_winners module. 124 aiogram.types.inaccessible_message module, 125 aiogram.types.inline_keyboard_button module, 125 module, 257 aiogram.types.inline_keyboard_markup module, 127 aiogram.types.inline_query module, 216 aiogram.types.inline_query_result module, 218 aiogram.types.inline_query_result_article module, 219 aiogram.types.inline_query_result_audio module, 220 aiogram.types.inline_query_result_cached_audicaiogram.types.input_media_photo module, 221 aiogram.types.inline_query_result_cached_docummentgram.types.input_media_video module, 223 aiogram.types.inline_query_result_cached_gif aiogram.types.input_message_content module, 225 module, 261 aiogram.types.inline_query_result_cached_mpeg4apigfram.types.input_sticker module, 227 aiogram.types.inline_query_result_cached_photoaiogram.types.input_text_message_content module, 229 module, 261 aiogram.types.inline_query_result_cached_stickariogram.types.input_venue_message_content module, 231 module, 262 aiogram.types.inline_query_result_cached_videcaiogram.types.invoice module, 233 module, 278

aiogram.types.inline_query_result_cached_voice module, 236 aiogram.types.inline_query_result_contact module, 238 aiogram.types.inline_query_result_document module, 239 aiogram.types.inline_query_result_game aiogram.types.inline_query_result_gif module, 242 aiogram.types.inline_query_result_location module, 244 aiogram.types.inline_query_result_mpeg4_gif module, 246 aiogram.types.inline_query_result_photo module, 249 aiogram.types.inline_query_result_venue aiogram.types.inline_query_result_video module, 252 aiogram.types.inline_query_result_voice module, 254 aiogram.types.inline_query_results_button module. 256 aiogram.types.input_contact_message_content module, 257 aiogram.types.input_file module, 127 aiogram.types.input_invoice_message_content aiogram.types.input_location_message_content module, 260 aiogram.types.input_media module, 128 aiogram.types.input_media_animation module, 128 aiogram.types.input_media_audio module, 129 aiogram.types.input_media_document module, 130 module, 132 module, 132 module, 263

aiogram.types.keyboard_button module, 134 aiogram.types.keyboard_button_poll_type module, 135 aiogram.types.keyboard_button_request_chat module, 135 aiogram.types.keyboard_button_request_user module, 137 aiogram.types.keyboard_button_request_users module, 138 aiogram.types.labeled_price module, 279 aiogram.types.link_preview_options module, 139 aiogram.types.location module, 140 aiogram.types.login_url module, 140 aiogram.types.mask_position module, 264 aiogram.types.maybe_inaccessible_message module, 141 aiogram.types.menu_button module. 141 aiogram.types.menu_button_commands module. 142 aiogram.types.menu_button_default module, 142 aiogram.types.menu_button_web_app module, 143 aiogram.types.message module, 143 aiogram.types.message_auto_delete_timer_changemaiogram.types.pre_checkout_query module, 191 aiogram.types.message_entity module, 192 aiogram.types.message_id module, 193 aiogram.types.message_origin module, 193 aiogram.types.message_origin_channel module, 193 aiogram.types.message_origin_chat module, 194 aiogram.types.message_origin_hidden_user module, 195 aiogram.types.message_origin_user module, 195 aiogram.types.message_reaction_count_updated module, 196 aiogram.types.message_reaction_updated module, 196 aiogram.types.order_info module, 279

aiogram.types.passport_data module, 269 aiogram.types.passport_element_error module, 269 aiogram.types.passport_element_error_data_field module, 270 aiogram.types.passport_element_error_file module, 271 aiogram.types.passport_element_error_files module, 272 aiogram.types.passport_element_error_front_side module, 273 aiogram.types.passport_element_error_reverse_side module, 273 aiogram.types.passport_element_error_selfie module, 274 aiogram.types.passport_element_error_translation_file module, 275 aiogram.types.passport_element_error_translation_files module, 276 aiogram.types.passport_element_error_unspecified module, 277 aiogram.types.passport_file module. 278 aiogram.types.photo_size module, 197 aiogram.types.poll module, 198 aiogram.types.poll_answer module, 199 aiogram.types.poll_option module, 199 module, 280 aiogram.types.proximity_alert_triggered module, 199 aiogram.types.reaction_count module, 200 aiogram.types.reaction_type module, 200 aiogram.types.reaction_type_custom_emoji module, 201 aiogram.types.reaction_type_emoji module, 201 aiogram.types.reply_keyboard_markup module, 202 aiogram.types.reply_keyboard_remove module, 203 aiogram.types.reply_parameters module, 203 aiogram.types.response_parameters module, 204 aiogram.types.sent_web_app_message module, 263

aiogram.types.shared_user module, 205 aiogram.types.shipping_address module, 281 aiogram.types.shipping_option module, 281 aiogram.types.shipping_query module, 282 aiogram.types.sticker module, 265 aiogram.types.sticker_set module, 266 aiogram.types.story module, 205 aiogram.types.successful_payment module, 283 aiogram.types.switch_inline_query_chosen_chat module, 206 aiogram.types.text_quote module, 207 aiogram.types.update module, 284 aiogram.types.user module, 207 aiogram.types.user_chat_boosts module, 209 aiogram.types.user_profile_photos module, 209 aiogram.types.user_shared module, 209 aiogram.types.users_shared module, 210 aiogram.types.venue module, 210 aiogram.types.video module, 211 aiogram.types.video_chat_ended module, 212 aiogram.types.video_chat_participants_invited allow_sending_without_reply module, 212 aiogram.types.video_chat_scheduled module, 212 aiogram.types.video_chat_started module, 213 aiogram.types.video_note module, 213 aiogram.types.voice module, 214 aiogram.types.web_app_data module, 214 aiogram.types.web_app_info module, 215 aiogram.types.webhook_info module, 286

aiogram.types.write_access_allowed module, 215 AiogramError, 540 AiohttpSession (class in aiogram.client.session.aiohttp), 14 ALL (aiogram.enums.currency.Currency attribute), 461 ALL_CHAT_ADMINISTRATORS (aiogram.enums.bot_command_scope_type.BotCommandScopeTy attribute), 457 ALL_GROUP_CHATS (aiogram.enums.bot_command_scope_type.BotComma attribute), 457 ALL_PRIVATE_CHATS (aiogram.enums.bot_command_scope_type.BotComm attribute), 457 allow_bot_chats (aiogram.types.switch_inline_query_chosen_chat.Switch_inline_query_chat.Swi attribute), 206 allow_channel_chats (aiogram.types.switch_inline_query_chosen_chat.SwitchInlineQu attribute), 206 allow_group_chats(aiogram.types.switch_inline_query_chosen_chat.Sw attribute), 206 allow_sending_without_reply (aiogram.methods.copy_message.CopyMessage attribute), 316 allow_sending_without_reply (aiogram.methods.send_animation.SendAnimation attribute), 362 allow_sending_without_reply (aiogram.methods.send_audio.SendAudio attribute), 365 allow_sending_without_reply (aiogram.methods.send_contact.SendContact attribute), 369 allow_sending_without_reply (aiogram.methods.send_dice.SendDice attribute), 371 allow_sending_without_reply (aiogram.methods.send_document.SendDocument attribute), 373 (aiogram.methods.send_game.SendGame attribute), 439 allow_sending_without_reply (aiogram.methods.send_invoice.SendInvoice attribute), 449 allow_sending_without_reply (aiogram.methods.send_location.SendLocation attribute), 376 allow_sending_without_reply (aiogram.methods.send_media_group.SendMediaGroup attribute), 378 allow_sending_without_reply (aiogram.methods.send_message.SendMessage attribute), 380 allow_sending_without_reply

(aiogram.methods.send_photo.SendPhoto	Animation (class in aiogram.types.animation), 17
attribute), 383	<pre>answer() (aiogram.types.callback_query.CallbackQuery</pre>
allow_sending_without_reply	method), 28
(aiogram.methods.send_poll.SendPoll at- tribute), 386	<pre>answer() (aiogram.types.chat_join_request.ChatJoinRequest method), 51</pre>
allow_sending_without_reply (aiogram.methods.send_sticker.SendSticker	answer() (aiogram.types.chat_member_updated.ChatMemberUpdated method), 94
attribute), 297	answer() (aiogram.types.inline_query.InlineQuery
allow_sending_without_reply	method), 217
(aiogram.methods.send_venue.SendVenue attribute), 388	answer() (aiogram.types.message.Message method), 168
allow_sending_without_reply	answer() (aiogram.types.pre_checkout_query.PreCheckoutQuery
(aiogram.methods.send_video.SendVideo	method), 280
attribute), 391	answer() (aiogram.types.shipping_query.ShippingQuery
allow_sending_without_reply	method), 282
(aiogram.methods.send_video_note.SendVideoN	otanswer_animation() (aiogram.types.chat_join_request.ChatJoinRequest
attribute), 394	<i>method</i>), 53
allow_sending_without_reply	answer_animation() (aiogram.types.chat_member_updated.ChatMember_
(aiogram.methods.send_voice.SendVoice	method), 95
attribute), 396	<pre>answer_animation() (aiogram.types.message.Message</pre>
allow_sending_without_reply	<i>method</i>), 151
(aiogram.types.reply_parameters.ReplyParamete	
attribute), 203	(aiogram.types.chat_join_request.ChatJoinRequest
allow_user_chats (<i>aiogram.types.switch_inline_query_</i>	
attribute), 206	answer_audio() (aiogram.types.chat_join_request.ChatJoinRequest
allowed_updates(aiogram.methods.get_updates.GetUpd attribute), 452	lates method), 55 answer_audio() (aiogram.types.chat_member_updated.ChatMemberUpda
allowed_updates (<i>aiogram.methods.set_webhook.SetWew attribute</i>), 454	bhook method), 96 answer_audio() (aiogram.types.message.Message
allowed_updates(aiogram.types.webhook_info.Webhook	
attribute), 287	answer_audio_pm() (aiogram.types.chat_join_request.ChatJoinRequest
allows_multiple_answers	method), 56
(aiogram.methods.send_poll.SendPoll at- tribute), 385	<pre>answer_contact() (aiogram.types.chat_join_request.ChatJoinRequest</pre>
allows_multiple_answers (aiogram.types.poll.Poll attribute), 198	<pre>answer_contact() (aiogram.types.chat_member_updated.ChatMemberUj method), 97</pre>
allows_write_to_pm(aiogram.utils.web_app.WebAppU	sænswer_contact() (aiogram.types.message.Message
attribute), 561	<i>method</i>), 156
AMD (aiogram.enums.currency.Currency attribute), 461	answer_contact_pm()
<pre>amount (aiogram.types.labeled_price.LabeledPrice at-</pre>	(aiogram.types.chat_join_request.ChatJoinRequest
tribute), 279	method), 59
ANIMATED (aiogram.enums.sticker_format.StickerFormat attribute), 469	<pre>answer_dice() (aiogram.types.chat_join_request.ChatJoinRequest</pre>
ANIMATION (aiogram.enums.content_type.ContentType attribute), 459	<pre>answer_dice() (aiogram.types.chat_member_updated.ChatMemberUpdat method), 106</pre>
ANIMATION (aiogram.enums.input_media_type.InputMedia attribute), 465	aTappswer_dice() (aiogram.types.message.Message method), 174
	<pre>tianswer_dice_pm() (aiogram.types.chat_join_request.ChatJoinRequest method), 76</pre>
	ep lylsufe r_document() (aiogram.types.chat_join_request.ChatJoinRequest
attribute), 117	method), 59
animation (aiogram.types.game.Game attribute), 288	answer_document() (aiogram.types.chat_member_updated.ChatMemberl
animation (aiogram.types.message.Message attribute),	method), 98
146	answer_document() (aiogram.types.message.Message

method), 158	answer	pol1() (aiogram.types.chat_member_updated.ChatMemberUpdat
answer_document_pm()	answer_	method), 104
(aiogram.types.chat_join_request.ChatJoinReque	esanswer_	
method), 61		<i>method</i>), 172
	iraResqueest	<pre>poll_pm() (aiogram.types.chat_join_request.ChatJoinRequest</pre>
method), 62	h and the same has	method), 73
method), 99	UULINGENIEUV <u>e</u>	<pre>stipdned() (aiogram.types.chat_join_request.ChatJoinRequest method), 76</pre>
	answer	sticker() (aiogram.types.chat_member_updated.ChatMemberU
<i>method</i>), 159		method), 107
<pre>answer_game_pm() (aiogram.types.chat_join_request.Chat_join_r</pre>	at dn in Rea qu	
method), 63		<i>method</i>), 175
answer_invoice() (aiogram.types.chat_join_request.Cha	at an in Rea qu	
<pre>method), 63 answer_invoice() (aiogram.types.chat_member_update</pre>	d ChatMa	(aiogram.types.chat_join_request.ChatJoinRequest
method), 100		venue() (aiogram.types.chat_join_request.ChatJoinRequest
answer_invoice() (aiogram.types.message.Message	unswer_	method), 78
method), 162	answer_	venue() (aiogram.types.chat_member_updated.ChatMemberUpd
answer_invoice_pm()		method), 107
(aiogram.types.chat_join_request.ChatJoinReque	esanswer_	
method), 65		method), 177
answer_location() (alogram.types.cnat_join_request.c. method), 67	n ainswen e	<pre>quentue_pm() (aiogram.types.chat_join_request.ChatJoinRequest method), 79</pre>
	e d u Shvat M	evilled(), 19 evilled())datiagram.types.chat_join_request.ChatJoinRequest
method), 102		method), 80
	answer_	video() (aiogram.types.chat_member_updated.ChatMemberUpd
<i>method</i>), 164		<i>method</i>), 108
answer_location_pm()		video() (aiogram.types.message.Message
(aiogram.types.chat_join_request.ChatJoinReque	est	method), 179
(aiogram.types.chat_join_request.ChatJoinReque method), 68	est	<pre>method), 179 video_note()</pre>
(aiogram.types.chat_join_request.ChatJoinReque method), 68 answer_media_group()	est answer_	<pre>method), 179 video_note() (aiogram.types.chat_join_request.ChatJoinRequest</pre>
(aiogram.types.chat_join_request.ChatJoinReque method), 68	est answer_ est	<pre>method), 179 video_note()</pre>
<pre>(aiogram.types.chat_join_request.ChatJoinReque method), 68 answer_media_group() (aiogram.types.chat_join_request.ChatJoinReque method), 69 answer_media_group()</pre>	est answer_ est answer_	<pre>method), 179 video_note() (aiogram.types.chat_join_request.ChatJoinRequest method), 83 video_note() (aiogram.types.chat_member_updated.ChatMemberUpdated</pre>
<pre>(aiogram.types.chat_join_request.ChatJoinRequest method), 68 answer_media_group() (aiogram.types.chat_join_request.ChatJoinRequest method), 69 answer_media_group() (aiogram.types.chat_member_updated.ChatMematic method)</pre>	est answer_ est answer_ bberUpdate	<pre>method), 179 video_note() (aiogram.types.chat_join_request.ChatJoinRequest method), 83 video_note() (aiogram.types.chat_member_updated.ChatMemberUpdated ednethod), 110</pre>
<pre>(aiogram.types.chat_join_request.ChatJoinReque method), 68 answer_media_group() (aiogram.types.chat_join_request.ChatJoinReque method), 69 answer_media_group() (aiogram.types.chat_member_updated.ChatMem method), 103</pre>	est answer_ est answer_ bberUpdate	<pre>method), 179 video_note() (aiogram.types.chat_join_request.ChatJoinRequest method), 83 video_note() (aiogram.types.chat_member_updated.ChatMemberUpdated ednethod), 110 video_note()</pre>
<pre>(aiogram.types.chat_join_request.ChatJoinReque method), 68 answer_media_group() (aiogram.types.chat_join_request.ChatJoinReque method), 69 answer_media_group() (aiogram.types.chat_member_updated.ChatMem method), 103 answer_media_group()</pre>	est answer_ est answer_ bberUpdate	<pre>method), 179 video_note() (aiogram.types.chat_join_request.ChatJoinRequest method), 83 video_note() (aiogram.types.chat_member_updated.ChatMemberUpdated ednethod), 110 video_note() (aiogram.types.message.Message method),</pre>
<pre>(aiogram.types.chat_join_request.ChatJoinReque method), 68 answer_media_group() (aiogram.types.chat_join_request.ChatJoinReque method), 69 answer_media_group() (aiogram.types.chat_member_updated.ChatMem method), 103 answer_media_group() (aiogram.types.message.Message method),</pre>	est answer_ est answer_ berUpdate answer_	<pre>method), 179 video_note() (aiogram.types.chat_join_request.ChatJoinRequest method), 83 video_note() (aiogram.types.chat_member_updated.ChatMemberUpdated ednethod), 110 video_note() (aiogram.types.message.Message method), 181</pre>
<pre>(aiogram.types.chat_join_request.ChatJoinReque method), 68 answer_media_group() (aiogram.types.chat_join_request.ChatJoinReque method), 69 answer_media_group() (aiogram.types.chat_member_updated.ChatMem method), 103 answer_media_group() (aiogram.types.message.Message method), 166</pre>	est answer_ est answer_ berUpdate answer_	<pre>method), 179 video_note() (aiogram.types.chat_join_request.ChatJoinRequest method), 83 video_note() (aiogram.types.chat_member_updated.ChatMemberUpdated ednethod), 110 video_note() (aiogram.types.message.Message method), 181 video_note_pm()</pre>
<pre>(aiogram.types.chat_join_request.ChatJoinReque method), 68 answer_media_group() (aiogram.types.chat_join_request.ChatJoinReque method), 69 answer_media_group() (aiogram.types.chat_member_updated.ChatMem method), 103 answer_media_group() (aiogram.types.message.Message method),</pre>	est answer_ est answer_ bberUpdate answer_ answer_	<pre>method), 179 video_note() (aiogram.types.chat_join_request.ChatJoinRequest method), 83 video_note() (aiogram.types.chat_member_updated.ChatMemberUpdated ednethod), 110 video_note() (aiogram.types.message.Message method), 181</pre>
<pre>(aiogram.types.chat_join_request.ChatJoinReque method), 68 answer_media_group() (aiogram.types.chat_join_request.ChatJoinReque method), 69 answer_media_group() (aiogram.types.chat_member_updated.ChatMem method), 103 answer_media_group() (aiogram.types.message.Message method), 166 answer_media_group_pm() (aiogram.types.chat_join_request.ChatJoinReque method), 69</pre>	est answer_ est answer_ bberUpdate answer_ answer_ est answer_	<pre>method), 179 video_note() (aiogram.types.chat_join_request.ChatJoinRequest method), 83 video_note() (aiogram.types.chat_member_updated.ChatMemberUpdated ednethod), 110 video_note() (aiogram.types.message.Message method), 181 video_note_pm() (aiogram.types.chat_join_request.ChatJoinRequest method), 84 video_pm() (aiogram.types.chat_join_request.ChatJoinRequest</pre>
<pre>(aiogram.types.chat_join_request.ChatJoinReque method), 68 answer_media_group() (aiogram.types.chat_join_request.ChatJoinReque method), 69 answer_media_group() (aiogram.types.chat_member_updated.ChatMem method), 103 answer_media_group() (aiogram.types.message.Message method), 166 answer_media_group_pm() (aiogram.types.chat_join_request.ChatJoinReque method), 69 answer_photo() (aiogram.types.chat_join_request.ChatJ</pre>	est answer_ est answer_ berUpdate answer_ answer_ est answer_ foinReques	<pre>method), 179 video_note() (aiogram.types.chat_join_request.ChatJoinRequest method), 83 video_note() (aiogram.types.chat_member_updated.ChatMemberUpdated ednethod), 110 video_note() (aiogram.types.message.Message method), 181 video_note_pm() (aiogram.types.chat_join_request.ChatJoinRequest method), 84 video_pm() (aiogram.types.chat_join_request.ChatJoinRequest timethod), 81</pre>
<pre>(aiogram.types.chat_join_request.ChatJoinReque method), 68 answer_media_group() (aiogram.types.chat_join_request.ChatJoinReque method), 69 answer_media_group() (aiogram.types.chat_member_updated.ChatMem method), 103 answer_media_group() (aiogram.types.message.Message method), 166 answer_media_group_pm() (aiogram.types.chat_join_request.ChatJoinReque method), 69 answer_photo() (aiogram.types.chat_join_request.ChatJ method), 70</pre>	est answer_ est answer_ dberUpdate answer_ answer_ est answer_ toinReques answer_	<pre>method), 179 video_note() (aiogram.types.chat_join_request.ChatJoinRequest method), 83 video_note() (aiogram.types.chat_member_updated.ChatMemberUpdated ednethod), 110 video_note() (aiogram.types.message.Message method), 181 video_note_pm() (aiogram.types.chat_join_request.ChatJoinRequest method), 84 video_pm() (aiogram.types.chat_join_request.ChatJoinRequest stmethod), 81 voice() (aiogram.types.chat_join_request.ChatJoinRequest</pre>
<pre>(aiogram.types.chat_join_request.ChatJoinReque method), 68 answer_media_group() (aiogram.types.chat_join_request.ChatJoinReque method), 69 answer_media_group() (aiogram.types.chat_member_updated.ChatMem method), 103 answer_media_group() (aiogram.types.message.Message method), 166 answer_media_group_pm() (aiogram.types.chat_join_request.ChatJoinReque method), 69 answer_photo() (aiogram.types.chat_join_request.ChatJ method), 70 answer_photo() (aiogram.types.chat_member_updated.ChatMem method), 70</pre>	est answer_ est answer_ berUpdata answer_ answer_ coinReques answer_ ChatMemb	<pre>method), 179 video_note() (aiogram.types.chat_join_request.ChatJoinRequest method), 83 video_note() (aiogram.types.chat_member_updated.ChatMemberUpdated ednethod), 110 video_note() (aiogram.types.message.Message method), 181 video_note_pm() (aiogram.types.chat_join_request.ChatJoinRequest method), 84 video_pm() (aiogram.types.chat_join_request.ChatJoinRequest extmethod), 81 voice() (aiogram.types.chat_join_request.ChatJoinRequest enléptudèd85</pre>
<pre>(aiogram.types.chat_join_request.ChatJoinReque method), 68 answer_media_group() (aiogram.types.chat_join_request.ChatJoinReque method), 69 answer_media_group() (aiogram.types.chat_member_updated.ChatMem method), 103 answer_media_group() (aiogram.types.message.Message method), 166 answer_media_group_pm() (aiogram.types.chat_join_request.ChatJoinReque method), 69 answer_photo() (aiogram.types.chat_join_request.ChatJ method), 70 answer_photo() (aiogram.types.chat_member_updated.ChatMem method), 103</pre>	est answer_ est answer_ berUpdata answer_ answer_ coinReques answer_ ChatMemb	<pre>method), 179 video_note() (aiogram.types.chat_join_request.ChatJoinRequest method), 83 video_note() (aiogram.types.chat_member_updated.ChatMemberUpdated ednethod), 110 video_note() (aiogram.types.message.Message method), 181 video_note_pm() (aiogram.types.chat_join_request.ChatJoinRequest method), 84 video_pm() (aiogram.types.chat_join_request.ChatJoinRequest stmethod), 81 voice() (aiogram.types.chat_join_request.ChatJoinRequest</pre>
<pre>(aiogram.types.chat_join_request.ChatJoinReque method), 68 answer_media_group() (aiogram.types.chat_join_request.ChatJoinReque method), 69 answer_media_group() (aiogram.types.chat_member_updated.ChatMem method), 103 answer_media_group() (aiogram.types.message.Message method), 166 answer_media_group_pm() (aiogram.types.chat_join_request.ChatJoinReque method), 69 answer_photo() (aiogram.types.chat_join_request.ChatJ method), 70 answer_photo() (aiogram.types.chat_member_updated.ChatMem method), 70</pre>	est answer_ est answer_ berUpdate answer_ answer_ coinReques answer_ ChatMemb answer_	<pre>method), 179 video_note() (aiogram.types.chat_join_request.ChatJoinRequest method), 83 video_note() (aiogram.types.chat_member_updated.ChatMemberUpdated ednethod), 110 video_note() (aiogram.types.message.Message method), 181 video_note_pm() (aiogram.types.chat_join_request.ChatJoinRequest method), 84 video_pm() (aiogram.types.chat_join_request.ChatJoinRequest etmethod), 81 voice() (aiogram.types.chat_join_request.ChatJoinRequest enléplad)d85 voice() (aiogram.types.chat_member_updated.ChatMemberUpdated</pre>
<pre>(aiogram.types.chat_join_request.ChatJoinReque method), 68 answer_media_group() (aiogram.types.chat_join_request.ChatJoinReque method), 69 answer_media_group() (aiogram.types.chat_member_updated.ChatMem method), 103 answer_media_group() (aiogram.types.message.Message method), 166 answer_media_group_pm() (aiogram.types.chat_join_request.ChatJoinReque method), 69 answer_photo() (aiogram.types.chat_join_request.ChatJ method), 70 answer_photo() (aiogram.types.chat_member_updated.ChatJoinRequest method), 103 answer_photo() (aiogram.types.chat_member_updated.ChatJoinRequest method), 103 answer_photo() (aiogram.types.chat_member_updated.ChatJoinRequest method), 103 answer_photo() (aiogram.types.chat_join_request.ChatJoinRequest method), 169 answer_photo_pm() (aiogram.types.chat_join_request.ChatJoinRequest method), 169</pre>	est answer_ est answer_ derUpdate answer_ answer_ coinReques answer_ ChatMemb answer_ answer_ hatJoinRed	<pre>method), 179 video_note() (aiogram.types.chat_join_request.ChatJoinRequest method), 83 video_note() (aiogram.types.chat_member_updated.ChatMemberUpdated ednethod), 110 video_note() (aiogram.types.message.Message method), 181 video_note_pm() (aiogram.types.chat_join_request.ChatJoinRequest method), 84 video_pm() (aiogram.types.chat_join_request.ChatJoinRequest enléptdod), 81 voice() (aiogram.types.chat_join_request.ChatJoinRequest enléptdod), 81 voice() (aiogram.types.chat_join_request.ChatJoinRequest enléptdod), 81 voice() (aiogram.types.chat_join_request.ChatJoinRequest enléptdod), 81 voice() (aiogram.types.chat_member_updated.ChatMemberUpde method), 111 voice() (aiogram.types.message.Message questhod), 183</pre>
<pre>(aiogram.types.chat_join_request.ChatJoinReque method), 68 answer_media_group() (aiogram.types.chat_join_request.ChatJoinReque method), 69 answer_media_group() (aiogram.types.chat_member_updated.ChatMem method), 103 answer_media_group() (aiogram.types.message.Message method), 166 answer_media_group_pm() (aiogram.types.chat_join_request.ChatJoinReque method), 69 answer_photo() (aiogram.types.chat_join_request.ChatJ method), 70 answer_photo() (aiogram.types.chat_member_updated.ChatMem method), 103 answer_photo() (aiogram.types.chat_member_updated.ChatMem method), 103 answer_photo() (aiogram.types.chat_member_updated.ChatMem method), 103 answer_photo() (aiogram.types.chat_join_request.ChatMem method), 103 answer_photo() (aiogram.types.chat_join_request.ChatMem method), 103 answer_photo() (aiogram.types.chat_join_request.ChatMem method), 169 answer_photo_pm() (aiogram.types.chat_join_request.ChatMem method), 71</pre>	est answer_ est answer_ dberUpdate answer_ answer_ coinReques answer_ ChatMemb answer_ hatJoinReq answer_	<pre>method), 179 video_note() (aiogram.types.chat_join_request.ChatJoinRequest method), 83 video_note() (aiogram.types.chat_member_updated.ChatMemberUpdated ednethod), 110 video_note() (aiogram.types.message.Message method), 181 video_note_pm() (aiogram.types.chat_join_request.ChatJoinRequest method), 84 video_pm() (aiogram.types.chat_join_request.ChatJoinRequest emléphiad), 81 voice() (aiogram.types.chat_join_request.ChatJoinRequest emléphiad)d85 voice() (aiogram.types.chat_member_updated.ChatMemberUpdated method), 111 voice() (aiogram.types.chat_member_updated.ChatMemberUpdated method), 111 voice() (aiogram.types.chat_member_updated.ChatMemberUpdated method), 113 voice() (aiogram.types.chat_member_updated.ChatMemberUpdated method), 113 voice() (aiogram.types.chat_join_request.ChatJoinRequest emléphiad), 183 voice_pm() (aiogram.types.chat_join_request.ChatJoinRequest</pre>
<pre>(aiogram.types.chat_join_request.ChatJoinReque method), 68 answer_media_group() (aiogram.types.chat_join_request.ChatJoinReque method), 69 answer_media_group() (aiogram.types.chat_member_updated.ChatMem method), 103 answer_media_group() (aiogram.types.message.Message method), 166 answer_media_group_pm() (aiogram.types.chat_join_request.ChatJoinReque method), 69 answer_photo() (aiogram.types.chat_join_request.ChatJ method), 70 answer_photo() (aiogram.types.chat_member_updated.ChatBer method), 103 answer_photo() (aiogram.types.chat_member_updated.ChatBer method), 103 answer_photo() (aiogram.types.chat_member_updated.ChatBer method), 103 answer_photo() (aiogram.types.chat_join_request.ChatBer method), 169 answer_photo() (aiogram.types.chat_join_request.ChatBer method), 71 answer_pm() (aiogram.types.chat_join_request.ChatBoinEver method), 71 answer_pm() (aiogram.types.chat_join_request.ChatBoinEver method), 71 answer_pm() (aiogram.types.chat_join_request.ChatBoinEver method), 71</pre>	est answer_ est answer_ berUpdate answer_ answer_ chatMemb answer_ hatJoinRed answer_ hatJoinRed answer_ Request	<pre>method), 179 video_note() (aiogram.types.chat_join_request.ChatJoinRequest method), 83 video_note() (aiogram.types.chat_member_updated.ChatMemberUpdated ednethod), 110 video_note() (aiogram.types.message.Message method), 181 video_note_pm() (aiogram.types.chat_join_request.ChatJoinRequest method), 84 video_pm() (aiogram.types.chat_join_request.ChatJoinRequest enléptidded85 voice() (aiogram.types.chat_member_updated.ChatMemberUpdated enléptidded85 voice() (aiogram.types.chat_member_updated.ChatMemberUpdated enléptidded85 voice() (aiogram.types.chat_member_updated.ChatMemberUpdated method), 111 voice() (aiogram.types.chat_member_updated.ChatMemberUpdated method), 183 voice_pm() (aiogram.types.chat_join_request.ChatJoinRequest method), 183 voice_pm() (aiogram.types.chat_join_request.ChatJoinRequest method), 183</pre>
<pre>(aiogram.types.chat_join_request.ChatJoinReque method), 68 answer_media_group() (aiogram.types.chat_join_request.ChatJoinReque method), 69 answer_media_group() (aiogram.types.chat_member_updated.ChatMem method), 103 answer_media_group() (aiogram.types.message.Message method), 166 answer_media_group_pm() (aiogram.types.chat_join_request.ChatJoinReque method), 69 answer_photo() (aiogram.types.chat_join_request.ChatJ method), 70 answer_photo() (aiogram.types.chat_member_updated.ChatBer method), 103 answer_photo() (aiogram.types.chat_member_updated.ChatBer method), 103 answer_photo() (aiogram.types.chat_member_updated.ChatBer method), 103 answer_photo() (aiogram.types.chat_join_request.ChatBer method), 103 answer_photo() (aiogram.types.chat_join_request.ChatBer method), 169 answer_photo_pm() (aiogram.types.chat_join_request.ChatBer method), 71 answer_pm() (aiogram.types.chat_join_request.ChatBoinEr method), 52</pre>	est answer_ est answer_ berUpdate answer_ answer_ coinReques answer_ ChatMemb answer_ hatJoinRea answer_ hatJoinRea AnswerC	<pre>method), 179 video_note() (aiogram.types.chat_join_request.ChatJoinRequest method), 83 video_note() (aiogram.types.chat_member_updated.ChatMemberUpdated ednethod), 110 video_note() (aiogram.types.message.Message method), 181 video_note_pm() (aiogram.types.chat_join_request.ChatJoinRequest method), 84 video_pm() (aiogram.types.chat_join_request.ChatJoinRequest timethod), 81 voice() (aiogram.types.chat_join_request.ChatJoinRequest enliptiond)d85 voice() (aiogram.types.chat_member_updated.ChatMemberUpdated method), 111 voice() (aiogram.types.chat_join_request.ChatJoinRequest enliptiond), 183 voice_pm() (aiogram.types.chat_join_request.ChatJoinRequest method), 183 voice() (aiogram.types.chat_join_request.ChatJoinRequest method), 183 voice_pm() (aiogram.types.chat_join_request.ChatJoinRequest method), 86 callbackQuery (class in</pre>
<pre>(aiogram.types.chat_join_request.ChatJoinReque method), 68 answer_media_group() (aiogram.types.chat_join_request.ChatJoinReque method), 69 answer_media_group() (aiogram.types.chat_member_updated.ChatMem method), 103 answer_media_group() (aiogram.types.message.Message method), 166 answer_media_group_pm() (aiogram.types.chat_join_request.ChatJoinReque method), 69 answer_photo() (aiogram.types.chat_join_request.ChatJ method), 70 answer_photo() (aiogram.types.chat_member_updated.ChatBer method), 103 answer_photo() (aiogram.types.chat_member_updated.ChatBer method), 103 answer_photo() (aiogram.types.chat_member_updated.ChatBer method), 103 answer_photo() (aiogram.types.chat_join_request.ChatBer method), 169 answer_photo() (aiogram.types.chat_join_request.ChatBer method), 71 answer_pm() (aiogram.types.chat_join_request.ChatBoinEver method), 71 answer_pm() (aiogram.types.chat_join_request.ChatBoinEver method), 71 answer_pm() (aiogram.types.chat_join_request.ChatBoinEver method), 71</pre>	est answer_ est answer_ berUpdate answer_ answer_ coinReques answer_ ChatMemb answer_ hatJoinRea answer_ hatJoinRea AnswerC	<pre>method), 179 video_note() (aiogram.types.chat_join_request.ChatJoinRequest method), 83 video_note() (aiogram.types.chat_member_updated.ChatMemberUpdated ednethod), 110 video_note() (aiogram.types.message.Message method), 181 video_note_pm() (aiogram.types.chat_join_request.ChatJoinRequest method), 84 video_pm() (aiogram.types.chat_join_request.ChatJoinRequest timethod), 81 voice() (aiogram.types.chat_join_request.ChatJoinRequest enliptiond)d85 voice() (aiogram.types.chat_member_updated.ChatMemberUpdated method), 111 voice() (aiogram.types.chat_join_request.ChatJoinRequest enliptiond), 183 voice_pm() (aiogram.types.chat_join_request.ChatJoinRequest method), 183 voice() (aiogram.types.chat_join_request.ChatJoinRequest method), 183 voice_pm() (aiogram.types.chat_join_request.ChatJoinRequest method), 86 callbackQuery (class in</pre>

property), 555 AUDIO (aiogram.endios.ims.input_media_type_InputMediaType AnswerInclineQuery (class aiogram.methods.answer_intine_query), audio (aiogram.methods.answer_pre_checkou_query), audio (aiogram.types.caternal_reply_info.ExternalReplyInfo aiogram.methods.answer_pre_checkou_query), audio (aiogram.types.caternal_reply_info.ExternalReplyInfo aiogram.methods.answer_wer_pre_checkou_query), audio (aiogram.types.caternal_reply_info.ExternalReplyInfo atistibute, 117 audio (clasgram.types.audio), 18 adorgram.methods.answer_web_app.query), audio (clasgram.types.inline_query_result_audio.InlineQueryRe AnswerBebAppQuery (class (aiogram.neuns.content_type ContentType authiote), 223 audio_cligoram.types.inline_query_result_audio.InlineQueryResultAi author_signature (aiogram.types.inline_query_result_audio.InlineQueryResultAi author_signature (aiogram.types.inline_query_result_audio.InlineQueryResultAi author_signature (aiogram.types.message_origin_channel.MessageOriginC author_signature (aiogram.types.chat.font author_signature (aiogram.types.chat.font author_signature (aiogram.types.chat.font author_signature (aiogram.types.chat.font author_signature (aiogram.types.chat.font author_signature (aiogram.types.chat.font author_signature, 465 asplattor (aiogram.types.chat.font	answered (aiogram.utils.callback_answer.CallbackAnswer	r attribute), 465
 audio (aiogram.methods.send_audio.Send_audio.SendAudio at- rrbuet, 364 AnswerPreCheckoutQuery (class in aiogram.methods.answer_pre_checkout_query), 442 AnswerShippingQuery (class in aiogram.types.message.Message attribute), 146 AnswerNebAppQuery (class in aiogram.types.message.attribute), 146 AnswerNebAppQuery (class in aiogram.types.inline_query_result_audio.InlineQueryRe attribute), 221 AnswerNebAppQuery (class in aiogram.methods.answer_web_app_query), 443 AnswerNebAppQuery (class in aiogram.nethods.answer_web_app_query), 445 AnswerNebAppQuery (class in aiogram.nethods.answer_web_app_query), 446 AnswerNebAppQuery (class in aiogram.types.content_type.ContentType at- tribute), 221 auth_date (aiogram.types.inline_query_result_audio.InlineQueryResultAi attribute), 221 auth_date (aiogram.types.message.Message approveQ (aiogram.types.chat_join_request.ChatJoinRequerst attribute), 146 author_slignature (aiogram.types.message_origin_chantel.MessageOriginC attribute), 194 author_slignature (aiogram.types.message_origin_chantel.MessageOriginC attribute), 194 args (aiogram.methods.approve_chat.join_request. 1511 as_hartel (aiogram.utils.formatting, 1567 as_hartel (aiogram.utils.formatting, 1567 as_marked_stertion (in module aiogram.utils.formatting), 567 as_marked_stertion (in module aiogram.utils.formatting, 1567 as_marked_stert		
443 rinhun, 364 AnswerSrecheckoutQuery audio (alogram.prpes.setteral.reply_info.ExternalReplyInfo 442 audio (alogram.prpes.message.Message attribute), 146 AnswerShippingQuery (class 443 audio (alogram.prpes.methods.answer_shipping.guery), 443 audio (alogram.types.inline_guery_result_audio.InlineQueryR 443 audio_file_i4(alogram.types.inline_guery_result_audio.InlineQueryR 443 audio_file_i4(alogram.types.inline_guery_result_audio.InlineQueryR 443 audio_url (alogram.types.inline_guery_result_audio.InlineQueryResult/a 443 autio_url (alogram.types.inline_guery_result_audio.InlineQueryResult/a 443 autio_url (alogram.types.inline_guery_result_audio.InlineQueryResult/a 444 autio_url (alogram.types.inline_guery_result_audio.InlineQueryResult/a 445 autio_url (alogram.types.inline_guery_result_audio.InlineQueryResult/a 446 autio_url (alogram.types.inline_guery_resul	AnswerInlineQuery (class in	attribute), 465
AnswerPreCheckoutQuery (class in audio (alogram.types.netsrage.Message attribute), 146 AnswerShipping.query (class in Audio (class in alogram.types.message.Message attribute), 146 AnswerShipping.query (class in aloi o (alogram.types.message.Message attribute), 146 AnswerShipping.query (class in audio (alogram.types.inline_query_result_audio.InlineQueryReattribute), 221 AnswerShipping.query (class in audio (alogram.types.inline_query_result_audio.InlineQueryReattribute), 221 AnswerShipping.query (class in audio_urit (alogram.types.inline_query_result_audio.InlineQueryReattribute), 221 autio_urit (alogram.types.inline_query_result_audio.InlineQueryReattribute), 450 autio_scipan.types.inline_query_result_audio.InlineQueryReattribute), 221 approveChat JoinRequest attribute), 501 author_signature (alogram.types.message_origin_chan.NessageOriginC alogram.methods.approve_chat.join_request), 501 author_signature (alogram.types.chat.Chat artribute), 486 author_signature (alogram.types.message_origin_chat.MessageOriginC attribute), 481 author_signature (alogram.types.chat.Chat ApproveChat JoinRequest attribute), 461 AZN (alogram.enums.currency.Currency attribute), 461 ARS (alogram.enums.tine_query_result_sype.InlineQueryResult), 94		
 aiogram.methods.answer_pre_checkout_query), aitribate', 117 audio (aiogram.types.message.Message attribute, 146 AnswerShippingQuery (class in aiogram.types.message.Message attribute, 221 audio_duration (aiogram.types.inline_query_result_audio.InlineQueryResult/sectors), 221 audio_fle_ide(aiogram.types.inline_query_result_audio.InlineQueryResult/sectors), 221 audio_fle_ide(aiogram.types.inline_query_result_audio.InlineQueryResult/sectors), 221 audio_fle_ide(aiogram.types.inline_query_result_audio.InlineQueryResult/sectors), 221 audio_fle_ide(aiogram.types.inline_query_result_audio.InlineQueryResult/sectors), 221 autio_fle_ide(aiogram.types.inline_query_result_audio.InlineQueryResult/sectors), 221 autio_fle_ide(aiogram.types.inline_query_result_audio.InlineQueryResult/sectors), 221 autio_fle_ide(aiogram.types.inline_query_result_audio.InlineQueryResult/sectors), 221 autio_fle_ide(aiogram.types.inline_query_result_audio.InlineQueryResult/sectors), 221 autio_fle_ide(aiogram.types.inline_query_result_audio.InlineQueryResult/sectors), 221 autio_fle_ide(aiogram.types.inline_query_result_sectors), 221 autio_fle_ide(aiogram.types.inline_query_result_sectors), 221 aution_signam.tel(aiogram.types.inline_query_result_audio.InlineQueryResult/sectors), 221 autio_fle_ide(aiogram.types.inline_query_result_audio.InlineQueryResult/sectors), 221 autio_fle_ide(aiogram.types.inline_guery_result_audio.InlineQueryResult/sectors), 221 autio_fle_ide(aiogram.types.inline_guery_result_audio.InlineQueryResult/sectors), 221 autio_fle_ide(aiogram.types.inline_guery_result_audio.InlineQueryResult/sectors), 221 autio_fle_ide(aiogram.types.inline_guery_result_aution_inline_guery_result_sectors), 221 autio_fle_ide(aiogram.types.inline_guery_result_aut	AnswerPreCheckoutQuery (class in	
 442 Answer Vehspip Query (class in alogram.rtpes.message.Message articute, 146 Answer Vehspip Query (class in alogram.rtpes.inline_query_result_audio.InlineQueryR attribute), 221 Answer Vehspip Query (class in alogram.rtpes.inline_query_result_audio.InlineQueryR attribute), 221 audio _duration (alogram.rtpes.inline_query_result_audio.InlineQueryR attribute), 223 audio _duration (alogram.rtpes.inline_query_result_audio.InlineQueryResultAu attribute), 561 autio_signam.tils.scommand.CommandObject attribute), 454 ARS (alogram.nems.currency: Currency attribute), 461 ARS (alogram.nems.currency: Currency attribute), 456 ARS (alogram.nems.currency: Currency attribute), 451 ARS (alogram.utils.formatting.Text method), 557 as_nanted_list() (in module alogram.utils.formatting), 567 as_numbered_list() (in module alogram.utils.formatting), 567		
AnswerShippingQuery (class in Audio (class in alogram.types.andio), 18 aiogram.methods.answer_shipping_query), 43 AnswerWebAppQuery (class (aiogram.nenums.content_type.class in audio_duration (aiogram.types.inline_query_result_cached_audio.InlineQueryR 436 audio_duration (aiogram.types.inline_query_result_cached_audio.InlineQueryR 437 audio_duration (aiogram.types.inline_query_result_cached_audio.InlineQueryR 438 audio_duration (aiogram.types.inline_query_result_cached_audio.InlineQueryR 439 audio_duration (aiogram.types.inline_query_result_audio.InlineQueryR 439 audio_duration (aiogram.types.inline_query_result_audio.InlineQueryR 436 audio_duration (aiogram.types.inline_query_result_cached_audio.InlineQueryR 438 audio_duration (aiogram.types.inline_query_result_audio.InlineQueryR 439 audio_duration (aiogram.types.inline_query_result_audio.InlineQueryR autio_triation, 221 autio_triation, 221 autio_triation, 30 autior_signature (aiogram.types.message.origin_channel.MessageOrigin		audio (aiogram.types.message.Message attribute), 146
 443 AnswerWebAppQuery (class in alignmentions.comment/obs.answer_web_app_query). 436 AnswerWebAppQuery (class in alignmentions.comment_type.contentType atribute), 423 autio_url (alogram.types.inline_query_result_audio.InlineQueryResultAtions), 124 autio_url (alogram.types.inline_query_result_audio.InlineQueryResultAtions), 124 autio_url (alogram.types.inline_query_result_audio.InlineQueryResultAtions), 124 autio_url (alogram.types.inline_query_result_audio.InlineQueryResultAtions), 124 autio_url (alogram.types.that.goin_request. ChatJoinRequest atribute), 146 autior_signature (alogram.types.message_origin_chat.MessageOrigin attribute), 146 autior_signature (alogram.types.message_origin_chat.MessageOrigin attribute), 146 autior_signature (alogram.types.message_origin_chat.MessageOrigin attribute), 146 autior_signature (alogram.types.chat.Chat attribute), 461 AES (alogram.enums.currency.Currency attribute), 461 AES (alogram.enums.inline_query_result_type.InlineQueryResultType attribute), 461 AES (alogram.enums.inline_query_result_type.InlineQueryResultType as_html O (alogram.tills.formatting.Text method), 507 as_markdwary (alogram.utils.formatting), 567 as_numbered_list() (in module alogram.utils.formatting), 567 as_numbered_section() (in module a	AnswerShippingQuery (class in	Audio (class in aiogram.types.audio), 18
aiogram.methods.answer_web_app_query), 436 ANY (aiogram.enums.content_type.ContentType at- tribute), 459 appi_url() (aiogram.tile.telegram.TelegramAPIServer method), 12 approve() (aiogram.types.chat_join_request.ChatJoinRequest method), 50 ApproveChatJoinRequest (class in altor_signature (aiogram.types.message_origin_chat.MessageOrigin_ author_signature (aiogram.types.message_origin_chat.MessageOrigin_ author_signature (aiogram.types.message_origin_chat.MessageOrigin_ author_signature (aiogram.types.message_origin_chat.MessageOrigin_ author_signature (aiogram.types.message_origin_chat.MessageOrigin_ author_signature (aiogram.types.chat.Chat author_signature (aiogram.types.chat.Chat author_signature (aiogram.types.chat.Chat author_signature(aiogram.types.chat.Chat author_signature(aiogram.types.chat.Chat author_signature(aiogram.types.chat.Chat author_signature(aiogram.types.chat.Chat author_signature(aiogram.types.chat.Chat author_signature(aiogram.types.chat.Chat author_signature(aiogram.types.chat.Chat author_signature(aiogram.types.chat.Chat author_signature(aiogram.types.chat.Chat author_signature(aiogram.types.chat.Chat author_signature(aiogram.types.chat.Chat author_signature(aiogram.types.chat.Chat author_signature(aiogram.types.chat.Chat method), 53 back(O (aiogram.types.chat.Chat method), 51 backGO (aiogram.types.chat.Chat method), 53 as_list() (in module aiogram.utils.formatting), 567 as_marked_section() (in module aiogram.utils.formatting), 567 as_numbered_list() (in module aiogram.utils.formatting), 567 as_numbered_section() (in module aiogram.utils.formatting), 567 as_num		
 436 ANY (aiogram.enums.content_type.ContentType at tribute), 459 apj_url() (aiogram.client.telegram.TelegramAPIServer method), 50 approve() (aiogram.types.chat_join_request.ChatJoinRequest attribute), 146 author_signature (aiogram.types.message_origin_channel.MessageOrigin attribute), 146 author_signature (aiogram.types.message_origin_channel.MessageOrigin attribute), 146 author_signature (aiogram.types.message_origin_channel.MessageOrigin attribute), 146 author_signature (aiogram.types.message_origin_channel.MessageOrigin attribute), 194 author_signature (aiogram.types.message_origin_channel.MessageOrigin attribute), 194 author_signature (aiogram.types.message_origin_channel.MessageOrigin attribute), 146 author_signature (aiogram.types.message_origin_channel.MessageOrigin attribute), 146 author_signature (aiogram.types.message_origin_channel.MessageOrigin attribute), 194 author_signature (aiogram.types.chat.Chat attribute), 146 ARS (aiogram.enums.currency.Currency attribute), 461 ARTICLE (aiogram.utils.formatting, Text method), 570 as_markdown() (aiogram.utils.formatting, 567 as_marked_list() (in module aiogram.utils.formatting, 567 as_numbered_list() (in module aiogram.utils.formatting, 567 as_numbered_list() (in module aiogram.utils.formatting, 567 as_numbered_list() (in module aiogram.utils.formatting, 567 as_numbered_section() (in module aiogram.utils.formatting, 567 as_section() (in module aiogram.utils.formatting	AnswerWebAppQuery (class in	<pre>audio_file_id(aiogram.types.inline_query_result_cached_audio.InlineQ</pre>
 ANY (aiogram.enums.content_type.ContentType at tribute), 459 argi_urll() (aiogram.telegram.TelegramAPI/Server method), 12 approve() (aiogram.types.chat_join_request.ChatJoinRequest (class in acigram.methods.approve_chat_join_request), 30 ApproveChatJoinRequest (class in acigram.types.message_origin_chat.MessageOrigin attribute), 194 args (aiogram.filters.command.CommandObject atribute), 456 ARS (aiogram.enums.currency.Currency attribute), 461 ARTICLE (aiogram.nums.currency.Currency attribute), 461 ARTICLE (aiogram.multis.formatting.Text method), 570 as_handler() (aiogram.utils.formatting), 567 as_marked_list() (in module aiogram.utils.formatting), 567 as_numbered_section() (in module aiogram.utils.formatting), 567 as_section() (in module aiogram.utils.formatting), 567 as_sec		
api_url() (aiogram.client.telegram.TelegramAPIServer method), 12 author_signature (aiogram.types.message_Message author_signature (aiogram.types.message_origin_channel.MessageOrigin author_signature (aiogram.types.message_origin_channel.MessageOrigin author_signature (aiogram.types.message_origin_channel.MessageOrigin author_signature (aiogram.types.message_origin_chat.MessageOrigin author_signature (aiogram.types.message_origin_chat.MessageOrigin author_signature (aiogram.types.message_origin_chat.MessageOrigin author_signature (aiogram.types.message_origin_chat.MessageOrigin author_signature (aiogram.types.message_origin_chat.MessageOrigin author_signature (aiogram.types.chat.Chat atribute), 194 args (aiogram.enums.currency.Currency attribute), 461 ARTICLE (aiogram.enums.currency.Currency attribute), 461 ARTICLE (aiogram.sils.formatting.Text method), 570 as_html() (aiogram.utils.formatting.text method), 570 as_marked_section() (in module aiogram.utils.formatting.formatting.for as_marked_section() (in module aiogram.utils.formatting), 567 as_marked_list() (in module aiogram.utils.formatting), 567 as_marked_section() (in module aiogram.utils.formatting), 567 as_marked_list() (in module aiogram.utils.formatting), 567 as_marked_list() (in module aiogram.utils.formatting), 567 as_marked_section() (in module aiogram.utils.formatting), 567 as_marked_list() (in module aiogram.utils.formatting), 567 as_section() (in module aiogram.utils.formatting), 567 as_section(
method), 12author_signature(aiogram.types.message.Messageapprove() (aiogram.types.chat.join_request.ChatJoinRequestattribute), 146author_signature (aiogram.types.message_origin_channel.MessageOrigApproveChatJoinRequest(classaiogram.methods.approve_chat_join_request),author_signature (aiogram.types.message_origin_chat.MessageOriginCautor_signature (aiogram.types.message_origin_chat.MessageOriginCautor_signature (aiogram.types.chat.Chatautor_signature (aiogram.types.chat.Chatargs (aiogram.methods.approve_chat_join_request),autor_signature (aiogram.types.chat.Chatargs (aiogram.methods.approve_chat.join_request),autor_signature (aiogram.types.chat.Chatargs (aiogram.methods.approve_chat.join_request),autor_signature (aiogram.types.chat.Chatargs (aiogram.methods.approve_chat.join_request),autor_signature (aiogram.types.chat.Chatargs (aiogram.methods.approve_chat.join_request),args (aiogram.enums.currency.Currency attribute), 461ARTICLE (aiogram.enums.currency.Currency attribute), 461ARTICLE (aiogram.utils.formatting.Text method),569as_line() (in module aiogram.utils.formatting), 567as_marked.list() (in moduleas_marked.list() (in moduleaiogram.utils.formatting), 567as_marked_section() (in moduleaiogram.utils.formatting), 567as_numbered_section() (in moduleaiogram.utils.formatting), 567as_numbered_section() (in moduleaiogram.utils.formatting), 567as_numbered_section() (in moduleas_numbered_sect	tribute), 459	<pre>auth_date (aiogram.utils.web_app.WebAppInitData at-</pre>
approve() (alogram.types.chat_join_request_ChatJoinRequest_attribute), 146 method), 50 author_signature (alogram.types.message_origin_channel.MessageOrig attribute), 194 args (alogram.methods.approve_chat_join_request), 308 args (alogram.filters.command.CommandObject_attribute), 194 args (alogram.methods.approve_chat_join_request), 308 args (alogram.methods.approve_chat_join_request), 308 args (alogram.filters.command.CommandObject_attribute), 461 ARTICLE (alogram.enums.currency.Currency attribute), 461 ARTICLE (alogram.enums.currency.Currency attribute), 461 ARTICLE (alogram.types.chat.Chat_attribute), 30 as_html() (alogram.utils.formatting,Text_method), 570 as_ilist() (in module alogram.utils.formatting), 567 as_numbered_list() (in module alogram.utils.formatting), 567 as_numbered_section() (in module alogram.utils.formatting), 567 as_numbered_list() (in module alogram.utils.formatting), 567 as_numbered_list() (in module alogram.utils.formatting), 567 as_numbered_list() (in module alogram.utils.formatting), 567 as_numbered_section() (in module alogram.utils.formatting), 567 as_numbered_section() (in module alogram.utils.formatting), 567 as_scection() (in module alogram.utils.formatting), 567 as_sce	<pre>api_url() (aiogram.client.telegram.TelegramAPIServer</pre>	tribute), 561
method), 50author_signature (aiogram.types.message_origin_channel.MessageOrigApproveChatJoinRequest(classinaiogram.methods.approve_chat_join_request), 308author_signature (aiogram.types.message_origin_chat.MessageOriginC attribute), 194args(aiogram.filters.command.CommandObject attribute), 486author_signature (aiogram.types.message_origin_chat.MessageOriginC attribute), 194args(aiogram.enums.currency.Currency attribute), 461ARTICLE (aiogram.enums.inline_query_result_type.InlineQueryResultTypeARS (aiogram.enums.inline_query_result_type.InlineQueryResultTypeBas_handler() (aiogram.sins.scene.Scene Class method), 531Back() (aiogram.types.chat.Chat attribute), 461Sa_bandler() (aiogram.utils.formatting.Text method), 569Bas_list() (in module aiogram.utils.formatting, 567Back() (aiogram.types.chat.Chat method), 41 bas_marked_list() (in module aiogram.utils.formatting), 567BanChatKember(class aiogram.enums.encryptes.chat.Chat method), 33as_numbered_section() as_numbered_section() (in aiogram.utils.formatting), 567BackY attribute), 464 baseBack (aiogram.enums.encrypted_passport_element.Encrypted attribute), 12as_numbered_section() as_numbered_section() (in as_numbered_section() (in as_numbered_section() (in as_numbered_section() (in as_numbered_section() (in as_numbered_section() (in as_numbered_section() (in as_numbered_section() (in as_numbered_section() (in as_numbered_section() (in as_numbered_section() (in as_numbered_section() (in as_numbered_section() (in as_numbered_section() (in as_numbered_section() (in as_numbered_section() <td></td> <td></td>		
ApproveChat JoinRequest (class in atribute), 194 aiggram.methods.approve_chat_join_request), author_signature (alogram.types.message_origin_chat.MessageOriginC 308 author_signature (alogram.types.message_origin_chat.MessageOriginC args (alogram.filters.command.CommandObject author_signature (alogram.types.chat.Chat args (alogram.tips.currency.currency attribute), 461 ARS (alogram.enums.currency.Currency attribute), 465 AZN (alogram.enums.currency.Currency attribute), 461 as_handlerO (alogram.utils.formatting.Text method), 550 back() (alogram.types.chat.Chat attribute), 31 ss_list() (in module alogram.utils.formatting), 567 ban(Alogram.methods.ban_chat_sender_chat), 461 as_markdown() (alogram.utils.formatting), 567 as_marked_section() as_marked_list() (in as_numbered_list() (in as_numbered_list() (in as_numbered_list() (in as_section() (in as_numbered_list() (in as_numbered_list() (in as_numbered_list() (in as_numbered_list() (in as_numbered_list() (in as_numbered_list() (in a		
aiogram.methods.approve_chat_join_request), 308 args (aiogram.filters.command.CommandObject at- tribute), 486 args (aiogram.filters.command.CommandObject at- tribute), 486 ARTICLE (aiogram.enums.currency.Currency attribute), 461 ARTICLE (aiogram.enums.inline_query_result_type.InlineQueryResulType attribute), 456 as_handler() (aiogram.sins.scene.Scene class method), 531 as_key_value() (in module aiogram.utils.formatting), 570 as_markdown() (aiogram.utils.formatting), 567 as_marked_list() (in module aiogram.utils.formatting), 567 as_numbered_list() (in module aiogram.utils.formatting), 567 as_numbered_section() (in module aiogram.utils.formatting), 567 as_numbered_section() (in module aiogram.utils.formatting), 567 as_secterion() (in module aiogram.utils.formatting), 567 as_secterion() (in module aiogram.atils.formatting), 567 as_		
308attribute), 194args (aiogram.filters.command.CommandObject at tribute), 486attribute), 194args (aiogram.filters.command.CommandObject at tribute), 486available_reactions (aiogram.types.chat.Chat attribute), 30ARS (aiogram.enums.currency.Currency attribute), 461AZN (aiogram.enums.currency.Currency attribute), 461ARTICLE (aiogram.enums.inine_query_result_type.InlineQueryResultType attribute), 465AZN (aiogram.enums.currency.Currency attribute), 461Bas_handler() (aiogram.stms.scene.Scene class method), 531back() (aiogram.enums.currency.Currency attribute), 31as_henyler() (aiogram.utils.formatting.Text method), 568back() (aiogram.types.chat.Chat method), 41as_list() (in module aiogram.utils.formatting), 567back() (aiogram.methods.ban_chat_methods.ban_chat_method), 33as_marked_list() (in module aiogram.utils.formatting), 567back (aiogram.enums.currency.Currency attribute), 464aiogram.utils.formatting), 567back (aiogram.client.telegram.TelegramAPIServer atirbute), 12as_numbered_list() (in module aiogram.utils.formatting), 567back (aiogram.client.telegram.TelegramAPIServer atirbute), 12as_numbered_list() (in module aiogram.utils.formatting), 567back (aiogram.dispatcher.midlewers.base), 537as_numbered_list() (in module aiogram.utils		
tribute), 486attribute), 461ARS (aiogram.enums.currency: Currency attribute), 461AZN (aiogram.enums.currency.Currency attribute), 461ARTICLE (aiogram.enums.inline_query_result_type.linle@ueryResultTypeBas_handler() (aiogram.fsm.scene.Scene class method), 531back() (aiogram.fsm.scene.Scene Wizard method), 533back() (aiogram.fsm.scene.Scene visas method), 568back() (aiogram.fsm.scene.Scene Wizard method), 533back() (aiogram.fsm.scene.Scene visas method), 568back() (aiogram.fsm.scene.Scene Wizard method), 533back() (aiogram.fsm.scene.Scene visas method), 568back() (aiogram.fsm.scene.Scene Vizard method), 513as_kwargs() (aiogram.utils.formatting.Text method), 569back() (aiogram.types.chat.Chat attribute), 461ban() (aiogram.utils.formatting), 567as_list() (in module aiogram.utils.formatting), 567as_marked_list() (in module aiogram.utils.formatting), 567module aiogram.utils.formatting), 567as_numbered_list() (in module aiogram.utils.formatting), 567module aiogram.utils.formatting), 567as_numbered_list() (in module aiogram.utils.formatting), 567module aiogram.utils.formatting), 567as_numbered_section() (in module aiogram.utils.formatting), 567module aiogram.utils.formatting), 567as_router() (aiogram.fsm.scene.Scene class method), 531567as_router() (aiogram.utils.formatting), 567module aiogram.utils.formatting), 567as_router() (aiogram.utils.formatting), 567module aiogram.utils.formatting), 567as_router() (aiogram.fsm.scene.Scene class method), 531module aiogram.utils.formatting), 567 <t< td=""><td>308</td><td>attribute), 194</td></t<>	308	attribute), 194
ARTICLE (aiogram.enums.inline_query_result_type.InlineQueryResultType B attribute), 465 B as_handler() (aiogram.fsm.scene.Scene kloss method), 531 back() (aiogram.fsm.scene.SceneWizard method), 533 as_html() (aiogram.utils.formatting.Text method), 570 back() (aiogram.types.chat.Chat attribute), 461 as_list() (in module aiogram.utils.formatting.Text method), 569 B as_list() (in module aiogram.utils.formatting, 567 B as_marked_list() (in module aiogram.utils.formatting), 567 BanChatSender_chat() (aiogram.methods.ban_chat_sender_chat), 311 as_marked_list() (in module aiogram.utils.formatting), 567 BANK_STATEMENT (aiogram.enums.encrypted_passport_element.Encrypted attribute), 12 as_numbered_list() (in module aiogram.utils.formatting), 567 BaseRequestHandler (class in aiogram.dispatcher.middlewares.base), 537 as_routerO((inomodule aiogram.utils.formatting), 567 BaseRequestHandler (class in aiogram.ser.sen.sen.sen.pyped BaseSession (class in aiogram.sen.sen.sen.sen.sen.sen.sen.sen.sen.sen		
attribute), 465Bas_handler() (aiogram.fsm.scene.Scene class methol), 531back() (aiogram.fsm.scene.SceneWizard method), 533as_html() (aiogram.utils.formatting.Text method), 568back() (aiogram.types.chat.Chat attribute), (aiogram.types.chat.Chat method), 569as_line() (in module aiogram.utils.formatting.Text method), 569back() (aiogram.types.chat.Chat attribute), (aiogram.types.chat.Chat method), 41as_line() (in module aiogram.utils.formatting), 567banchatKember (class in aiogram.utils.formatting), 567as_marked_list() (in module aiogram.utils.formatting), 567module aiogram.utils.formatting), 567as_numbered_list() (in module aiogram.utils.formatting), 567module aiogram.utils.formatting), 567as_numbered_list() (in module aiogram.utils.formatting), 567module aiogram.utils.formatting), 567as_numbered_list() (in module aiogram.utils.formatting), 567module aiogram.utils.formatting), 567as_numbered_list() (in module aiogram.utils.formatting), 567module aiogram.utils.formatting), 567as_numbered_list() (in module aiogram.utils.formatting), 567module aiogram.utils.formatting), 567as_numbered_section() (in module aiogram.utils.formatting), 567module aiogram.utils.formatting), 567as_router() (aiogram.fsm.scene.Scene class method), 531assection() (in module aiogram.utils.formatting), 567as_scetion() (in module aiogram.utils.formatting), 567module aiogram.utils.formatting), 567as_scetion() (in module aiogram.utils.formatting), 567module aiogram	ARS (aiogram.enums.currency.Currency attribute), 461	AZN (aiogram.enums.currency.Currency attribute), 461
 background_custom_emoji_id class background_custom_emoji_id class background_custom_emoji_id class background_custom_emoji_id class background_custom_emoji_id class background_custom_emoji_id class <liclass< li=""> class class class clas</liclass<>		QueryResultType B
531as_html() (aiogram.utils.formatting.Text method), 570as_key_value() (in module aiogram.utils.formatting.Text method), 569as_line() (in module aiogram.utils.formatting), 566as_list() (in module aiogram.utils.formatting), 567as_marked_list() (aiogram.utils.formatting), 567as_marked_list() (in module aiogram.utils.formatting), 567as_marked_section() (in module aiogram.utils.formatting), 567as_numbered_list() (in module aiogram.utils.formatting), 567as_router() (aiogram.formatting), 567as_router() (aiogram.formatting), 567as_scettor() (in module aiogram.utils.formatting), 567as_scettor() (in module aiogram.utils.formatting), 567as_scettor() (in module aiogram.utils.formatting), 567as_scettor() (in module aiogram.utils.formatting), 567as_scettor() (aiogram.formatting), 567as_scettor() (aiogram.formatting), 567as_scettor() (aiogram.formatting), 567as_scettor() (aiogram.formatting), 567as_scettor()	<pre>as_handler() (aiogram.fsm.scene.Scene class method),</pre>	back() (aiogram.fsm.scene.SceneWizard method), 533
as_html() (aiogram.utils.formatting.Text method), 570(aiogram.types.chat.Chat attribute), 31as_kwargs() (aiogram.utils.formatting.Text method), 569569as_line() (in module aiogram.utils.formatting), 567ban() (aiogram.types.chat.Chat method), 41as_list() (in module aiogram.utils.formatting), 567ban_sender_chat() (aiogram.types.chat.Chat method), 41as_list() (in module aiogram.utils.formatting), 567ban_sender_chat() (aiogram.types.chat.Chat method), 41as_marked_nist() (in module aiogram.utils.formatting), 567ban() (aiogram.types.chat.Chat method), 41as_marked_list() (in module aiogram.utils.formatting), 567ban(hat Member (class in aiogram.methods.ban_chat_member), 309as_marked_list() (in module aiogram.utils.formatting), 567ban(hat Member (class in aiogram.methods.ban_chat_member), 309as_marked_list() (in module aiogram.utils.formatting), 567base (aiogram.enums.encrypted_passport_element.Encryptedas_numbered_list() (in module aiogram.utils.formatting), 567base (aiogram.client.telegram.TelegramAPIServeras_numbered_section() (in module aiogram.utils.formatting), 567baseMiddleware (class in aiogram.dispatcher.middlewares.base), 537as_router() (aiogram.fsm.scene.Scene class method), 531567as_section() (in module aiogram.utils.formatting), 567baseSession (class in aiogram.client.session.base), 13as_section() (in module aiogram.utils.formatting), 567baseSetorage (class in aiogram.ge.base), 516as_section() (in module aiogram.utils.formatting), 567baseSetorage (class in aiogram.ge.base), 516as_section() (in module aiogram.utils.formatting), 567baseSetorage (class in aiogram.ge.base), 516	531	
 568 ban () (aiogram.types.chat.Chat method), 41 ban_sender_chat() (aiogram.types.chat.Chat method), 569 as_line() (in module aiogram.utils.formatting), 566 as_marked_list() (in module aiogram.utils.formatting), 567 as_marked_list() (in module aiogram.utils.formatting), 567 as_marked_section() (in module aiogram.utils.formatting), 567 as_numbered_list() (in module aiogram.utils.formatting), 567 baseMiddleware (class in aiogram.client.session.base), 537 BaseRequestHandler (class in aiogram.client.session.base), 13 BaseSettion() (in module aiogram.utils.formatting), 567 BASKETBALL (aiogram.enums.dice_emoji.DiceEmoji at- 		
as_kwargs() (aiogram.utils.formatting.Text method), 569 as_line() (in module aiogram.utils.formatting), 566 as_list() (in module aiogram.utils.formatting), 567 as_marked_list() (in module aiogram.utils.formatting), 567 as_marked_section() (in module aiogram.utils.formatting), 567 as_numbered_list() (in module aiogram.utils.formatting), 567 as_numbered_section() (in module aiogram.utils.formatting), 567 as_numbered_section() (in module aiogram.utils.formatting), 567 as_numbered_section() (in module aiogram.utils.formatting), 567 as_numbered_section() (in module aiogram.utils.formatting), 567 as_router() (aiogram.fsm.scene.Scene class method), 531 as_section() (in module aiogram.utils.formatting), 567 AUD (aiogram.enums.currency.Currency attribute), 461 ban_sender_chat() (aiogram.enums.currency.Currency attribute), 461 ban_sender_chat() (aiogram.enums.currency.Currency attribute), 461 banCatSender_Chat (class in aiogram.enums.encrypted_passport_element.Encrypted attribute), 12 BaseMiddleware (class in aiogram.webhook.aiohttp_server), 499 BaseSession (class in aiogram.client.session.base), 13 BaseStorage (class in aiogram.fsm.storage.base), 516 BASKETBALL (aiogram.enums.dice_emoji.DiceEmoji at-		BAM (aiogram.enums.currency.Currency attribute), 461
 as_line() (in module aiogram.utils.formatting), 566 as_markdown() (aiogram.utils.formatting), 567 as_marked_list() (in module aiogram.utils.formatting), 567 as_marked_section() (in module aiogram.utils.formatting), 567 as_numbered_list() (in module aiogram.utils.formatting), 567 as_numbered_section() (in module aiogram.utils.formatting), 567 as_router() (aiogram.fsm.scene.Scene class method), 531 as_section() (in module aiogram.utils.formatting), 567 as_section() (in module aiogram.utils.formatting), 567 as_section() (in module aiogram.utils.formatting), 567 as_scetton() (in module aiogram.utils.formatting), 567 BaseStorage (class in aiogram.fsm.storage.base), 516 BASKETBALL (aiogram.enums.dice_emoji.DiceEmoji at- 		ban() (aiogram.types.chat.Chat method), 41
 as_line() (in module aiogram.utils.formatting), 566 as_list() (in module aiogram.utils.formatting), 567 as_marked_list() (in module aiogram.utils.formatting), 567 as_marked_section() (in module aiogram.utils.formatting), 567 as_numbered_list() (in module aiogram.utils.formatting), 567 as_numbered_list() (in module aiogram.utils.formatting), 567 as_numbered_list() (in module aiogram.utils.formatting), 567 as_numbered_section() (in module aiogram.utils.formatting), 567 as_numbered_section() (in module aiogram.utils.formatting), 567 as_numbered_section() (in module aiogram.utils.formatting), 567 as_router() (aiogram.fsm.scene.Scene class method), 531 as_section() (in module aiogram.utils.formatting), 567 Baschiddleware (class in aiogram.dispatcher.middlewares.base), 537 BaseRequestHandler (class in aiogram.client.session.base), 13 BaseSession (class in aiogram.client.session.base), 516 BASKETBALL (aiogram.enums.dure_emoji.DiceEmoji at- 		
as_list() (in module aiogram.utils.formatting), 567aiogram.methods.bachat_member), 309as_markdown() (aiogram.utils.formatting), 567aiogram.methods.bachat_sender_chat),as_marked_list()(inaiogram.utils.formatting), 567aiogram.methods.bachat_sender_chat),as_marked_section()(inaiogram.utils.formatting), 567BANK_STATEMENT (aiogram.enums.encrypted_passport_element.Encryptedas_marked_section()(inaiogram.utils.formatting), 567Base (aiogram.client.telegram.TelegramAPIServeras_numbered_list()(inaiogram.utils.formatting), 567BaseMiddlewareas_numbered_section()(inas_router()(aiogram.fsm.scene.Scene class method), 531as_scection()(in module aiogram.utils.formatting), 567AUD (aiogram.enums.currency.Currency attribute), 461BaseStorage (class in aiogram.fsm.storage.base), 516AUD (aiogram.enums.currency.Currency attribute), 461BaseKetBALL (aiogram.enums.dice_emoji.DiceEmoji at-		
as_markdown() (aiogram.utils.formatting.Text method), 570 as_marked_list() (in module aiogram.utils.formatting), 567 as_marked_section() (in module aiogram.utils.formatting), 567 as_numbered_list() (in module aiogram.utils.formatting), 567 as_numbered_section() (in module aiogram.utils.formatting), 567 as_router() (aiogram.fsm.scene.Scene class method), 531 as_section() (in module aiogram.utils.formatting), 567 BaseSession (class in aiogram.client.session.base), 13 BaseStorage (class in aiogram.fsm.storage.base), 516 BASKETBALL (aiogram.enums.currency.Currency attribute), 461		
570aiogram.util classaiogram.util classaingram.util class </td <td></td> <td></td>		
as_marked_list()(inmoduleaiogram.utils.formatting), 567311as_marked_section()(inaiogram.utils.formatting), 567BANK_STATEMENT (aiogram.enums.encrypted_passport_element.Encrypted.as_numbered_list()(inaiogram.utils.formatting), 567baseas_numbered_section()(inaiogram.utils.formatting), 567baseas_numbered_section()(inaiogram.utils.formatting), 567baseas_numbered_section()(inaiogram.utils.formatting), 567baseas_router()(aiogram.fsm.scene.Scene class method), 531as_section()(in module aiogram.utils.formatting), 567as_section()(in module aiogram.utils.formatting), 567as_section()(in module aiogram.utils.formatting), 567as_section()(in module aiogram.utils.formatting), 567as_section()(in module aiogram.utils.formatting), 567baseSection()(in module aiogram.utils.formatting), 567as_section()(in module aiogram.utils.formatting), 567baseSection()(in module aiogram.utils.formatting), 567<		
aiogram.utils.formatting), 567 as_marked_section() (in module aiogram.utils.formatting), 567 as_numbered_list() (in module aiogram.utils.formatting), 567 as_numbered_section() (in module aiogram.utils.formatting), 567 as_router() (aiogram.fsm.scene.Scene class method), 531 as_section() (in module aiogram.utils.formatting), 567 as_section() (in module aiogram.utils.formatting), 567 AUD (aiogram.enums.currency.Currency attribute), 461 BANK_STATEMENT (aiogram.enums.encrypted_passport_element.Encrypted. attribute), 464 base (aiogram.client.telegram.TelegramAPIServer attribute), 12 BaseMiddleware (class in aiogram.dispatcher.middlewares.base), 537 BaseRequestHandler (class in aiogram.webhook.aiohttp_server), 499 BaseStorage (class in aiogram.fsm.storage.base), 516 BASKETBALL (aiogram.enums.dice_emoji.DiceEmoji at-		
as_marked_section()(inmoduleaiogram.utils.formatting), 567aiogram.utils.formatting), 567base(aiogram.client.telegram.TelegramAPIServeras_numbered_list()(inmoduleattribute), 464aiogram.utils.formatting), 567base(aiogram.client.telegram.TelegramAPIServeras_numbered_section()(inmoduleaiogram.utils.formatting), 567BaseMiddleware(classas_router()(aiogram.fsm.scene.Scene class method), 531BaseSession (class in aiogram.client.session.base), 13as_section()(in module aiogram.utils.formatting), 567BaseStorage (class in aiogram.client.session.base), 13AUD (aiogram.enums.currency.Currency attribute), 461BASKETBALL (aiogram.enums.dice_emoji.DiceEmoji at-		
aiogram.utils.formatting), 567base(aiogram.client.telegram.TelegramAPIServer attribute), 12as_numbered_list()(inmodule aiogram.utils.formatting), 567base(classinas_numbered_section()(inmodule aiogram.utils.formatting), 567baseMiddleware(classinas_router()(aiogram.fsm.scene.Sceneclassmodule aiogram.utils.formatting), 567baseRequestHandler(classinas_section()(in module aiogram.utils.formatting), 567baseRequestHandler(classinas_section()(in module aiogram.utils.formatting), 567baseSession (class in aiogram.client.session.base), 13baseStorage (class in aiogram.client.session.base), 13BaseStorage (class in aiogram.fsm.storage.base), 516baseKETBALL (aiogram.enums.dice_emoji.DiceEmoji at-		
as_numbered_list()(inmoduleattribute), 12aiogram.utils.formatting), 567BaseMiddleware(classinas_numbered_section()(inmoduleaiogram.dispatcher.middlewares.base), 537BaseRequestHandler(classinaiogram.utils.formatting), 567BaseRequestHandler(classinas_router()(aiogram.fsm.scene.Sceneclassmethod),531as_section()(in module aiogram.utils.formatting), 567BaseSession (class in aiogram.client.session.base), 13BaseStorage (class in aiogram.storage.base), 516BASKETBALL (aiogram.enums.dice_emoji.DiceEmoji at-	aiogram.utils.formatting), 567	
as_numbered_section() (in module aiogram.utils.formatting), 567 as_router() (aiogram.fsm.scene.Scene class method), 531 as_section() (in module aiogram.utils.formatting), 567 AUD (aiogram.enums.currency.Currency attribute), 461 aiogram.dispatcher.middlewares.base), 537 BaseRequestHandler (class in aiogram.dispatcher.middlewares.base), 537 BaseSession (class in aiogram.client.session.base), 13 BaseStorage (class in aiogram.fsm.storage.base), 516 BASKETBALL (aiogram.enums.dice_emoji.DiceEmoji at-	as_numbered_list() (in module	
aiogram.utils.formatting), 567 as_router() (aiogram.fsm.scene.Scene class method), 531 as_section() (in module aiogram.utils.formatting), 567 AUD (aiogram.enums.currency.Currency attribute), 461 BaseStorage (class in aiogram.fsm.storage.base), 516 BASKETBALL (aiogram.enums.dice_emoji.DiceEmoji at-		BaseMiddleware (class in
as_router() (aiogram.fsm.scene.Scene class method), 531 as_section() (in module aiogram.utils.formatting), 567 AUD (aiogram.enums.currency.Currency attribute), 461 AUD (aiogram.enums.currency.Currency attribute), 461		aiogram.dispatcher.middlewares.base), 537
531 as_section() (in module aiogram.utils.formatting), 567 AUD (aiogram.enums.currency.Currency attribute), 461 BaseStorage (class in aiogram.client.session.base), 13 BaseStorage (class in aiogram.fsm.storage.base), 516 BASKETBALL (aiogram.enums.dice_emoji.DiceEmoji at-		
as_section() (in module aiogram.utils.formatting), 567 AUD (aiogram.enums.currency.Currency attribute), 461 BaseStorage (class in aiogram.fsm.storage.base), 516 BASKETBALL (aiogram.enums.dice_emoji.DiceEmoji at-		· · ·
AUD (aiogram.enums.currency.Currency attribute), 461 BASKETBALL (aiogram.enums.dice_emoji.DiceEmoji at-		
	AUDIO (alogram.enums.currency.currency attribute), 461 AUDIO (alogram.enums.content_type.ContentType at-	
tribute), 459 BASKETBALL (aiogram.types.dice.DiceEmoji attribute), AUDIO (aiogram.enums.inline_query_result_type.InlineQueryResultType5		eryResultType5

BDT (<i>aiogram.enums.currency.Currency attribute</i>), 461 BGN (<i>aiogram.enums.currency.Currency attribute</i>), 461	<i>aiogram.types.bot_command_scope_all_private_chats</i>), 21
<pre>big_file_id (aiogram.types.chat_photo.ChatPhoto at-</pre>	
tribute), 113	aiogram.types.bot_command_scope_chat),
big_file_unique_id (aiogram.types.chat_photo.ChatPho	
	BotCommandScopeChatAdministrators (class in
<pre>bio (aiogram.types.chat.Chat attribute), 31 bio (aiogram.types.chat_join_request.ChatJoinRequest</pre>	aiogram.types.bot_command_scope_chat_administrators), 22
	BotCommandScopeChatMember (class in
birthdate (aiogram.types.chat.Chat attribute), 30	aiogram.types.bot_command_scope_chat_member),
Birthdate (<i>class in aiogram.types.birthdate</i>), 19	23
BLOCKQUOTE (aiogram.enums.message_entity_type.Message	e Bort Roffmand Scope Default (class in
attribute), 467	aiogram.types.bot_command_scope_default),
BLUE (aiogram.enums.topic_icon_color.TopicIconColor	23
	BotCommandScopeType (class in
BND (<i>aiogram.enums.currency.Currency attribute</i>), 461	aiogram.enums.bot_command_scope_type), 457
BOB (aiogram.enums.currency.Currency attribute), 461 BOLD (aiogram.enums.message_entity_type.MessageEntityT	
attribute), 467	aiogram.types.bot_description), 24
	BotName (<i>class in aiogram.types.bot_name</i>), 24
boost (aiogram.types.chat_boost_updated.ChatBoostUpda	
attribute), 49	aiogram.types.bot_short_description), 24
BOOST_ADDED (<i>aiogram.enums.content_type.ContentType</i> <i>attribute</i>), 460	BOWLING (aiogram.enums.dice_emoji.DiceEmoji at- tribute), 464
boost_added (aiogram.types.message.Message at-	BOWLING (aiogram.types.dice.DiceEmoji attribute), 115
	BRL (aiogram.enums.currency.Currency attribute), 461
boost_count (aiogram.types.chat_boost_added.ChatBoost	
attribute), 46 boost_id (aiogram.types.chat_boost.ChatBoost at-	127, 474
tribute), 45	method), 516
boost_id(aiogram.types.chat_boost_removed.ChatBoostR	
attribute), 46	<i>method</i>), 516
<pre>boosts(aiogram.types.user_chat_boosts.UserChatBoosts</pre>	<pre>build() (aiogram.utils.media_group.MediaGroupBuilder</pre>
attribute), 209	<i>method</i>), 576
-	BUSINESS_CONNECTION
	yboardBut(airBequestGlust.update_type.UpdateType
attribute), 137 BOT_COMMAND (aiogram.enums.message_entity_type.Messa	attribute), 470
attribute), 467	attribute), 284
<pre>bot_is_member(aiogram.types.keyboard_button_request_</pre>	
attribute), 137	$(aiogram.methods.get_business_connection.GetBusinessConnection.g$
bot_username (aiogram.types.login_url.LoginUrl	attribute), 334
attribute), 141	business_connection_id
BotCommand (class in aiogram.types.bot_command), 19 BotCommand (class in aiogram.utils.formatting), 570	(aiogram.methods.send_animation.SendAnimation attribute), 361
	business_connection_id
aiogram.types.bot_command_scope), 20	(aiogram.methods.send_audio.SendAudio
BotCommandScopeAllChatAdministrators (class in	attribute), 364
aiogram.types.bot_command_scope_all_chat_adr	
20	(aiogram.methods.send_chat_action.SendChatAction
BotCommandScopeAllGroupChats (class in	attribute), 367
aiogram.types.bot_command_scope_all_group_ci 21	
BotCommandScopeAllPrivateChats (class in	(aiogram.methods.send_contact.SendContact attribute), 368
becommunitatopentititivateenats (trass in	<i>unione</i>), 500

business_connection_id attribute), 30 (aiogram.methods.send_dice.SendDice BusinessConnection (class in attribute), 370 aiogram.types.business connection), 25 BusinessIntro (class in aiogram.types.business_intro), business_connection_id (aiogram.methods.send document.SendDocument 25 attribute), 373 BusinessLocation (class in business connection id aiogram.types.business location), 26 (aiogram.methods.send_game.SendGame BusinessMessagesDeleted (class in attribute), 439 aiogram.types.business messages deleted), 26 business_connection_id BusinessOpeningHours (class in (aiogram.methods.send_location.SendLocation aiogram.types.business_opening_hours), 27 attribute), 375 BusinessOpeningHoursInterval (class business_connection_id in (aiogram.methods.send_media_group.SendMediaGroup) aiogram.types.business_opening_hours_interval), attribute), 378 27 business_connection_id button (aiogram.methods.answer_inline_query.AnswerInlineQuery (aiogram.methods.send_message.SendMessage attribute), 435 attribute), 380 button_text(aiogram.types.web_app_data.WebAppData business_connection_id attribute), 214 (aiogram.methods.send photo.SendPhoto buttons (aiogram.utils.keyboard.InlineKeyboardBuilder property), 550 attribute), 382 business_connection_id buttons (aiogram.utils.keyboard.ReplyKeyboardBuilder (aiogram.methods.send_poll.SendPoll property), 551 attribute), 385 BYN (aiogram.enums.currency.Currency attribute), 461 business_connection_id С (aiogram.methods.send sticker.SendSticker attribute), 296 cache_time(aiogram.methods.answer callback query.AnswerCallbackQu business_connection_id attribute), 307 (aiogram.methods.send_venue.SendVenue cache_time(aiogram.methods.answer_inline_query.AnswerInlineQuery attribute), 387 attribute), 434 business_connection_id cache_time(aiogram.utils.callback_answer.CallbackAnswer (aiogram.methods.send_video.SendVideo property), 566 attribute), 390 CAD (aiogram.enums.currency.Currency attribute), 461 business_connection_id callback_data(aiogram.handlers.callback_query.CallbackQueryHandle (aiogram.methods.send video note.SendVideoNote property), 544 attribute), 393 callback_data (aiogram.types.inline_keyboard_button.InlineKeyboardBu business_connection_id attribute), 126 (aiogram.methods.send_voice.SendVoice callback_game (aiogram.types.inline_keyboard_button.InlineKeyboardBu attribute), 396 attribute), 126 business_connection_id CALLBACK_QUERY (*aiogram.enums.update_type.UpdateType* (aiogram.types.business_messages_deleted.BusinessMessagesfpalated, 470 attribute), 26 (aiogram.types.update.Update callback_query atbusiness_connection_id tribute), 285 (aiogram.types.message.Message attribute), callback_query_id(aiogram.methods.answer callback query.AnswerCallback_q 145 attribute). 307 business_intro (*aiogram.types.chat.Chat attribute*), callback_query_without_state 30 (aiogram.fsm.scene.SceneConfig attribute), business_location (aiogram.types.chat.Chat at-532 tribute), 30 (class CallbackAnswer in BUSINESS_MESSAGE (aiogram.enums.update_type.UpdateType aiogram.utils.callback answer), 565 attribute), 470 CallbackAnswerException, 540 business_message (aiogram.types.update.Update at-CallbackAnswerMiddleware (class in tribute), 284 aiogram.utils.callback_answer), 565 business_opening_hours (aiogram.types.chat.Chat

CallbackData (class in aiogram.filters.callback_data),	attribute), 89
492	<pre>can_invite_users(aiogram.methods.promote_chat_member.PromoteChat_</pre>
CallbackGame (class in aiogram.types.callback_game),	attribute), 355
287	<pre>can_invite_users(aiogram.types.chat_administrator_rights.ChatAdmin</pre>
CallbackQuery (class in	attribute), 44
aiogram.types.callback_query), 28	<pre>can_invite_users(aiogram.types.chat_member_administrator.ChatMem</pre>
CallbackQueryHandler (class in	attribute), 89
aiogram.handlers.callback_query), 544	<pre>can_invite_users(aiogram.types.chat_member_restricted.ChatMember]</pre>
can_add_web_page_previews	attribute), 93
	nbanRastvärtæl_users(aiogram.types.chat_permissions.ChatPermissions
attribute), 93	attribute), 113
can_add_web_page_previews	<pre>can_join_groups (aiogram.types.user.User attribute),</pre>
$(aiogram.types.chat_permissions.ChatPermissions)$	
attribute), 113	<pre>can_manage_chat (aiogram.methods.promote_chat_member.PromoteChat</pre>
<pre>can_be_edited(aiogram.types.chat_member_administrat</pre>	
attribute), 89	can_manage_chat (aiogram.types.chat_administrator_rights.ChatAdminis
can_change_info(aiogram.methods.promote_chat_memb	
attribute), 354	<pre>can_manage_chat (aiogram.types.chat_member_administrator.ChatMemb</pre>
can_change_info(aiogram.types.chat_administrator_rig	
attribute), 44	<pre>can_manage_topics (aiogram.methods.promote_chat_member.PromoteCha</pre>
can_change_info(aiogram.types.chat_member_administ	
attribute), 89	can_manage_topics (aiogram.types.chat_administrator_rights.ChatAdmin
can_change_info(aiogram.types.chat_member_restricted	
attribute), 93	can_manage_topics (aiogram.types.chat_member_administrator.ChatMen
can_change_info(aiogram.types.chat_permissions.Chath	
attribute), 113	<pre>can_manage_topics(aiogram.types.chat_member_restricted.ChatMember_</pre>
can_connect_to_business (aiogram.types.user.User	attribute), 93
attribute), 208	can_manage_topics (aiogram.types.chat_permissions.ChatPermissions
can_delete_messages	attribute), 113
(aiogram.methods.promote_chat_member.Promote_	-
attribute), 354	(aiogram.methods.promote_chat_member.PromoteChatMember
can_delete_messages	attribute), 354
(aiogram.types.chat_administrator_rights.ChatAd	
attribute), 44	(aiogram.types.chat_administrator_rights.ChatAdministratorRigh
can_delete_messages	attribute), 44
(aiogram.types.chat_member_administrator.Chat	
attribute), 89	(aiogram.types.chat_member_administrator.ChatMemberAdminis
can_delete_stories(aiogram.methods.promote_chat_m	
attribute), 355	can_pin_messages (aiogram.methods.promote_chat_member.PromoteCha
can_delete_stories(aiogram.types.chat_administrator_	- 0
attribute), 45	can_pin_messages (aiogram.types.chat_administrator_rights.ChatAdmini
can_delete_stories(aiogram.types.chat_member_admu	
attribute), 89 can_edit_messages(aiogram.methods.promote_chat_me	can_pin_messages (aiogram.types.chat_member_administrator.ChatMem
attribute), 355	can_pin_messages (aiogram.types.chat_member_restricted.ChatMember]
<pre>can_edit_messages(aiogram.types.chat_administrator_n</pre>	
attribute), 45 can_edit_messages (<i>aiogram.types.chat_member_admin</i>	can_pin_messages (aiogram.types.chat_permissions.ChatPermissions
attribute), 89	can_post_messages (aiogram.methods.promote_chat_member.PromoteCl
<pre>can_edit_stories(aiogram.methods.promote_chat_men</pre>	can_post_messages(aiogram.types.chat_administrator_rights.ChatAdmin
can_edit_stories(aiogram.types.chat_administrator_ri	
attribute), 45	
can_edit_stories(<i>aiogram.types.chat_member_adminis</i>	can_post_messages (aiogram.types.chat_member_administrator.ChatMen
can_eurc_scorres(alogram.types.cnat_member_daminis	

	emPsendte60ikitsManberam.types.chat_permissions.ChatPermissions
attribute), 355	attribute), 113
<pre>can_post_stories(aiogram.types.chat_administrator_right)</pre>	
attribute), 45	$(aiogram.types.chat_member_restricted.ChatMemberRestricted$
<pre>can_post_stories(aiogram.types.chat_member_administr</pre>	
	an_send_video_notes
can_promote_members	(aiogram.types.chat_permissions.ChatPermissions
(aiogram.methods.promote_chat_member.Promote)	
	an_send_videos(aiogram.types.chat_member_restricted.ChatMemberR
<pre>can_promote_members</pre>	attribute), 92
	ainissendr_Rigdeos (aiogram.types.chat_permissions.ChatPermissions
attribute), 44	attribute), 112
-	an_send_voice_notes
attribute), 89	emberAd mingstratoy pes.chat_member_restricted.ChatMemberRestricted attribute), 93
	an_send_voice_notes
(aiogram.types.user.User attribute), 208	(aiogram.types.chat_permissions.ChatPermissions
<pre>can_reply (aiogram.types.business_connection.BusinessCon</pre>	
	an_set_sticker_set (aiogram.types.chat.Chat
can_restrict_members	attribute), 32
attribute), 354	alpttMorn(beogram.methods.copy_message.CopyMessage attribute), 316
can_restrict_members c	aption (aiogram.methods.edit_message_caption.EditMessageCaption
(aiogram.types.chat_administrator_rights.ChatAdm	
	aption (aiogram.methods.send_animation.SendAnimation
<pre>can_restrict_members</pre>	attribute), 362
	apibioaAdminiogramo.methods.send_audio.SendAudio at-
attribute), 89	tribute), 364
<pre>can_send_after (aiogram.utils.web_app.WebAppInitDatac</pre>	
attribute), 561	attribute), 373
can_send_audios (aiogram.types.chat_member_restricted attribute), 92	tribute), 382
can_send_audios (<i>aiogram.types.chat_permissions.ChatPe</i> <i>attribute</i>), 112	aptsions (aiogram.methods.send_video.SendVideo attribute), 391
<pre>can_send_documents(aiogram.types.chat_member_restric</pre>	apl:CloutMembe(diesgrintednethods.send_voice.SendVoice
attribute), 92	attribute), 396
<pre>can_send_documents(aiogram.types.chat_permissions.Cha attribute), 112</pre>	ជម្នាប់ចាំស៊េដែលក្ល ram.types.inline_query_result_audio.InlineQueryResultAudi attribute), 221
	dpflictMentburch, 221 dpflictMentburchestypieteihline_query_result_cached_audio.InlineQueryRe
attribute), 92	attribute), 223
attribute), 112	Rphinixmi(ano gram.types.inline_query_result_cached_document.InlineQuer attribute), 225
	aption (aiogram.types.inline_query_result_cached_gif.InlineQueryResult_
(aiogram.types.chat_member_restricted.ChatMemb	
	aption(aiogram.types.inline_query_result_cached_mpeg4_gif.InlineQue
can_send_other_messages	attribute), 229
	aption(aiogram.types.inline_query_result_cached_photo.InlineQueryRe
attribute), 113	attribute), 231
can_send_photos (<i>aiogram.types.chat_member_restricted</i> <i>attribute</i>), 92	ApttMan(biorgramitypeds.inline_query_result_cached_video.InlineQueryRes attribute), 235
can_send_photos (<i>aiogram.types.chat_permissions.ChatPe</i> <i>attribute</i>), 112	aptsion(aiogram.types.inline_query_result_cached_voice.InlineQueryRes attribute), 237
	appiMankaiRestarictydes.inline_query_result_document.InlineQueryResult
attribute), 93	attribute), 241

caption (aiogram.types.inline_query_result_gif.InlineQue		
attribute), 244 caption(aiogram.types.inline_query_result_mpeg4_gif.Ir		_entities(aiogram.types.inline_query_result_gif.InlineQueryRe RatrihMev=444if
attribute), 248		_entities(aiogram.types.inline_query_result_mpeg4_gif.Inline(
caption (aiogram.types.inline_query_result_photo.Inline		
attribute), 250		_entities(aiogram.types.inline_query_result_photo.InlineQuery
<pre>caption(aiogram.types.inline_query_result_video.InlineQ</pre>		
attribute), 254		_entities (aiogram.types.inline_query_result_video.InlineQuery
caption (aiogram.types.inline_query_result_voice.Inline)		
attribute), 255 caption (aiogram.types.input_media_animation.InputMed		_entities (aiogram.types.inline_query_result_voice.InlineQuery
attribute), 129		_entities(aiogram.types.input_media_animation.InputMediaAr
caption (aiogram.types.input_media_audio.InputMediaA		attribute), 129
attribute), 130		_entities (aiogram.types.input_media_audio.InputMediaAudio
caption (aiogram.types.input_media_document.InputMed		
attribute), 131		_entities(aiogram.types.input_media_document.InputMediaDo
<pre>caption(aiogram.types.input_media_photo.InputMediaPl</pre>		attribute), 131
attribute), 132		_entities (aiogram.types.input_media_photo.InputMediaPhoto
caption(aiogram.types.input_media_video.InputMediaVi		attribute), 132
attribute), 133	-	_entities (aiogram.types.input_media_video.InputMediaVideo
caption (aiogram.types.message.Message attribute), 147		attribute), 133
<pre>caption_entities(aiogram.methods.copy_message.Cop attribute), 316</pre>		attribute), 147
caption_entities (<i>aiogram.methods.edit_message_cap</i>		
attribute), 423		attribute), 467
caption_entities (aiogram.methods.send_animation.Se	n clashiTnaq i	
attribute), 362		cate (aiogram.methods.set_webhook.SetWebhook
caption_entities(aiogram.methods.send_audio.SendA		attribute), 454
attribute), 364		(aiogram.enums.chat_type.ChatType attribute),
caption_entities (aiogram.methods.send_document.Se		
attribute), 373		(aiogram.enums.message_origin_type.MessageOriginType
caption_entities (<i>aiogram.methods.send_photo.SendP</i>		attribute), 468
attribute), 382		_CHAT_CREATED
caption_entities (aiogram.methods.send_video.SendV attribute), 391		(aiogram.enums.content_type.ContentType attribute), 460
caption_entities(<i>aiogram.methods.send_voice.SendVo</i>		
attribute), 396	/ chainer.	(aiogram.types.message.Message attribute),
caption_entities (aiogram.types.inline_query_result_c	udio.Inline	
attribute), 221		POST (aiogram.enums.update_type.UpdateType
caption_entities(aiogram.types.inline_query_result_c		
attribute), 223	channel	_post (aiogram.types.update.Update attribute),
caption_entities (<i>aiogram.types.inline_query_result_c</i>		
attribute), 225		gram.enums.bot_command_scope_type.BotCommandScopeType
caption_entities (<i>aiogram.types.inline_query_result_c</i>		
attribute), 227		gram.enums.message_origin_type.MessageOriginType
<pre>caption_entities(aiogram.types.inline_query_result_c attribute), 229</pre>	-	egen_types.business_messages_deleted.BusinessMessagesDeleted
caption_entities (<i>aiogram.types.inline_query_result_c</i>		
attribute), 231	-	gram.types.chat_boost_removed.ChatBoostRemoved
caption_entities (<i>aiogram.types.inline_query_result_c</i>		· · ·
attribute), 235		gram.types.chat_boost_updated.ChatBoostUpdated
caption_entities(<i>aiogram.types.inline_query_result_c</i>		
attribute), 237		ogram.types.chat_join_request.ChatJoinRequest
caption entities(<i>aiogram.types.inline query result a</i>	locument h	nditmeiDuten v ResultDocument

chat (aiogram.types.chat_member_updated.ChatMemberU attribute), 93	Up ahated id(aiogram.methods.delete_message.DeleteMessage attribute), 421
chat (aiogram.types.external_reply_info.ExternalReplyInfo attribute), 117	ochat_id(aiogram.methods.delete_messages.DeleteMessages attribute), 422
chat (aiogram.types.giveaway_winners.GiveawayWinners attribute), 124	<pre>chat_id(aiogram.methods.edit_chat_invite_link.EditChatInviteLink</pre>
chat (aiogram.types.inaccessible_message.InaccessibleMe attribute), 125	esshget_id(aiogram.methods.edit_forum_topic.EditForumTopic attribute), 328
<pre>chat (aiogram.types.message.Message attribute), 145 chat (aiogram.types.message_origin_channel.MessageOri</pre>	chat_id(aiogram.methods.edit_general_forum_topic.EditGeneralForumTeginChannehttribute), 329
<pre>attribute), 193 chat (aiogram.types.message_reaction_count_updated.Me</pre>	chat_id (aiogram.methods.edit_message_caption.EditMessageCaption essageReact ion(Initn) LPDPated
attribute), 196	chat_id(aiogram.methods.edit_message_live_location.EditMessageLiveLocation.editMes
chat (aiogram.types.message_reaction_updated.MessageR	
attribute), 196 chat (aiogram.types.story.Story attribute), 205	<pre>chat_id(aiogram.methods.edit_message_media.EditMessageMedia</pre>
chat (alogram.types.slory.slory attribute), 205 chat (alogram.utils.web_app.WebAppInitData attribute), 560	
Chat (<i>class in aiogram.types.chat</i>), 29 CHAT_ADMINISTRATORS	chat_id (aiogram.methods.edit_message_text.EditMessageText attribute), 430
	CochatndScapeType.methods.export_chat_invite_link.ExportChatInviteLink attribute), 330
CHAT_BOOST (<i>aiogram.enums.update_type.UpdateType attribute</i>), 471	<pre>chat_id(aiogram.methods.forward_message.ForwardMessage</pre>
	chat_id (aiogram.methods.forward_messages.ForwardMessages attribute), 333
<pre>chat_has_username(aiogram.types.keyboard_button_red attribute), 136</pre>	
	pcbacClicd.lainBeamestethods.get_chat_administrators.GetChatAdministrator attribute), 336
	ember_id(aiogram.methods.get_chat_member.GetChatMember attribute), 337
	atStatleid((atogram.methods.get_chat_member_count.GetChatMemberCoun attribute), 338
	n Eppaic_id (aiogram.methods.get_chat_menu_button.GetChatMenuButton attribute), 339
<pre>chat_id(aiogram.methods.close_general_forum_topic.Clo attribute), 314</pre>	o sdGatneird/Eiogun/Fioprie thods.get_game_high_scores.GetGameHighScores attribute), 438
<pre>chat_id (aiogram.methods.copy_message.CopyMessage</pre>	<pre>chat_id(aiogram.methods.get_user_chat_boosts.GetUserChatBoosts</pre>
<pre>chat_id(aiogram.methods.copy_messages.CopyMessages</pre>	s chat_id(aiogram.methods.hide_general_forum_topic.HideGeneralForum' attribute), 349
<pre>chat_id(aiogram.methods.create_chat_invite_link.Create</pre>	e Gh#f nyiteLink (aiogram.methods.leave_chat.LeaveChat attribute), 350
	un Hapiad (aiogram.methods.pin_chat_message.PinChatMessage attribute), 352
	cl ahaCh_citd.(iniRgquuest methods.promote_chat_member.PromoteChatMember attribute), 354
	Phat_id (aiogram.methods.reopen_forum_topic.ReopenForumTopic attribute), 356
	Consticka (Seogram.methods.reopen_general_forum_topic.ReopenGeneralFo attribute), 357
	un Mapicid (aiogram.methods.restrict_chat_member.RestrictChatMember attribute), 358

chat_id	(aiogram.methods.revoke_chat_invite_link.Revoke attribute), 359		hing ram.methods.set_message_reaction.SetMessageReaction attribute), 405
chat_id		chat_id(aiogram.methods.stop_message_live_location.StopMessageLiveLattribute), 431
chat_id	(aiogram.methods.send_audio.SendAudio at- tribute), 364		aiogram.methods.stop_poll.StopPoll attribute), 432
chat_id	(aiogram.methods.send_chat_action.SendChatAct attribute), 366		aiogram.methods.unban_chat_member.UnbanChatMember attribute), 412
chat_id	(aiogram.methods.send_contact.SendContact attribute), 368		aiogram.methods.unban_chat_sender_chat.UnbanChatSenderCh attribute), 414
chat_id	(aiogram.methods.send_dice.SendDice at- tribute), 370		aiogram.methods.unhide_general_forum_topic.UnhideGeneralFo attribute), 415
chat_id	(aiogram.methods.send_document.SendDocument attribute), 373		aiogram.methods.unpin_all_chat_messages.UnpinAllChatMessag attribute), 416
chat_id	(aiogram.methods.send_game.SendGame attribute), 439		aiogram.methods.unpin_all_forum_topic_messages.UnpinAllFor attribute), 417
chat_id	(aiogram.methods.send_invoice.SendInvoice at- tribute), 447		aiogram.methods.unpin_all_general_forum_topic_messages.Unp attribute), 418
chat_id	(<i>aiogram.methods.send_location.SendLocation attribute</i>), 375		aiogram.methods.unpin_chat_message.UnpinChatMessage attribute), 419
chat_id	(aiogram.methods.send_media_group.SendMedia attribute), 377	-	aiogram.types.bot_command_scope_chat.BotCommandScopeCha attribute), 22
chat_id	. (aiogram.methods.send_message.SendMessage attribute), 380		aiogram.types.bot_command_scope_chat_administrators.BotCon attribute), 22
chat_id	(aiogram.methods.send_photo.SendPhoto at- tribute), 382		aiogram.types.bot_command_scope_chat_member.BotCommand.attribute), 23
chat_id	. (aiogram.methods.send_poll.SendPoll at- tribute), 384		(aiogram.types.chat_shared.ChatShared at- tribute), 114
chat_id	(aiogram.methods.send_sticker.SendSticker at- tribute), 296		aiogram.types.reply_parameters.ReplyParameters attribute), 203
chat_id	(<i>aiogram.methods.send_venue.SendVenue at-</i> <i>tribute</i>), 387		tance (aiogram.types.callback_query.CallbackQuery attribute), 28
chat_id	(aiogram.methods.send_video.SendVideo attribute), 390		tance (aiogram.utils.web_app.WebAppInitData attribute), 560
chat_id	(aiogram.methods.send_video_note.SendVideoNot attribute), 393		channel (aiogram.types.keyboard_button_request_chat.Keyboar attribute), 136
chat_id	(aiogram.methods.send_voice.SendVoice attribute), 395		created (aiogram.types.keyboard_button_request_chat.Keyboar attribute), 136
chat_id	(aiogram.methods.set_chat_administrator_custom attribute), 398		H ærfulm(imiløgraton:Gpret o heFjblæ ard_button_request_chat.KeyboardB attribute), 136
chat_id	(aiogram.methods.set_chat_description.SetChatDe attribute), 399	<u> </u>	N_REQUEST (aiogram.enums.update_type.UpdateType attribute), 471
chat_id	(aiogram.methods.set_chat_menu_button.SetChat_attribute), 400		n_ request (aiogram.types.update.Update at- tribute), 285
chat_id	(aiogram.methods.set_chat_permissions.SetChatP attribute), 401		BER (aiogram.enums.bot_command_scope_type.BotCommandSco attribute), 457
chat_id	(aiogram.methods.set_chat_photo.SetChatPhoto attribute), 402		BER (<i>aiogram.enums.update_type.UpdateType attribute</i>), 471
chat_id	(aiogram.methods.set_chat_sticker_set.SetChatSticattribute), 403		ber (aiogram.types.update.Update attribute), 285
chat_id		CHAT_SHA	
chat_id	(aiogram.methods.set_game_score.SetGameScore attribute), 441	chat_sha	

<pre>chat_type (aiogram.types.inline_query.InlineQuery at- tribute), 216</pre>
chat_type (aiogram.utils.web_app.WebAppInitData at- tribute), 560
ChatAction (<i>class in aiogram.enums.chat_action</i>), 457
ChatActionMiddleware (class in
aiogram.utils.chat_action), 558
ChatActionSender (<i>class in aiogram.utils.chat_action</i>), 556
ChatAdministratorRights (class in
<i>aiogram.types.chat_administrator_rights</i>), 43
ChatBoost (class in aiogram.types.chat_boost), 45
ChatBoostAdded (class in
aiogram.types.chat_boost_added), 46
ChatBoostRemoved (class in
aiogram.types.chat_boost_removed), 46
ChatBoostSource (class in
aiogram.types.chat_boost_source), 47
ChatBoostSourceGiftCode (class in
aiogram.types.chat_boost_source_gift_code),
47
ChatBoostSourceGiveaway (class in
aiogram.types.chat_boost_source_giveaway),
47
ChatBoostSourcePremium (class in
aiogram.types.chat_boost_source_premium), 48
ChatBoostSourceType (class in
aiogram.enums.chat_boost_source_type), 458
ChatBoostUpdated (class in
aiogram.types.chat_boost_updated), 49
ChatInviteLink (class in
aiogram.types.chat_invite_link), 49
ChatJoinRequest (class in
aiogram.types.chat_join_request), 50
ChatLocation (class in aiogram.types.chat_location),
87
ChatMember (<i>class in aiogram.types.chat_member</i>), 87
ChatMemberAdministrator (class in
aiogram.types.chat_member_administrator), 88
ChatMemberBanned (class in
aiogram.types.chat_member_banned), 90
ChatMemberLeft (class in
aiogram.types.chat_member_left), 90
ChatMemberMember (class in
aiogram.types.chat_member_member), 91
ChatMemberOwner (class in
aiogram.types.chat_member_owner), 91
ChatMemberRestricted (class in
aiogram, types, chat member restricted).
aiogram.types.chat_member_restricted), 92

ChatMemberStatus	(class	in
aiogram.enums.chat		458
ChatMemberUpdated	(class	in
aiogram.types.chat_	member_updated),	93
ChatMemberUpdatedFilter	(class	in
aiogram.filters.chat_	_member_updated),	, 487
ChatPermissions	(class	in
aiogram.types.chat_	permissions), 112	
ChatPhoto (class in aiogram.	.types.chat_photo),	113
chats (aiogram.types.giveawa	ay.Giveaway attribi	ıte), 122
ChatShared (class in aiogram	n.types.chat_shared	l), 114
ChatType (class in aiogram.e	nums.chat_type), 4	59
<pre>check_flags() (in module</pre>	aiogram.dispatche	er.flags),
542		
<pre>check_response() (aiogram</pre>	n.client.session.base	e.BaseSession
<i>method</i>), 13		
<pre>check_webapp_signature(</pre>) (in	module
aiogram.utils.web_a	upp), 559	
CHF (aiogram.enums.currency	Currency attribute), 461
CHIN (aiogram.enums.mask_p	osition_point.Mask	PositionPoint
attribute), 466		
CHOOSE_STICKER (aiogram.en	nums.chat_action.C	ChatAction
attribute), 458		
<pre>choose_sticker() (aiogram</pre>	n.utils.chat_action.C	ChatActionSender
class method), 556		
CHOSEN_INLINE_RESULT		
(aiogram.enums.upa	late_type.UpdateTy	pe
attribute), 470		
chosen_inline_result (ai	ogram.types.update	e.Update
attribute), 285		
ChosenInlineResult	(class	in
aiogram.types.chose		
city (aiogram.types.shipping	g_address.Shipping	Address
attribute), 281	с с	**** 1
	ram.fsm.scene.Scen	eWizard
method), 533		
ClientDecodeError, 541		
Close (<i>class in aiogram.meth</i>		c ·
	nt.session.base.Base	eSession
method), 13	Som og Man agon	weth od)
close() (aiogram.fsm.scene	e.scenesmanager n	netnoa),
532	w atowaco haao Daa	Ctongoo
close() (aiogram.fsr method), 517	n.storage.base.Base	esiorage
close() (aiogram.webhook.a	iohttn server Simn	leReauestHandler
method), 500	toniip_server.simpt	enequesinanuer
close_date (aiogram.metho	ods send noll Sendi	Poll at-
tribute), 385	Jus.senu_pon.senu	
close_date (<i>aiogram.types.p</i>	oll Poll attribute)	198
CloseForumTopic	(class	in
aiogram.methods.cl		
CloseGeneralForumTopic	(class	in
aiogram.methods.cl		
314		- • * ′

closing	_minute(<i>aiogram</i> . <i>attribute</i>), 27	types.business_openi	ng_hours		BusinessOpsecuingHu invite_link()	o y.:Cinteenvey attrib (aiogram.types		
	·	cy.Currency attribute			method), 35			
		cy.Currency attribute			start_link()	(in	module	
CODE (aid	-	ge_entity_type.Messc	igeEntity'		aiogram.utils.deep	_linking), 577		
	attribute), 467			CreateC	hatInviteLink	(class	in	
	ass in aiogram.utils.	-			aiogram.methods.c	create_chat_invite	_link),	
command		mmand.CommandOl	oject at-		319			
	tribute), 486			CreateF	orumTopic	(class	in	
command		_command.BotComn	and at-		aiogram.methods.c		c), 320	
	tribute), 19			CreateI	nvoiceLink	(class	in	
		ilters.command), 485			aiogram.methods.c		k), 444	
		ogram.filters.commar				(class	in	
	attribute), 467	nenu_button_type.Me			aiogram.methods.c 289	create_new_sticker	r_set),	
command		s.set_my_commands.	SetMyCor	noncenatives				
	attribute), 407				(aiogram.methods.	create_chat_invite	e_link.Create	<i>ChatInviteLink</i>
CONNECT		ram.enums.content_ty	pe.Conte		attribute), 319			
	attribute), 460			creates	_join_request			
connect	ed_website (<i>aiog attribute</i>), 148	gram.types.message.l	Aessage		(<i>aiogram.methods. attribute</i>), 327	edit_chat_invite_l	ink.EditChati	'nviteLink
ConstI1	8nMiddleware	(class	in	creates	_join_request			
	aiogram.utils.i18n.	middleware), 553			(aiogram.types.chd	ut_invite_link.Cha	tInviteLink	
CONTACT	(aiogram.enums.co tribute), 459	ontent_type.Content1	Type at-	CREATOR	attribute), 49 (aiogram.enums.ch	at_member_status	s.ChatMembe	rStatus
CONTACT	(aiogram.enums.in	line_query_result_typ	oe.InlineQ					
	attribute), 465				(aiogram.types.che	at_invite_link.Cha	tInviteLink	
contact	(aiogram.types.exte	ernal_reply_info.Exte	rnalReply	yInfo	attribute), 49			
	attribute), 118			credent	ials(aiogram.type	es.passport_data.P	PassportData	
contact	(aiogram.types.mes	ssage.Message attribi	ıte), 147		attribute), 269			
Contact	(class in aiogram.t	ypes.contact), 114		currenc	y (aiogram.methods	s.create_invoice_li	ink.CreateInv	oiceLink
content	_type (aiogram.ty	pes.message.Messag	e prop-		attribute), 445			
	<i>erty</i>), 150			currenc	y (aiogram.metho	ods.send_invoice.S	endInvoice	
Content	Type (class in a	iogram.enums.conten	t_type),		attribute), 448			
	459			currenc	y (aiogram.types.in	put_invoice_messa	age_content.I	nputInvoiceMes
COP (aiog	gram.enums.currenc	cy.Currency attribute), 461		attribute), 259			
copy()	(aiogram.utils.keyb	oard.InlineKeyboard	lBuilder		y (aiogram.types.in			
	method), 550			currenc	y (aiogram.types.pr	e_checkout_query	PreCheckout	Query
copy()	(aiogram.utils.keyl	board.ReplyKeyboard	lBuilder		attribute), 280			
	<i>method</i>), 551			currenc	y (aiogram.types.su	ccessful_payment.	.SuccessfulPa	yment
copy_to	() (aiogram.types.	.message.Message n	nethod),		attribute), 283			
	184			Currenc	y (class in aiogram	.enums.currency),	461	
CopyMes	sage (class in aiogr	ram.methods.copy_m	essage),	CUSTOM_	EMOJI (aiogram.en	ums.message_enti	ty_type.Messa	igeEntityType
	315				attribute), 467			
CopyMes	sages	(class	in	CUSTOM_	EMOJI (aiogram.en	ums.reaction_type	_type.Reactio	nTypeType
	aiogram.methods.c	copy_messages), 317			attribute), 469			
correct	_option_id(aiogr	am.methods.send_po	ll.SendPo	CUSTOM_	EMOJI (<i>aiogram.en</i>	ums.sticker_type.S	StickerType	
	attribute), 385				attribute), 470			
correct	_option_id(<i>aiogr</i> 198	ram.types.poll.Poll at	tribute),	custom_	<pre>emoji_id(aiogran attribute), 299</pre>	n.methods.set_cust	tom_emoji_sti	cker_set_thuml
country	code (<i>aiogram.typ attribute</i>), 281	pes.shipping_address	Shipping	A clalse.co m_		n.types.message_e	ntity.Message	Entity
country		ypes.giveaway.Givea	way at-	custom_		n.types.reaction_ty	pe_custom_e	moji.ReactionT

<pre>custom_emoji_id (aiogram.types.sticker.Sticker at- tribute), 265</pre> day (aiogram.types.birthdate.Birthdate attrib decline() (aiogram.types.chat_join_request	
custom_emoji_ids (aiogram.methods.get_custom_emoji_stickers.Getethstah),EthojiStickers	1
attribute), 293 DeclineChatJoinRequest (class	in
custom_emoji_sticker_set_name aiogram.methods.decline_chat_join	n_request),
(aiogram.types.chat.Chat attribute), 32 321	
custom_title (aiogram.methods.set_chat_administrator_cdustcod_etiplex%2tGdd(t)Administrator CinstomTitle	e module
attribute), 398 aiogram.utils.deep_linking), 577	
custom_title(aiogram.types.chat_member_administratorDEFANULAr(biorgvdmixeisturatschrot_command_scop_ attribute), 90 attribute), 457	
custom_title (aiogram.types.chat_member_owner.ChatMDEBAUOTwwwiogram.enums.menu_button_type.	MenuButtonType
attribute), 91 attribute), 467	in
CustomEmoji (class in aiogram.utils.formatting), 572DefaultKeyBuilder(classCZK (aiogram.enums.currency.Currency attribute), 462aiogram.fsm.storage.redis), 516	in
_ delete() (aiogram.types.message.Message	e method)
D 189	
DART (aiogram.enums.dice_emoji.DiceEmoji attribute), DELETE_CHAT_PHOTO (aiogram.enums.conten 464 attribute), 460	
DART (<i>aiogram.types.dice.DiceEmoji attribute</i>), 115 delete_chat_photo (<i>aiogram.types.messa</i>)	ige.Message
DATA (aiogram.enums.passport_element_error_type.PassportElementEltribute), 147 attribute), 468	icker.Sticker
data (aiogram.types.callback_query.CallbackQuery at- method), 266	
tribute), 28 delete_message() (aiogram.types.chat.Ch	at method),
data (aiogram.types.encrypted_credentials.EncryptedCredentials 34 attribute), 267 delete_photo() (aiogram.types.chat.Chat i	nethod), 42
data(aiogram.types.encrypted_passport_element.Encryptedpassport_enplay_markup()	method),
	memou),
tribute), 269 delete_sticker_set() (aiogram.type	es.chat.Chat
data (aiogram.types.web_app_data.WebAppData at- tribute) 214	110
110410), 214	in
data_hash(aiogram.types.passport_element_error_data_field.Passp@HPBrememethedhDdalefterkhat_photo attribute), 271 DeleteChatStickerSet (class	in
date (aiogram.types.business_connection.BusinessConnectionaiogram.methods.delete_chat_stickedattribute), 25323	er_set),
date (aiogram.types.chat_join_request.ChatJoinRequest DELETED_BUSINESS_MESSAGES	
attribute), 50 (aiogram.enums.update_type.Update_type.	teType
date (aiogram.types.chat_member_updated.ChatMemberUpdatedattribute), 470	
attribute), 94 deleted_business_messages date(aiogram types inaccessible message InaccessibleMessage (aiogram.types.update.Update	attribute),
attribute), 125	
date (aiogram.types.message.Message attribute), 145 DeleteForumTopic (class	in
date (aiogram.types.message_origin_channel.MessageOriginChannepiogram.methods.delete_forum_top attribute), 193 DeleteMessage (class	ic), 324 in
date(aiogram.types.message_origin_chat.MessageOriginChat aiogram.methods.delete_message),	420
attribute), 194 DeleteMessages (class	in
date (aiogram.types.message_origin_hidden_user.MessageOriginHiddlePapapp_methods.delete_messages) attribute), 195 DeleteMyCommands (class	, 422 in
date(aiogram.types.message_origin_user.MessageOriginUser aiogram.methods.delete_my_comm	ands),
attribute), 195 325 date (aiogram.types.message_reaction_count_updated.MessageRetestionConfigurated (class	in
attribute), 196 aiogram.methods.delete_sticker_fro	om_set),
date (aiogram.types.message_reaction_updated.MessageReactionUpdated attribute), 197 DeleteStickerSet (class	in

aiogram.methods.delete_sticker_set), 292	attribute), 131
DeleteWebhook (class in	disable_edit_message
aiogram.methods.delete_webhook), 450	(aiogram.methods.set_game_score.SetGameScore
description (aiogram.methods.create_invoice_link.Creat	
	disable_notification
description(aiogram.methods.send_invoice.SendInvoice	
attribute), 448	attribute), 316
description (aiogram.methods.set_chat_description.SetC	
attribute), 399	(aiogram.methods.copy_messages.CopyMessages
description(aiogram.methods.set_my_description.SetMy	
attribute), 409	disable_notification
<pre>description(aiogram.types.bot_command.BotCommand</pre>	attribute), 332
description(aiogram.types.bot_description.BotDescripti	
attribute), 24	(aiogram.methods.forward_messages.ForwardMessages
description (aiogram.types.chat.Chat attribute), 31	attribute), 333
description (aiogram.types.game.Game attribute), 287	
attribute), 220	nlineQuery(ResyltAntinele hods.pin_chat_message.PinChatMessage attribute), 352
<pre>description(aiogram.types.inline_query_result_cached_</pre>	
attribute), 225	(aiogram.methods.send_animation.SendAnimation
description(aiogram.types.inline_query_result_cached_	
attribute), 231	disable_notification
<pre>description(aiogram.types.inline_query_result_cached_</pre>	video.Inlin a.govercyNesvetiCaskaatidi<u>d</u>avad io.SendAudio attribute), 365
description (aiogram.types.inline_query_result_document	
attribute), 241	(aiogram.methods.send_contact.SendContact
description (aiogram.types.inline_query_result_photo.In	
attribute), 250	disable_notification
description(aiogram.types.inline_query_result_video.In	lineQuery RaisogtVinden ethods.send_dice.SendDice at-
attribute), 254	tribute), 370
description(aiogram.types.input_invoice_message_content	e di.sapilennoitaMecatgect ontent
attribute), 258	(aiogram.methods.send_document.SendDocument
description (aiogram.types.invoice.Invoice attribute),	attribute), 373
278	disable_notification
DetailedAiogramError, 540	(aiogram.methods.send_game.SendGame
DICE (aiogram.enums.content_type.ContentType at-	attribute), 439
tribute), 459	disable_notification
DICE (aiogram.enums.dice_emoji.DiceEmoji attribute), 464	(aiogram.methods.send_invoice.SendInvoice attribute), 449
DICE (aiogram.types.dice.DiceEmoji attribute), 115	disable_notification
dice (aiogram.types.external_reply_info.ExternalReplyInfo	
attribute), 118 dice (aiogram.types.message.Message attribute), 147	<i>attribute</i>), 376 disable_notification
Dice (class in aiogram.types.dice), 115	(aiogram.methods.send_media_group.SendMediaGroup
DiceEmoji (class in aiogram.enums.dice_emoji), 464	attribute), 378
DiceEmoji (class in alogram.types.dice), 115	disable_notification
disable() (aiogram.utils.callback_answer.CallbackAnswe	
method), 565	attribute), 380
disable_content_type_detection	disable_notification
(aiogram.methods.send_document.SendDocumen	t (aiogram.methods.send_photo.SendPhoto
attribute), 373	attribute), 383
disable_content_type_detection	disable_notification
(aiogram.types.input_media_document.InputMed	iaDocumentaiogram.methods.send_poll.SendPoll at-

tribute), 385	attribute), 464
disable_notification	drop_pending_updates
(aiogram.methods.send_sticker.SendSticker	(aiogram.methods.delete_webhook.DeleteWebhook
attribute), 297	attribute), 450
disable_notification	drop_pending_updates
(aiogram.methods.send_venue.SendVenue attribute) 288	(aiogram.methods.set_webhook.SetWebhook
attribute), 388 disable_notification	attribute), 454 duration (aiogram.methods.send_animation.SendAnimation
(aiogram.methods.send_video.SendVideo	attribute), 361
attribute), 391	duration (aiogram.methods.send_audio.SendAudio at-
disable_notification	tribute), 364
	nduration (aiogram.methods.send_video.SendVideo at-
attribute), 394	tribute), 390
disable_notification	<pre>duration(aiogram.methods.send_video_note.SendVideoNote</pre>
(aiogram.methods.send_voice.SendVoice	attribute), 393
attribute), 396	duration (aiogram.methods.send_voice.SendVoice at-
disable_web_page_preview	tribute), 396
(aiogram.methods.edit_message_text.EditMessag attribute), 430	edfaration (aiogram.types.animation.Animation at- tribute), 17
disable_web_page_preview	duration (aiogram.types.audio.Audio attribute), 18
(aiogram.methods.send_message.SendMessage	${\tt duration} (a iogram.types.input_media_animation.InputMediaAnimation$
attribute), 380	attribute), 129
disable_web_page_preview	duration (aiogram.types.input_media_audio.InputMediaAudio
(aiogram.types.input_text_message_content.Input attribute), 262	duration (aiogram.types.input_media_video.InputMediaVideo
disabled (aiogram.utils.callback_answer.CallbackAnswer	
property), 565	duration (aiogram.types.video.Video attribute), 211
Dispatcher (class in aiogram.dispatcher.dispatcher),	duration (aiogram.types.video_chat_ended.VideoChatEnded
481	attribute), 212
<pre>distance (aiogram.types.proximity_alert_triggered.Proxin</pre>	n dnyxAzeri Bri gge(naibgram.types.video_note.VideoNote at- tribute), 213
DKK (aiogram.enums.currency.Currency attribute), 462	duration (aiogram.types.voice.Voice attribute), 214
	DZD (aiogram.enums.currency.Currency attribute), 462
DOCUMENT (aiogram.enums.content_type.ContentType at-	
tribute), 459	E
DOCUMENT (aiogram.enums.inline_query_result_type.Inline	Querv <u>ReaphTom()</u> (aiogram.types.message.Message
attribute), 465	method), 189
DOCUMENT (aiogram.enums.input_media_type.InputMediaT_ attribute), 466	Vedit_date (aiogram.types.message.Message attribute),
document (aiogram.methods.send_document.SendDocument	146
attribute), 373	<pre>%dit_invite_link() (aiogram.types.chat.Chat method), 34</pre>
document (aiogram.types.external_reply_info.ExternalRepl	Wellfe_live_location()
attribute), 117	(aiogram.types.message.Message method),
document (aiogram.types.message.Message attribute),	188
146	<pre>edit_media() (aiogram.types.message.Message</pre>
Document (<i>class in aiogram.types.document</i>), 116	method), 187
<pre>document_file_id(aiogram.types.inline_query_result_co</pre>	
attribute), 225	(aiogram.types.message.Message method),
<pre>document_url (aiogram.types.inline_query_result_document_ attribute), 241</pre>	107
DOP (aiogram.enums.currency.Currency attribute), 462	<pre>edit_text() (aiogram.types.message.Message method),</pre>
download() (aiogram.client.bot.Bot method), 472	185 EditChatTruitaLink (alass in
download_file() (aiogram.client.bot.Bot method), 472	EditChatInviteLink (class in
DRIVER_LICENSE (aiogram.enums.encrypted_passport_ele	aiogram.methods.edit_chat_invite_link), ment.EncryptedPassportElement
pussponetc	JZU az absport Zienien

EDITED_BUSINESS_MESSAGE (aiogram.enums.update_type.UpdateType attribute), 470	<pre>emoji (aiogram.types.sticker.Sticker attribute), 265 emoji_list (aiogram.methods.set_sticker_emoji_list.SetStickerEmojiList</pre>
edited_business_message	emoji_list (aiogram.types.input_sticker.InputSticker
(aiogram.types.update.Update attribute), 284	attribute), 264
	emoji_status_custom_emoji_id
EDITED_CHANNEL_POST	(aiogram.types.chat.Chat attribute), 31
(aiogram.enums.update_type.UpdateType	emoji_status_expiration_date
attribute), 470	(aiogram.types.chat.Chat attribute), 31 EncryptedCredentials (class in
edited_channel_post (<i>aiogram.types.update.Update</i> <i>attribute</i>), 284	
EDITED_MESSAGE (aiogram.enums.update_type.UpdateTyp	aiogram.types.encrypted_credentials), 267 ÆncryptedPassportElement (class in
attribute), 470	
	aiogram.enums.encrypted_passport_element), 464
edited_message (aiogram.types.update.Update at-	
tribute), 284	EncryptedPassportElement (class in
EditForumTopic (class in	aiogram.types.encrypted_passport_element),
aiogram.methods.edit_forum_topic), 328	268
	enter() (aiogram.fsm.scene.ScenesManager method),
aiogram.methods.edit_general_forum_topic),	532 5 () () () () () () () () () (
329	enter() (aiogram.fsm.scene.SceneWizard method), 533
	entities (aiogram.methods.edit_message_text.EditMessageText
aiogram.methods.edit_message_caption),	attribute), 430
423	<pre>entities(aiogram.methods.send_message.SendMessage</pre>
EditMessageLiveLocation (class in	attribute), 380
aiogram.methods.edit_message_live_location),	entities (aiogram.types.input_text_message_content.InputTextMessageContent.InputSegeContent.InputSegeContent.InputSegeContent.InputSegeContent.InputSegeContent.InputSegeContent.InputSegeContent.InputSegeContent.InputSegeContent.InputSegeContent.InputSegeContent.InputSegeContent.InputSegeContent.InputSegeContent.InputSegeContent.InputSegeCon
424	attribute), 262
EditMessageMedia (class in	entities (aiogram.types.message.Message attribute),
aiogram.methods.edit_message_media), 426	146
EditMessageReplyMarkup (class in	entities (aiogram.types.text_quote.TextQuote at-
aiogram.methods.edit_message_reply_markup),	tribute), 207
428	error_message (aiogram.methods.answer_pre_checkout_query.AnswerPre_chec
EditMessageText (class in	attribute), 442
aiogram.methods.edit_message_text), 429	error_message (aiogram.methods.answer_shipping_query.AnswerShipping_
EGP (aiogram.enums.currency.Currency attribute), 462	attribute), 443
<pre>element_hash(aiogram.types.passport_element_error_un</pre>	n SpecifinEd/Bast sp olictEl/emaintErrorn:Uppepsecrified_ event), 539
attribute), 277	errors (aiogram.methods.set_passport_data_errors.SetPassportDataError
EMAIL (aiogram.enums.encrypted_passport_element.Encry	
attribute), 465	ETB (aiogram.enums.currency.Currency attribute), 462
EMAIL (aiogram.enums.message_entity_type.MessageEntity	Expre (aiogram.enums.currency.Currency attribute), 462
attribute), 467	<pre>event (aiogram.types.update.Update property), 286</pre>
<pre>email(aiogram.types.encrypted_passport_element.Encryp</pre>	t exRasspbytFel emaintgram.types.update.Update property),
attribute), 268	286
<pre>email (aiogram.types.order_info.OrderInfo attribute),</pre>	exception (aiogram.types.error_event.ErrorEvent at-
279	tribute), 539
Email (class in aiogram.utils.formatting), 571	ExceptionMessageFilter (class in
EMOJI (aiogram.enums.reaction_type_type.ReactionTypeTy	pe aiogram.filters.exception), 495
attribute), 469	exceptions (aiogram.filters.exception.ExceptionTypeFilter
<pre>emoji (aiogram.methods.send_dice.SendDice attribute),</pre>	attribute), 495
370	ExceptionTypeFilter (class in
emoji (aiogram.methods.send_sticker.SendSticker	aiogram.filters.exception), 495
attribute), 297	exit() (aiogram.fsm.scene.SceneWizard method), 533
emoji (aiogram.types.dice.Dice attribute), 115	expiration_date(aiogram.types.chat_boost.ChatBoost
emoji (aiogram.types.reaction_type_emoji.ReactionTypeE	
attribute), 201	expire_date(aiogram.methods.create_chat_invite_link.CreateChatInvite

attribute), 319	file_hash(aiogram.types.passport_element_error_reverse_side.Passport_
<pre>expire_date(aiogram.methods.edit_chat_invite_link.Edit</pre>	ChatInviteLintibute), 274
attribute), 327	$\verb"file_hash"(a iogram.types.passport_element_error_selfie.PassportElemen$
<pre>expire_date(aiogram.types.chat_invite_link.ChatInviteL</pre>	ink attribute), 275
attribute), 50	<pre>file_hash(aiogram.types.passport_element_error_translation_file.Passport_element_error_translation_filement_error_translation_file.Passport_element_error_transla</pre>
explanation (aiogram.methods.send_poll.SendPoll at-	attribute), 276
tribute), 385	file_hashes(aiogram.types.passport_element_error_files.PassportEleme
explanation (aiogram.types.poll.Poll attribute), 198	attribute), 272
explanation_entities	file_hashes(aiogram.types.passport_element_error_translation_files.Pa
(aiogram.methods.send_poll.SendPoll at-	attribute), 277
tribute), 385	file_id (aiogram.methods.get_file.GetFile attribute),
explanation_entities (aiogram.types.poll.Poll	340
attribute), 198	<pre>file_id (aiogram.types.animation.Animation attribute),</pre>
explanation_parse_mode	17
	file_id (aiogram.types.audio.Audio attribute), 18
tribute), 385	<pre>file_id (aiogram.types.document.Document attribute),</pre>
$\verb+export() (a iogram.utils.keyboard.In lineKeyboardBuilder$	116
<i>method</i>), 550	<pre>file_id (aiogram.types.file.File attribute), 119</pre>
$\verb+export()(a iogram.utils.keyboard.ReplyKeyboardBuilder$	
<i>method</i>), 551	tribute), 278
<pre>export_invite_link() (aiogram.types.chat.Chat</pre>	<pre>file_id (aiogram.types.photo_size.PhotoSize attribute),</pre>
method), 35	197
ExportChatInviteLink (class in	<pre>file_id (aiogram.types.sticker.Sticker attribute), 265</pre>
aiogram.methods.export_chat_invite_link),	<pre>file_id (aiogram.types.video.Video attribute), 211</pre>
330	file_id (aiogram.types.video_note.VideoNote at-
<pre>external_reply (aiogram.types.message.Message at-</pre>	tribute), 213
tribute), 145	<pre>file_id (aiogram.types.voice.Voice attribute), 214</pre>
	file_name (aiogram.types.animation.Animation at-
aiogram.types.external_reply_info), 117	tribute), 18
<pre>extract_flags() (in module aiogram.dispatcher.flags),</pre>	<pre>file_name (aiogram.types.audio.Audio attribute), 18</pre>
542	file_name (aiogram.types.document.Document at-
<pre>extract_from() (aiogram.types.message_entity.Message)</pre>	
<i>method</i>), 192	<pre>file_name (aiogram.types.video.Video attribute), 211</pre>
EYES (aiogram.enums.mask_position_point.MaskPositionPeters)	
attribute), 466	file_size (aiogram.types.animation.Animation at-
F	tribute), 18
F	file_size (aiogram.types.audio.Audio attribute), 19
<pre>feed_raw_update() (aiogram.dispatcher.dispatcher.Disp</pre>	afile_size (aiogram.types.document.Document at-
method), 481	tribute), 116
<pre>feed_update() (aiogram.dispatcher.dispatcher.Dispatche</pre>	rfile_size (aiogram.types.file.File attribute), 119
<i>method</i>), 481	file_size (aiogram.types.passport_file.PassportFile at-
<pre>field_name(aiogram.types.passport_element_error_data</pre>	field.PassfröhteromentErrorDataField
attribute), 270	file_size (aiogram.types.photo_size.PhotoSize at-
file (aiogram.client.telegram.TelegramAPIServer	tribute), 197
attribute), 12	<pre>file_size (aiogram.types.sticker.Sticker attribute), 266</pre>
FILE (aiogram.enums.passport_element_error_type.Passpo	h illensing (diggra m.types.video.Video attribute), 212
attribute), 468	file_size (aiogram.types.video_note.VideoNote at-
File (class in aiogram.types.file), 119	tribute), 213
<pre>file_date(aiogram.types.passport_file.PassportFile at-</pre>	<pre>file_size (aiogram.types.voice.Voice attribute), 214</pre>
tribute), 278	file_unique_id (aiogram.types.animation.Animation
file_hash(aiogram.types.passport_element_error_file.Pa	ssportElen####################################
attribute), 271	file_unique_id (aiogram.types.audio.Audio attribute),
<pre>file_hash(aiogram.types.passport_element_error_front_</pre>	side.PassportElementErrorFrontSide
attribute), 273	file_unique_id (aiogram.types.document.Document

attribute), 116	attribute), 466
file_unique_id (<i>aiogram.types.file.File attribute</i>), 119 file_unique_id (<i>aiogram.types.passport_file.PassportFil</i>	<pre>format (aiogram.methods.set_sticker_set_thumbnail.SetStickerSetThumbna e attribute), 304</pre>
attribute), 278	format (aiogram.types.input_sticker.InputSticker at-
<pre>file_unique_id (aiogram.types.photo_size.PhotoSize</pre>	tribute), 263
attribute), 197	<pre>FORUM_TOPIC_CLOSED (aiogram.enums.content_type.ContentType</pre>
file_unique_id (aiogram.types.sticker.Sticker at-	attribute), 460
tribute), 265	<pre>forum_topic_closed (aiogram.types.message.Message</pre>
file_unique_id (aiogram.types.video.Video attribute),	attribute), 149
211	FORUM_TOPIC_CREATED
<pre>file_unique_id (aiogram.types.video_note.VideoNote</pre>	(aiogram.enums.content_type.ContentType attribute), 460
	forum_topic_created
214	(aiogram.types.message.Message attribute),
<pre>file_url() (aiogram.client.telegram.TelegramAPIServer</pre>	149
	FORUM_TOPIC_EDITED (<i>aiogram.enums.content_type.ContentType</i>
FILES (aiogram.enums.passport_element_error_type.Passp	
attribute), 468	forum_topic_edited (aiogram.types.message.Message
files (aiogram.types.encrypted_passport_element.Encrypt attribute), 268	FORUM_TOPIC_REOPENED
Filter (class in aiogram.filters.base), 496	(aiogram.enums.content_type.ContentType
filter() (aiogram.filters.callback_data.CallbackData	attribute), 460
class method), 493	forum_topic_reopened
FIND_LOCATION (aiogram.enums.chat_action.ChatAction	(aiogram.types.message.Message attribute),
attribute), 458	149
find_location() (aiogram.utils.chat_action.ChatActionS	FordramTopic (class in aiogram.types.forum_topic), 120ForumTopicClosed(class in
first_name (aiogram.methods.send_contact.SendContact)	aiogram.types.forum_topic_closed), 120
	ForumTopicCreated (class in
first_name (<i>aiogram.types.chat.Chat attribute</i>), 30	aiogram.types.forum_topic_created), 121
<pre>first_name (aiogram.types.contact.Contact attribute),</pre>	
114	aiogram.types.forum_topic_edited), 121
<pre>first_name(aiogram.types.inline_query_result_contact.In</pre>	horQuTopRcRettponedt (class in
attribute), 238	aiogram.types.forum_topic_reopened), 122
<pre>first_name (aiogram.types.input_contact_message_conten</pre>	n £durmanCont ad thilegisauge Gp asemie ssage.Message method), 186
<pre>first_name (aiogram.types.shared_user.SharedUser at-</pre>	forward_date (aiogram.types.message.Message at-
tribute), 205	tribute), 149
<pre>first_name (aiogram.types.user.User attribute), 207</pre>	forward_from (aiogram.types.message.Message at-
<pre>first_name (aiogram.utils.web_app.WebAppUser</pre>	tribute), 150
attribute), 561	forward_from_chat (aiogram.types.message.Message
FOOTBALL (aiogram.enums.dice_emoji.DiceEmoji	attribute), 150
attribute), 464	forward_from_message_id
FOOTBALL (aiogram.types.dice.DiceEmoji attribute), 115	(aiogram.types.message.Message attribute),
for_channels (aiogram.methods.get_my_default_administ	
attribute), 344	forward_origin (aiogram.types.message.Message at-
<pre>for_channels(aiogram.methods.set_my_default_administ attribute), 409</pre>	forward_sender_name
<pre>force (aiogram.methods.set_game_score.SetGameScore</pre>	(aiogram.types.message.Message attribute),
attribute), 441	150
	forward_signature (aiogram.types.message.Message attribute), 150
ForceReply (class in aiogram.types.force_reply), 119	forward_text (aiogram.types.login_url.LoginUrl
FOREHEAD (aiogram.enums.mask_position_point.MaskPosit	

Index

ForwardMessage (class in tribute), 216
aiogram.methods.forward_message), 331 from_user (aiogram.types.message.Message attribute),
ForwardMessages (<i>class</i> in 145
aiogram.methods.forward_messages), 333 from_user(aiogram.types.pre_checkout_query.PreCheckoutQuery
foursquare_id(aiogram.methods.send_venue.SendVenue attribute), 280
attribute), 388 from_user (aiogram.types.shipping_query.ShippingQuery
foursquare_id(aiogram.types.inline_query_result_venue.InlineQueartyRidsult),ettale
attribute), 252 FRONT_SIDE (aiogram.enums.passport_element_error_type.PassportEleme
foursquare_id(aiogram.types.input_venue_message_content.InputVatmibMe);AgeContent attribute), 263 front_side(aiogram.types.encrypted_passport_element.EncryptedPasspo
foursquare_id (aiogram.types.venue.Venue attribute), attribute), 268
210 FSInputFile (class in aiogram.types.input_file), 127,
foursquare_type (aiogram.methods.send_venue.SendVenue 473
attribute), 388 FSMI18nMiddleware (class in
foursquare_type(aiogram.types.inline_query_result_venue.InlineQuiogyResultiVe.inlu8n.middleware), 554
attribute), 252 full_name (aiogram.types.chat.Chat property), 33
foursquare_type (aiogram.types.input_venue_message_cdmlelut_hapnet Vaioger Musssyge Closede.ldser property), 208
attribute), 263 foursquare_type (aiogram.types.venue.Venue at- G
tribute), 211GAME(aiogram.enums.content_type.ContentTypeat-from_attachment_menutribute), 459
(aiogram.types.write_access_allowed.WriteAccessAllowedlogram.enums.inline_query_result_type.InlineQueryResultType
attribute), 215 attribute) 465
from_base() (aiogram.client.telegram.TelegramAPIServergame (aiogram.types.external_reply_info.ExternalReplyInfo
class method), 13 attribute), 118
from_chat_id (aiogram.methods.copy_message.CopyMessgame (aiogram.types.message.Message attribute), 147
attribute), 315 Game (class in aiogram.types.game), 287
<pre>from_chat_id(aiogram.methods.copy_messages.CopyMessages_short_name(aiogram.methods.send_game.SendGame attribute), 317</pre>
attribute), 317 from_chat_id (aiogram.methods.forward_message.Forward_messge6rt_name (aiogram.types.callback_query.CallbackQuery
attribute), 331 attribute) 28
from_chat_id(aiogram.methods.forward_messages.Forward_messn895t_name(aiogram.types.inline_query_result_game.InlineQueryF
attribute), 333 attribute), 242
from_file() (aiogram.types.input_file.BufferedInputFile GameHighScore (class in
class method), 127 aiogram.types.game_high_score), 288
from_markup() (aiogram.utils.keyboard.InlineKeyboardBuitger(aiogram.enums.currency.Currency attribute), 462 class method), 550 GFL (aiogram enums currency Currency attribute), 462
class method), 550 GEL (aiogram.enums.currency.Currency attribute), 462 from_markup() (aiogram.utils.keyboard.ReplyKeyboardBuileNERAL_FORUM_TOPIC_HIDDEN
class method), 551 (aiogram.enums.content_type.ContentType
from_request (aiogram.types.write_access_allowed.WriteAccessAllowedbute), 460
attribute), 215 general_forum_topic_hidden
from_url() (aiogram.fsm.storage.redis.RedisStorage (aiogram.types.message.Message attribute),
class method), 515 149
from_user (aiogram.handlers.callback_query.CallbackQueryHernalerFORUM_TOPIC_UNHIDDEN property), 544 (aiogram enums content_type ContentType
property), 544 (aiogram.enums.content_type.ContentType from_user(aiogram.types.callback_query.CallbackQuery attribute), 460
<i>attribute</i>), 28 general_forum_topic_unhidden
from_user(aiogram.types.chat_join_request.ChatJoinRequest (aiogram.types.message.Message attribute),
attribute), 50
from_user(aiogram.types.chat_member_updated.ChatMember_linder6fumTopicHidden (class in
attribute), 93 aiogram.types.general_forum_topic_hidden),
from_user(aiogram.types.chosen_inline_result.ChosenInlineResult 122
attribute), 216 GeneralForumTopicUnhidden (class in aiogram.types.inline_query.InlineQuery at-
from_user (aiogram.types.inline_query.InlineQuery at- aiogram.types.general_forum_topic_unhidden),

122	346
get() (aiogram.fsm.scene.SceneRegistry method), 531	GetMyShortDescription (class in
<pre>get_administrators() (aiogram.types.chat.Chat method), 33</pre>	aiogram.methods.get_my_short_description), 347
<pre>get_data() (aiogram.fsm.scene.SceneWizard method), 533</pre>	GetStickerSet (class in aiogram.methods.get_sticker_set), 294
<pre>get_data() (aiogram.fsm.storage.base.BaseStorage</pre>	GetUpdates (class in aiogram.methods.get_updates), 451
get_flag() (in module aiogram.dispatcher.flags), 542	GetUserChatBoosts (class in
<pre>get_locale() (aiogram.utils.i18n.middleware.I18nMiddl</pre>	eware aiogram.methods.get_user_chat_boosts), 347
<pre>get_member() (aiogram.types.chat.Chat method), 37</pre>	GetUserProfilePhotos (class in
get_member_count() (aiogram.types.chat.Chat method), 37	aiogram.methods.get_user_profile_photos), 348
<pre>get_profile_photos() (aiogram.types.user.User</pre>	GetWebhookInfo (class in aiogram.methods.get_webhook_info), 453
<pre>get_state() (aiogram.fsm.storage.base.BaseStorage</pre>	GIF (aiogram.enums.inline_query_result_type.InlineQueryResultType attribute), 465
<pre>get_url() (aiogram.types.message.Message method),</pre>	<pre>gif_duration(aiogram.types.inline_query_result_gif.InlineQueryResultG</pre>
GetBusinessConnection (class in	gif_file_id(aiogram.types.inline_query_result_cached_gif.InlineQueryl
aiogram.methods.get_business_connection), 334	attribute), 226 gif_height (aiogram.types.inline_query_result_gif.InlineQueryResultGif
GetChat (<i>class in aiogram.methods.get_chat</i>), 335	attribute), 243
GetChatAdministrators (class in aiogram.methods.get_chat_administrators),	<pre>gif_url (aiogram.types.inline_query_result_gif.InlineQueryResultGif</pre>
336	gif_width(aiogram.types.inline_query_result_gif.InlineQueryResultGif
GetChatMember (class in	attribute), 243
aiogram.methods.get_chat_member), 337 GetChatMemberCount (class in	GIFT_CODE (aiogram.enums.chat_boost_source_type.ChatBoostSourceType attribute), 458
aiogram.methods.get_chat_member_count), 338	GIVEAWAY (aiogram.enums.chat_boost_source_type.ChatBoostSourceType attribute), 458
GetChatMenuButton (class in	~ ~ ~
aiogram.methods.get_chat_menu_button),	tribute), 460
339	giveaway (aiogram.types.external_reply_info.ExternalReplyInfo
GetCustomEmojiStickers (class in	attribute), 118
aiogram.methods.get_custom_emoji_stickers), 293	giveaway (aiogram.types.message.Message attribute), 149
GetFile (class in aiogram.methods.get_file), 340	Giveaway (class in aiogram.types.giveaway), 122
-	GIVEAWAY_COMPLETED (aiogram.enums.content_type.ContentType
aiogram.methods.get_forum_topic_icon_stickers)	
341	M
CatCamaHighScores (class in	giveaway_completed (aiogram.types.message.Message
GetGameHighScores (class in aiogram.methods.get_game_high_scores), 437	attribute), 149 GIVEAWAY_CREATED (aiogram.enums.content_type.ContentType
aiogram.methods.get_game_high_scores), 437 GetMe (class in aiogram.methods.get_me), 342	attribute), 149 GIVEAWAY_CREATED (aiogram.enums.content_type.ContentType attribute), 460 giveaway_created (aiogram.types.message.Message
aiogram.methods.get_game_high_scores), 437 GetMe (class in aiogram.methods.get_me), 342 GetMyCommands (class in	attribute), 149 GIVEAWAY_CREATED (aiogram.enums.content_type.ContentType attribute), 460 giveaway_created (aiogram.types.message.Message attribute), 149
aiogram.methods.get_game_high_scores), 437 GetMe (class in aiogram.methods.get_me), 342 GetMyCommands (class in aiogram.methods.get_my_commands), 343	attribute), 149 GIVEAWAY_CREATED (aiogram.enums.content_type.ContentType attribute), 460 giveaway_created (aiogram.types.message.Message attribute), 149 giveaway_message (aiogram.types.giveaway_completed.GiveawayComple
aiogram.methods.get_game_high_scores), 437 GetMe (class in aiogram.methods.get_me), 342 GetMyCommands (class in aiogram.methods.get_my_commands), 343	attribute), 149 GIVEAWAY_CREATED (aiogram.enums.content_type.ContentType attribute), 460 giveaway_created (aiogram.types.message.Message attribute), 149 giveaway_message (aiogram.types.giveaway_completed.GiveawayComple attribute), 123
aiogram.methods.get_game_high_scores), 437 GetMe (class in aiogram.methods.get_me), 342 GetMyCommands (class in aiogram.methods.get_my_commands), 343 GetMyDefaultAdministratorRights (class in aiogram.methods.get_my_default_administrator_ 344	attribute), 149 GIVEAWAY_CREATED (aiogram.enums.content_type.ContentType attribute), 460 giveaway_created (aiogram.types.message.Message attribute), 149 giveaway_message (aiogram.types.giveaway_completed.GiveawayComple attribute), 123
aiogram.methods.get_game_high_scores), 437 GetMe (class in aiogram.methods.get_me), 342 GetMyCommands (class in aiogram.methods.get_my_commands), 343 GetMyDefaultAdministratorRights (class in aiogram.methods.get_my_default_administrator_ 344 GetMyDescription (class in	attribute), 149 GIVEAWAY_CREATED (aiogram.enums.content_type.ContentType attribute), 460 giveaway_created (aiogram.types.message.Message attribute), 149 giveaway_message(aiogram.types.giveaway_completed.GiveawayComple attribute), 123 roghueaway_message_id (aiogram.types.chat_boost_source_giveaway.ChatBoostSourceGi attribute), 48
aiogram.methods.get_game_high_scores), 437 GetMe (class in aiogram.methods.get_me), 342 GetMyCommands (class in aiogram.methods.get_my_commands), 343 GetMyDefaultAdministratorRights (class in aiogram.methods.get_my_default_administrator_ 344	attribute), 149 GIVEAWAY_CREATED (aiogram.enums.content_type.ContentType attribute), 460 giveaway_created (aiogram.types.message.Message attribute), 149 giveaway_message (aiogram.types.giveaway_completed.GiveawayComple attribute), 123 rigiture), way_message_id (aiogram.types.chat_boost_source_giveaway.ChatBoostSourceGi

attribute) 124	has private forwards (giogram types chat Chat at
attribute), 124 GIVEAWAY_WINNERS (aiogram.enums.content_type.Conten	has_private_forwards (<i>aiogram.types.chat.Chat at-tType tribute</i>), 31
attribute), 460	has_protected_content (aiogram.types.chat.Chat at-
giveaway_winners (aiogram.types.external_reply_info.E	
attribute), 118	has_protected_content
giveaway_winners (aiogram.types.message.Message attribute), 149	(aiogram.types.message.Message attribute), 146
GiveawayCompleted (class in aiogram.types.giveaway_completed), 123	<pre>has_public_winners(aiogram.types.giveaway.Giveaway attribute), 123</pre>
GiveawayCreated (class in	has_restricted_voice_and_video_messages
aiogram.types.giveaway_created), 124	(aiogram.types.chat.Chat attribute), 31
aiogram.types.giveaway_winners), 124	has_spoiler (<i>aiogram.methods.send_animation.SendAnimation</i> <i>attribute</i>), 362
<pre>google_place_id(aiogram.methods.send_venue.SendVenu</pre>	
attribute), 388	attribute), 382
<pre>google_place_id(aiogram.types.inline_query_result_ve attribute), 252</pre>	n ha.s<u>n</u>/specywen yRamalgYamunethods.send_video.SendVideo attribute), 391
	aintoute), 591 c hass<u>n</u>spoill&enaedgessagefaminepu t_media_animation.InputMediaAnimatio
attribute), 263	attribute), 129
	<pre>has_spoiler (aiogram.types.input_media_photo.InputMediaPhoto</pre>
	Varuse_spoiler (aiogram.types.input_media_video.InputMediaVideo
attribute), 388	attribute), 134
<pre>google_place_type (aiogram.types.inline_query_result_ attribute), 252</pre>	vanse_viikine@aehiResortyVenue (aiogram.types.chat.Chat attribute), 32
<pre>google_place_type (aiogram.types.input_venue_messag attribute), 263</pre>	ehashteniolgypunt. Veypune Messeggtedon anendentials. Encrypted Credentials attribute), 267
	hash (aiogram.types.encrypted_passport_element.EncryptedPassportEleme attribute), 268
goto() (aiogram.fsm.scene.SceneWizard method), 533	hash (aiogram.utils.web_app.WebAppInitData attribute),
GREEN (aiogram.enums.topic_icon_color.TopicIconColor attribute), 470	561 HASHTAG (aiogram.enums.message_entity_type.MessageEntityType
GROUP (aiogram.enums.chat_type.ChatType attribute),	attribute), 467
459	HashTag (class in aiogram.utils.formatting), 570
GROUP_CHAT_CREATED (aiogram.enums.content_type.Cont attribute), 460	teheBding (aiogram.methods.edit_message_live_location.EditMessageLiveLe attribute), 425
group_chat_created (aiogram.types.message.Message attribute), 147	heading (aiogram.methods.send_location.SendLocation attribute), 375
GTQ (aiogram.enums.currency.Currency attribute), 462	<pre>heading(aiogram.types.inline_query_result_location.InlineQueryResultLo</pre>
Н	heading (aiogram.types.input_location_message_content.InputLocationMe
handlers (<i>aiogram.fsm.scene.SceneConfig attribute</i>), 532	attribute), 261
	heading (aiogram.types.location.Location attribute),
has_aggressive_anti_spam_enabled	140
(aiogram.types.chat.Chat attribute), 32	140 height (<i>aiogram.methods.send_animation.SendAnimation</i>
(aiogram.types.chat.Chat attribute), 32 has_custom_certificate (aiogram.types.webhook_info.WebhookInfo	140 height (aiogram.methods.send_animation.SendAnimation attribute), 361 height (aiogram.methods.send_video.SendVideo at-
<pre>(aiogram.types.chat.Chat attribute), 32 has_custom_certificate (aiogram.types.webhook_info.WebhookInfo attribute), 286 has_hidden_members (aiogram.types.chat.Chat at-</pre>	140 height (aiogram.methods.send_animation.SendAnimation attribute), 361
(aiogram.types.chat.Chat attribute), 32 has_custom_certificate (aiogram.types.webhook_info.WebhookInfo attribute), 286 has_hidden_members (aiogram.types.chat.Chat at- tribute), 32	140 height (aiogram.methods.send_animation.SendAnimation attribute), 361 height (aiogram.methods.send_video.SendVideo at- tribute), 390 height (aiogram.types.animation.Animation attribute), 17
<pre>(aiogram.types.chat.Chat attribute), 32 has_custom_certificate (aiogram.types.webhook_info.WebhookInfo attribute), 286 has_hidden_members (aiogram.types.chat.Chat at- tribute), 32 has_media_spoiler(aiogram.types.external_reply_info.</pre>	140 height (aiogram.methods.send_animation.SendAnimation attribute), 361 height (aiogram.methods.send_video.SendVideo at- tribute), 390
(aiogram.types.chat.Chat attribute), 32 has_custom_certificate (aiogram.types.webhook_info.WebhookInfo attribute), 286 has_hidden_members (aiogram.types.chat.Chat at- tribute), 32	140 height (aiogram.methods.send_animation.SendAnimation attribute), 361 height (aiogram.methods.send_video.SendVideo at- tribute), 390 height (aiogram.types.animation.Animation attribute), 17 Excipitr(ajngraph.types.input_media_animation.InputMediaAnimation

height (<i>aiogram.types.photo_size.PhotoSize attribute</i>), 197	<pre>icon_custom_emoji_id (aiogram.types.forum_topic_created.ForumTopicCreated</pre>
height (aiogram.types.sticker.Sticker attribute), 265	attribute), 121
height (alogram.types.video.Video attribute), 211	icon_custom_emoji_id
	ageOriginTypiogram.types.forum_topic_edited.ForumTopicEdited
attribute), 468	attribute), 121
hide_url (aiogram.types.inline_query_result_article.Inlin	eQuariogeesmitAypteslbusiness_connection.BusinessConnection
attribute), 220	attribute), 25
HideGeneralForumTopic (class in aiogram.methods.hide_general_forum_topic),	id (aiogram.types.callback_query.CallbackQuery at- tribute), 28
349	id (aiogram.types.chat.Chat attribute), 30
HKD (aiogram.enums.currency.Currency attribute), 462	id (aiogram.types.inline_query.InlineQuery attribute), 216
HNL (<i>aiogram.enums.currency.Currency attribute</i>), 462 horizontal_accuracy	
(aiogram.methods.edit_message_live_location.Ed	id (aiogram.types.inline_query_result_article.InlineQueryResultArticle ditMessagehitzibasettidt9
attribute), 425	id(aiogram.types.inline_query_result_audio.InlineQueryResultAudio
horizontal_accuracy	attribute), 221
(aiogram.methods.send_location.SendLocation	id (aiogram.types.inline_query_result_cached_audio.InlineQueryResultCa
attribute), 375	attribute), 222
horizontal_accuracy	id (aiogram.types.inline_query_result_cached_document.InlineQueryResult_ca
(aiogram.types.inline_query_result_location.Inli	neQueryResultIInda)j@25
attribute), 246	id (aiogram.types.inline_query_result_cached_gif.InlineQueryResultCached_gif.inlineQue
horizontal_accuracy	attribute), 226
(aiogram.types.input_location_message_content. attribute), 260	Inpl(Miogations)/MessaigliGongenetry_result_cached_mpeg4_gif.InlineQueryResi attribute), 229
horizontal_accuracy	$\verb"id"(a iogram.types.inline_query_result_cached_photo.InlineQueryResultCached_photo.InlineQuer$
(aiogram.types.location.Location attribute),	attribute), 231
140 HRK (aiogram.enums.currency.Currency attribute), 462	id (aiogram.types.inline_query_result_cached_sticker.InlineQueryResultCo attribute), 233
HTML (aiogram.enums.parse_mode.ParseMode attribute), 468	<pre>id(aiogram.types.inline_query_result_cached_video.InlineQueryResultCau attribute), 235</pre>
<pre>html_text (aiogram.types.message.Message property),</pre>	id (aiogram.types.inline_query_result_cached_voice.InlineQueryResultCac
150	attribute), 237
HUF (aiogram.enums.currency.Currency attribute), 462	id (aiogram.types.inline_query_result_contact.InlineQueryResultContact
1	attribute), 238
I18nMiddleware (class in	id (aiogram.types.inline_query_result_document.InlineQueryResultDocum attribute), 240
aiogram.utils.i18n.middleware), 554	id (aiogram.types.inline_query_result_game.InlineQueryResultGame
icon_color (aiogram.methods.create_forum_topic.Create	
attribute), 320	id (aiogram.types.inline_query_result_gif.InlineQueryResultGif
<pre>icon_color (aiogram.types.forum_topic.ForumTopic at-</pre>	attribute), 243
tribute), 120	id (aiogram.types.inline_query_result_location.InlineQueryResultLocation
<pre>icon_color(aiogram.types.forum_topic_created.ForumTe</pre>	opicCreateduttribute), 245
attribute), 121	id (aiogram.types.inline_query_result_mpeg4_gif.InlineQueryResultMpeg4
icon_custom_emoji_id	attribute), 248
(aiogram.methods.create_forum_topic.CreateFor attribute), 320	und faiogram.types.inline_query_result_photo.InlineQueryResultPhoto attribute), 249
icon_custom_emoji_id	id (aiogram.types.inline_query_result_venue.InlineQueryResultVenue
(aiogram.methods.edit_forum_topic.EditForumTe	
attribute), 328	id (aiogram.types.inline_query_result_video.InlineQueryResultVideo
icon_custom_emoji_id	attribute), 253
(aiogram.types.forum_topic.ForumTopic	id (aiogram.types.inline_query_result_voice.InlineQueryResultVoice
attribute), 120	attribute), 255
	id (aiogram.types.poll.Poll attribute), 198

id (aiogram.types.pre_checkout_query.PreCheckoutQuery		
attribute), 280 id (aiogram.types.shipping_option.ShippingOption at-	aiogram.types.inline_keyboard_markup), 127	
<i>tribute</i>), 281	InlineQuery (class in aiogram.types.inline_query), 216	
<pre>id (aiogram.types.shipping_query.ShippingQuery at- tribute), 282</pre>	InlineQueryResult (class in aiogram.types.inline_query_result), 218	
id (aiogram.types.story.Story attribute), 205 id (aiogram.types.user.User attribute), 207	InlineQueryResultArticle (class in aiogram.types.inline_query_result_article),	
id (aiogram.utils.web_app.WebAppChat attribute), 562	219	
id (aiogram.utils.web_app.WebAppUser attribute), 561	InlineQueryResultAudio (class in	
IDENTITY_CARD (aiogram.enums.encrypted_passport_elen attribute), 464	ment.Encry ptedPanspyptElinliew t_query_result_audio), 220	
IDR (aiogram.enums.currency.Currency attribute), 462	InlineQueryResultCachedAudio (class in	
ILS (<i>aiogram.enums.currency.Currency attribute</i>), 462 InaccessibleMessage (<i>class</i> in	aiogram.types.inline_query_result_cached_audio), 221	
aiogram.types.inaccessible_message), 125	InlineQueryResultCachedDocument (class in	
<pre>include_router() (aiogram.dispatcher.router.Router</pre>	aiogram.types.inline_query_result_cached_document), 223	,
<pre>include_routers() (aiogram.dispatcher.router.Router method), 475</pre>	<pre>InlineQueryResultCachedGif (class in</pre>	
inline_keyboard (aiogram.types.inline_keyboard_marki		
attribute), 127	InlineQueryResultCachedMpeg4Gif (class in	
<pre>inline_message_id(aiogram.methods.edit_message_cap attribute), 423</pre>	ption.EditMainsgramCappiaminline_query_result_cached_mpeg4_gif 227	[;]),
inline_message_id(aiogram.methods.edit_message_liv	e Indiane Quality Mesulge Cavelled Phiono (class in	
attribute), 425	aiogram.types.inline_query_result_cached_photo),	
<pre>inline_message_id(aiogram.methods.edit_message_me</pre>		
attribute), 427	InlineQueryResultCachedSticker (class in	
attribute), 428	ply_markupdidgn/desssygeeRipllyde/apkenpy_result_cached_sticker), 231	
attribute), 428 inline_message_id(aiogram.methods.edit_message_tex	231 ct.FaliiMeQueryResultCachedVideo (class in	
attribute), 428 inline_message_id(aiogram.methods.edit_message_tex attribute), 430	231 ct. Falinivle Query TResult Cached Video (class in aiogram.types.inline_query_result_cached_video),	
attribute), 428 inline_message_id(aiogram.methods.edit_message_tex attribute), 430 inline_message_id(aiogram.methods.get_game_high_s	231 ct.EnlindeQuageTResultCachedVideo (class in aiogram.types.inline_query_result_cached_video), ccores.GetGumeHighScores	
attribute), 428 inline_message_id(aiogram.methods.edit_message_tex attribute), 430 inline_message_id(aiogram.methods.get_game_high_s attribute), 438	231 ct.EnlinMeQueryResultCachedVideo (class in aiogram.types.inline_query_result_cached_video), ccores.GetG2meHighScores InlineQueryResultCachedVoice (class in	
attribute), 428 inline_message_id(aiogram.methods.edit_message_tex attribute), 430 inline_message_id(aiogram.methods.get_game_high_s attribute), 438 inline_message_id(aiogram.methods.set_game_score.S attribute), 441	231 ct.EnliiMeQuegeYRersultCachedVideo (class in aiogram.types.inline_query_result_cached_video), ccores.GetGameHighScores InlineQueryResultCachedVoice (class in SetGameScaniogram.types.inline_query_result_cached_voice), 236	
attribute), 428 inline_message_id(aiogram.methods.edit_message_tex attribute), 430 inline_message_id(aiogram.methods.get_game_high_s attribute), 438 inline_message_id(aiogram.methods.set_game_score.S attribute), 441 inline_message_id(aiogram.methods.stop_message_liv	231 ct.EnditiveQueryResultCachedVideo (class in aiogram.types.inline_query_result_cached_video), ccores.GetGameHighScores InlineQueryResultCachedVoice (class in SetGameScaniogram.types.inline_query_result_cached_voice), 236 ee_thkinieQueryResulgClointLactation (class in	
attribute), 428 inline_message_id(aiogram.methods.edit_message_tex attribute), 430 inline_message_id(aiogram.methods.get_game_high_s attribute), 438 inline_message_id(aiogram.methods.set_game_score.S attribute), 441 inline_message_id(aiogram.methods.stop_message_liv attribute), 432	231 ct.EnditMeQueryResultCachedVideo (class in aiogram.types.inline_query_result_cached_video), ccores.GetG2meHighScores InlineQueryResultCachedVoice (class in SetGameScaniogram.types.inline_query_result_cached_voice), 236 pe_LtkinieQuEroyResudgElcimeEacation (class in aiogram.types.inline_query_result_contact),	
attribute), 428 inline_message_id(aiogram.methods.edit_message_tex attribute), 430 inline_message_id(aiogram.methods.get_game_high_s attribute), 438 inline_message_id(aiogram.methods.set_game_score.S attribute), 441 inline_message_id(aiogram.methods.stop_message_liv attribute), 432 inline_message_id(aiogram.types.callback_query.Call	231 ct.EnditMeQuegrYRersultCachedVideo (class in aiogram.types.inline_query_result_cached_video), ccores.GetG2meHighScores InlineQueryResultCachedVoice (class in SetGameScaniogram.types.inline_query_result_cached_voice), 236 pe_thkinieQueryResultgElomeBacation (class in aiogram.types.inline_query_result_contact), backQuery 238	
attribute), 428 inline_message_id(aiogram.methods.edit_message_tex attribute), 430 inline_message_id(aiogram.methods.get_game_high_ss attribute), 438 inline_message_id(aiogram.methods.set_game_score.Ss attribute), 441 inline_message_id(aiogram.methods.stop_message_liv attribute), 432 inline_message_id(aiogram.types.callback_query.Call attribute), 28	231 ct.EnditiveQuegrYRersultCachedVideo (class in aiogram.types.inline_query_result_cached_video), ccores.GetG2meHighScores InlineQueryResultCachedVoice (class in SetGameScaniogram.types.inline_query_result_cached_voice), 236 ve_tkinieQu&royResultgeContbactation (class in aiogram.types.inline_query_result_contact), backQuery 238 InlineQueryResultDocument (class in	
attribute), 428 inline_message_id(aiogram.methods.edit_message_tex attribute), 430 inline_message_id(aiogram.methods.get_game_high_s attribute), 438 inline_message_id(aiogram.methods.set_game_score.S attribute), 441 inline_message_id(aiogram.methods.stop_message_liv attribute), 432 inline_message_id(aiogram.types.callback_query.Call attribute), 28 inline_message_id(aiogram.types.chosen_inline_result	231 ct.EnditiveQuegrYRersultCachedVideo (class in aiogram.types.inline_query_result_cached_video), ccores.GetG2meHighScores InlineQueryResultCachedVoice (class in SetGameScaniogram.types.inline_query_result_cached_voice), 236 ve_tkinieQu&royResultgeContbactation (class in aiogram.types.inline_query_result_contact), backQuery 238 InlineQueryResultDocument (class in	
attribute), 428 inline_message_id (aiogram.methods.edit_message_tex attribute), 430 inline_message_id (aiogram.methods.get_game_high_s attribute), 438 inline_message_id (aiogram.methods.set_game_score.S attribute), 441 inline_message_id (aiogram.methods.stop_message_liv attribute), 432 inline_message_id (aiogram.types.callback_query.Call attribute), 28 inline_message_id (aiogram.types.chosen_inline_result attribute), 216	231 ct.EnlintleQuegrYRersultCachedVideo (class in aiogram.types.inline_query_result_cached_video), ccores.GetGameHighScores InlineQueryResultCachedVoice (class in SetGameScariogram.types.inline_query_result_cached_voice), 236 re_ttkinieQueryResultGameLadation (class in aiogram.types.inline_query_result_contact), backQuery 238 InlineQueryResultDocument (class in t.ChosenInline@esultypes.inline_query_result_document), 239	
attribute), 428 inline_message_id(aiogram.methods.edit_message_tex attribute), 430 inline_message_id(aiogram.methods.get_game_high_s attribute), 438 inline_message_id(aiogram.methods.set_game_score.S attribute), 441 inline_message_id(aiogram.methods.stop_message_liv attribute), 432 inline_message_id(aiogram.types.callback_query.Call attribute), 28 inline_message_id(aiogram.types.chosen_inline_result	231 ct.EnlintleQuegrYRersultCachedVideo (class in aiogram.types.inline_query_result_cached_video), ccores.GetGameHighScores InlineQueryResultCachedVoice (class in SetGameScariogram.types.inline_query_result_cached_voice), 236 re_ttkinieQueryResultGameLadation (class in aiogram.types.inline_query_result_contact), backQuery 238 InlineQueryResultDocument (class in t.ChosenInline@esultypes.inline_query_result_document), 239	
attribute), 428 inline_message_id(aiogram.methods.edit_message_tex attribute), 430 inline_message_id(aiogram.methods.get_game_high_s attribute), 438 inline_message_id(aiogram.methods.set_game_score.S attribute), 441 inline_message_id(aiogram.methods.stop_message_liv attribute), 432 inline_message_id(aiogram.types.callback_query.Call attribute), 28 inline_message_id(aiogram.types.chosen_inline_result attribute), 216 inline_message_id(aiogram.types.sent_web_app_mess	231 ct.FaliiMeQueryResultCachedVideo (class in aiogram.types.inline_query_result_cached_video), ccores.GetG2meHighScores InlineQueryResultCachedVoice (class in SetGameScaniogram.types.inline_query_result_cached_voice), 236 re_ItkinieQueryResultCachedVoice (class in aiogram.types.inline_query_result_cached_voice), 236 re_ItkinieQueryResultGameLadation (class in aiogram.types.inline_query_result_contact), backQuery 238 InlineQueryResultDocument (class in t.ChosenInlineResultypes.inline_query_result_document), 239 agenSintQueryPResultGame (class in	
attribute), 428 inline_message_id (aiogram.methods.edit_message_tex attribute), 430 inline_message_id (aiogram.methods.get_game_high_ss attribute), 438 inline_message_id (aiogram.methods.set_game_score.Ss attribute), 441 inline_message_id (aiogram.methods.stop_message_liv attribute), 432 inline_message_id (aiogram.types.callback_query.Call attribute), 28 inline_message_id (aiogram.types.chosen_inline_result attribute), 216 inline_message_id (aiogram.types.sent_web_app_mess attribute), 263 INLINE_QUERY (aiogram.enums.update_type.UpdateType attribute), 470	231 ct.EnlinMeQueryResultCachedVideo (class in aiogram.types.inline_query_result_cached_video), ccores.GetG2meHighScores InlineQueryResultCachedVoice (class in SetGameScaniogram.types.inline_query_result_cached_voice), 236 ve_thkinieQu&toyResultGameBactation (class in aiogram.types.inline_query_result_contact), backQuery 238 InlineQueryResultDocument (class in t.ChosenInline@esulttypes.inline_query_result_document), 239 a&mSint&veAryResultGame (class in aiogram.types.inline_query_result_document), 241 InlineQueryResultGif (class in	
attribute), 428 inline_message_id(aiogram.methods.edit_message_tex attribute), 430 inline_message_id(aiogram.methods.get_game_high_s attribute), 438 inline_message_id(aiogram.methods.set_game_score.S attribute), 441 inline_message_id(aiogram.methods.stop_message_liv attribute), 432 inline_message_id(aiogram.types.callback_query.Call attribute), 28 inline_message_id(aiogram.types.chosen_inline_result attribute), 216 inline_message_id(aiogram.types.sent_web_app_mess attribute), 263 INLINE_QUERY(aiogram.enums.update_type.UpdateType attribute), 470 inline_query(aiogram.types.update.Update attribute),	231 ct.EnlintMeQuegrYRersultCachedVideo (class in aiogram.types.inline_query_result_cached_video), ccores.GetG2meHighScores InlineQueryResultCachedVoice (class in SetGameScaniogram.types.inline_query_result_cached_voice), 236 ve_ttkinteQu&royRessulgConteLatation (class in aiogram.types.inline_query_result_contact), backQuery 238 InlineQueryResultDocument (class in t.ChosenInlineQueryResultDocument (class in t.ChosenInlineQueryResultDocument (class in aiogram.types.inline_query_result_document), 239 a&mSintQueleAppRefsult&ame (class in aiogram.types.inline_query_result_game), 241 InlineQueryResultGif (class in aiogram.types.inline_query_result_gif), 242	
attribute), 428 inline_message_id (aiogram.methods.edit_message_tex attribute), 430 inline_message_id (aiogram.methods.get_game_high_s attribute), 438 inline_message_id (aiogram.methods.set_game_score.S attribute), 441 inline_message_id (aiogram.methods.stop_message_liv attribute), 432 inline_message_id (aiogram.types.callback_query.Call attribute), 28 inline_message_id (aiogram.types.chosen_inline_result attribute), 28 inline_message_id (aiogram.types.chosen_inline_result attribute), 216 inline_message_id (aiogram.types.sent_web_app_mess attribute), 263 INLINE_QUERY (aiogram.enums.update_type.UpdateType attribute), 470 inline_query (aiogram.types.update.Update attribute), 285	231 ct.EnlintMeQueryResultCachedVideo (class in aiogram.types.inline_query_result_cached_video), ccores.GetGameHighScores InlineQueryResultCachedVoice (class in SetGameScaniogram.types.inline_query_result_cached_voice), 236 re_ItkinieQuinyResultGameLadation (class in aiogram.types.inline_query_result_contact), backQuery 238 InlineQueryResultDocument (class in t.ChosenInlinieResultypes.inline_query_result_document), 239 agenSintQueryPMesultGiame (class in aiogram.types.inline_query_result_game), 241 InlineQueryResultGif (class in aiogram.types.inline_query_result_gif), 242 InlineQueryResultLocation (class in	
attribute), 428 inline_message_id (aiogram.methods.edit_message_tex attribute), 430 inline_message_id (aiogram.methods.get_game_high_s attribute), 438 inline_message_id (aiogram.methods.set_game_score.S attribute), 441 inline_message_id (aiogram.methods.stop_message_liv attribute), 432 inline_message_id (aiogram.types.callback_query.Call attribute), 28 inline_message_id (aiogram.types.chosen_inline_result attribute), 216 inline_message_id (aiogram.types.sent_web_app_mess attribute), 263 INLINE_QUERY (aiogram.enums.update_type.UpdateType attribute), 470 inline_query (aiogram.types.update.Update attribute), 285 inline_query_id (aiogram.methods.answer_inline_query attribute), 434	231 ct.FnliiMeQueryResultCachedVideo (class in aiogram.types.inline_query_result_cached_video), ccores.GetG2meHighScores InlineQueryResultCachedVoice (class in SetGameScaniogram.types.inline_query_result_cached_voice), 236 ve_thtimeQu&royResultGamebacation (class in aiogram.types.inline_query_result_contact), backQuery 238 InlineQueryResultDocument (class in t.ChosenInline@esultypes.inline_query_result_document), 239 a&mSintQueryResultDocument (class in aiogram.types.inline_query_result_document), 239 a&mSintQueryResultGif (class in aiogram.types.inline_query_result_game), 241 InlineQueryResultGif (class in aiogram.types.inline_query_result_gif), 242 InlineQueryResultLocation (class in ty.AnswerInline@uurytypes.inline_query_result_location), 244	
attribute), 428 inline_message_id (aiogram.methods.edit_message_tex attribute), 430 inline_message_id (aiogram.methods.get_game_high_s attribute), 438 inline_message_id (aiogram.methods.set_game_score.S attribute), 441 inline_message_id (aiogram.methods.stop_message_liv attribute), 432 inline_message_id (aiogram.types.callback_query.Call attribute), 28 inline_message_id (aiogram.types.chosen_inline_result attribute), 216 inline_message_id (aiogram.types.sent_web_app_mess attribute), 263 INLINE_QUERY (aiogram.enums.update_type.UpdateType attribute), 470 inline_query (aiogram.types.update.Update attribute), 285 inline_query_id (aiogram.methods.answer_inline_quer attribute), 434 InlineKeyboardBuilder (class in	231 ct.EnliiMeQueryResultCachedVideo (class in aiogram.types.inline_query_result_cached_video), ccores.GetG2meHighScores InlineQueryResultCachedVoice (class in SetGameScaniogram.types.inline_query_result_cached_voice), 236 ve_ttkinieQu&tryRfssslg&Comthatation (class in aiogram.types.inline_query_result_contact), backQuery 238 InlineQueryResultDocument (class in t.ChosenInline@esultypes.inline_query_result_document), 239 a&rSintQuefypRfssslg&ame (class in aiogram.types.inline_query_result_document), 241 InlineQueryResultGif (class in aiogram.types.inline_query_result_game), 241 InlineQueryResultLocation (class in y.AnswerInline@uuryppes.inline_query_result_location), 244 InlineQueryResultMpeg4Gif (class in	
attribute), 428 inline_message_id (aiogram.methods.edit_message_tex attribute), 430 inline_message_id (aiogram.methods.get_game_high_ss attribute), 438 inline_message_id (aiogram.methods.set_game_score.S attribute), 441 inline_message_id (aiogram.methods.stop_message_liv attribute), 432 inline_message_id (aiogram.types.callback_query.Call attribute), 28 inline_message_id (aiogram.types.chosen_inline_result attribute), 28 inline_message_id (aiogram.types.chosen_inline_result attribute), 216 inline_message_id (aiogram.types.sent_web_app_mess attribute), 263 INLINE_QUERY (aiogram.enums.update_type.UpdateType attribute), 470 inline_query_id (aiogram.methods.answer_inline_quer attribute), 434 InlineKeyboardBuilder (class in aiogram.utils.keyboard), 549	231 ct.FnliiMeQueryResultCachedVideo (class in aiogram.types.inline_query_result_cached_video), ccores.GetG2meHighScores InlineQueryResultCachedVoice (class in SetGameScariogram.types.inline_query_result_cached_voice), 236 ve_thkinieQu&tryRfsssldgeComthatation (class in aiogram.types.inline_query_result_contact), backQuery 238 InlineQueryResultDocument (class in t.ChosenInlineQtesultypes.inline_query_result_document), 239 a&TSintQteAtypRfsssldgeame (class in aiogram.types.inline_query_result_document), 241 InlineQueryResultGif (class in aiogram.types.inline_query_result_gif), 242 InlineQueryResultLocation (class in ty.AnswerInlineQuertypes.inline_query_result_location), 244 InlineQueryResultMpeg4Gif (class in aiogram.types.inline_query_result_location), 244	
attribute), 428 inline_message_id (aiogram.methods.edit_message_tex attribute), 430 inline_message_id (aiogram.methods.get_game_high_s attribute), 438 inline_message_id (aiogram.methods.set_game_score.S attribute), 441 inline_message_id (aiogram.methods.stop_message_liv attribute), 432 inline_message_id (aiogram.types.callback_query.Call attribute), 28 inline_message_id (aiogram.types.chosen_inline_result attribute), 216 inline_message_id (aiogram.types.sent_web_app_mess attribute), 263 INLINE_QUERY (aiogram.enums.update_type.UpdateType attribute), 470 inline_query (aiogram.types.update.Update attribute), 285 inline_query_id (aiogram.methods.answer_inline_quer attribute), 434 InlineKeyboardBuilder (class in	231 ct.EnliiMeQueryResultCachedVideo (class in aiogram.types.inline_query_result_cached_video), ccores.GetG2meHighScores InlineQueryResultCachedVoice (class in SetGameScaniogram.types.inline_query_result_cached_voice), 236 ve_ttkinieQu&tryRfssslg&Comthatation (class in aiogram.types.inline_query_result_contact), backQuery 238 InlineQueryResultDocument (class in t.ChosenInline@esultypes.inline_query_result_document), 239 a&rSintQuefypRfssslg&ame (class in aiogram.types.inline_query_result_document), 241 InlineQueryResultGif (class in aiogram.types.inline_query_result_game), 241 InlineQueryResultLocation (class in y.AnswerInline@uuryppes.inline_query_result_location), 244 InlineQueryResultMpeg4Gif (class in	

<i>aiogram.types.inline_query_result_photo</i>), 249	(aiogram.types.inline_query_result_contact.InlineQueryResultCo
	attribute), 239
aiogram.types.inline_query_results_button),	nput_message_content (aiogram.types.inline_query_result_document.InlineQueryResult)
256	attribute), 241
	nput_message_content
aiogram.enums.inline_query_result_type), 465	(aiogram.types.inline_query_result_gif.InlineQueryResultGif attribute), 244
	nput_message_content
aiogram.types.inline_query_result_venue), 251	(aiogram.types.inline_query_result_location.InlineQueryResultLa attribute), 246
	nput_message_content
aiogram.types.inline_query_result_video), 252	(aiogram.types.inline_query_result_mpeg4_gif.InlineQueryResult attribute), 248
InlineQueryResultVoice (class in in	nput_message_content
aiogram.types.inline_query_result_voice), 254	(aiogram.types.inline_query_result_photo.InlineQueryResultPhot attribute), 250
	nput_message_content
(aiogram.types.force_reply.ForceReply at- tribute), 120	(aiogram.types.inline_query_result_venue.InlineQueryResultVenu attribute), 252
input_field_placeholder in	nput_message_content
(aiogram.types.reply_keyboard_markup.ReplyKeybo attribute), 202	pardMar (aip gram.types.inline_query_result_video.InlineQueryResultVideo attribute), 254
input_message_content in	nput_message_content
(aiogram.types.inline_query_result_article.InlineQu attribute), 219	ueryResu (ailogicu e.types.inline_query_result_voice.InlineQueryResultVoice attribute), 256
	nputContactMessageContent (class in
(aiogram.types.inline_query_result_audio.InlineQu attribute), 221	eryResul tiaglio m.types.input_contact_message_content), 257
	nputFile (class in aiogram.types.input_file), 127
(aiogram.types.inline_query_result_cached_audioII	
attribute), 223	aiogram.types.input_invoice_message_content),
input_message_content	257
(aiogram.types.inline_query_result_cached_docur h	apukilipæQuionyRessbaGaCbartDencument (class in
attribute), 225	aiogram.types.input_location_message_content),
input_message_content	260
	npQuMecRes(dlGxschedGigram.types.input_media), 128
	nputMediaAnimation (class in
input_message_content	aiogram.types.input_media_animation), 128
(aiogram.types.inline_query_result_cached_mpeg	
attribute), 229	aiogram.types.input_media_audio), 129 nputMediaDocument (class in
	nputMediaDocument (class in nlineQu enivResult©pekidpHtom edia_document), 130
	nputMediaPhoto (class in
input_message_content	aiogram.types.input_media_photo), 132
(aiogram.types.inline_query_result_cached_sticke	
attribute), 233	aiogram.enums.input_media_type), 465
	nputMediaVideo (class in
(aiogram.types.inline_query_result_cached_video.I	
	nputMessageContent (class in
input_message_content	aiogram.types.input_message_content), 261
(aiogram.types.inline_query_result_cached_voice 1 attribute), 237	n ļint Otvir iškea su lt Causheid Voüce gram.types.input_sticker), 263
input_message_content In	nputTextMessageContent (class in

aiogram.types.input_text_message_content), 261	(aiogram.types.message.Message attribute), 145
InputVenueMessageContent (class in aiogram.types.input_venue_message_content),	is_big(aiogram.methods.set_message_reaction.SetMessageReaction attribute), 406
262	is_bot (aiogram.types.user.User attribute), 207
INR (aiogram.enums.currency.Currency attribute), 462	is_bot (alogram.utils.web_app.WebAppUser attribute),
INTERNAL_PASSPORT (aiogram.enums.encrypted_passport	
attribute), 464	is_closed (aiogram.methods.send_poll.SendPoll
invite_link (aiogram.methods.edit_chat_invite_link.Edi	
attribute), 327	is_closed (aiogram.types.poll.Poll attribute), 198
	ReisskelCisalDeticlingram.types.link_preview_options.LinkPreviewOptions
attribute), 360	attribute), 139
invite_link (aiogram.types.chat.Chat attribute), 31	is_enabled(aiogram.types.business_connection.BusinessConnection
invite_link (aiogram.types.chat_invite_link.ChatInviteL	
attribute), 49	<pre>is_flexible(aiogram.methods.create_invoice_link.CreateInvoiceLink</pre>
<pre>invite_link(aiogram.types.chat_join_request.ChatJoin]</pre>	
attribute), 50	is_flexible(aiogram.methods.send_invoice.SendInvoice
<pre>invite_link(aiogram.types.chat_member_updated.Chat</pre>	
attribute), 94	is_flexible(aiogram.types.input_invoice_message_content.InputInvoice.
INVOICE (<i>aiogram.enums.content_type.ContentType at-</i>	attribute), 260
tribute), 460	is_forum (aiogram.types.chat.Chat attribute), 30
invoice (aiogram.types.external_reply_info.ExternalReply_	yhte_from_offline (aiogram.types.message.Message at-
attribute), 118	tribute), 146
invoice (aiogram.types.message.Message attribute), 148	<pre>is_local (aiogram.client.telegram.TelegramAPIServer</pre>
Invoice (class in aiogram.types.invoice), 278	attribute), 13
<pre>invoice_payload(aiogram.types.pre_checkout_query.Pr attribute), 280</pre>	re LstenkavatQ uerstaiogram.types.text_quote.TextQuote at- tribute), 207
<pre>invoice_payload(aiogram.types.shipping_query.Shippin</pre>	n g@umember (aiogram.types.chat_member_restricted.ChatMemberRestrictea attribute), 92
<pre>invoice_payload(aiogram.types.successful_payment.Su</pre>	c àcssfidPayssen t (aiogram.types.reply_keyboard_markup.ReplyKeyboardM attribute), 202
<pre>ip_address (aiogram.methods.set_webhook.SetWebhook</pre>	<pre>is_personal (aiogram.methods.answer_inline_query.AnswerInlineQuery attribute), 434</pre>
<pre>ip_address (aiogram.types.webhook_info.WebhookInfo</pre>	
attribute), 286	is_premium (aiogram.utils.web_app.WebAppUser
<pre>ip_filter_middleware() (in module</pre>	attribute), 561
aiogram.webhook.aiohttp_server), 502	<pre>is_primary(aiogram.types.chat_invite_link.ChatInviteLink</pre>
IPFilter (class in aiogram.webhook.security), 502	attribute), 49
<pre>is_animated (aiogram.types.sticker.Sticker attribute),</pre>	<pre>is_revoked(aiogram.types.chat_invite_link.ChatInviteLink attribute), 49</pre>
is_animated (aiogram.types.sticker_set.StickerSet at-	
tribute), 267	attribute), 145
	PicsnuteClaitMed(locogram.types.chat_boost_source_giveaway.ChatBoostSou attribute), 48
is_anonymous (aiogram.methods.send_poll.SendPoll at-	
tribute), 385	is_video (aiogram.types.sticker_set.StickerSet at-
is_anonymous (<i>aiogram.types.chat_administrator_rights</i> .	
attribute), 44	ISK (aiogram.enums.currency.Currency attribute), 462
	or IIIA11 Memberskubmenistristnessage_entity_type.MessageEntityType
attribute), 89	attribute), 467
is_anonymous (aiogram.types.chat_member_owner.Chatl attribute), 91	
is_anonymous (<i>aiogram.types.poll.Poll attribute</i>), 198	J
is_automatic_forward	JMD (aiogram.enums.currency.Currency attribute), 462
	and framenums, currency, currency unifolder, roz

<pre>join_by_request (aiogram.types.chat.Chat attribute), 31</pre>	<pre>language_code (aiogram.methods.get_my_short_description.GetMyShorth attribute), 347</pre>
join_to_send_messages (<i>aiogram.types.chat.Chat at-</i> <i>tribute</i>), 31	language_code (<i>aiogram.methods.set_my_commands.SetMyCommands</i> <i>attribute</i>), 407
JPY (aiogram.enums.currency.Currency attribute), 462	language_code (<i>aiogram.methods.set_my_description.SetMyDescription</i> <i>attribute</i>), 410
К	language_code (aiogram.methods.set_my_name.SetMyName
KES (aiogram.enums.currency.Currency attribute), 462	attribute), 411
keyboard (aiogram.types.reply_keyboard_markup.ReplyK attribute), 202	e]pogyager/code (aiogram.methods.set_my_short_description.SetMyShortL attribute), 412
KeyboardButton (class in	language_code (<i>aiogram.types.user.User attribute</i>), 207
aiogram.types.keyboard_button), 134	language_code (aiogram.utils.web_app.WebAppUser
KeyboardButtonPollType (class in	attribute), 561
aiogram.types.keyboard_button_poll_type), 135	<pre>last_error_date(aiogram.types.webhook_info.WebhookInfo</pre>
KeyboardButtonPollTypeType (class in	<pre>last_error_message(aiogram.types.webhook_info.WebhookInfo</pre>
aiogram.enums.keyboard_button_poll_type_type), attribute), 286
466	<pre>last_name(aiogram.methods.send_contact.SendContact</pre>
KeyboardButtonRequestChat (class in	attribute), 368
aiogram.types.keyboard_button_request_chat), 135	<pre>last_name (aiogram.types.chat.Chat attribute), 30 last_name (aiogram.types.contact.Contact attribute),</pre>
KeyboardButtonRequestUser (class in	115
aiogram.types.keyboard_button_request_user), 137	<pre>last_name (aiogram.types.inline_query_result_contact.InlineQueryResult attribute), 238</pre>
KeyboardButtonRequestUsers (class in	<pre>last_name(aiogram.types.input_contact_message_content.InputContactM</pre>
aiogram.types.keyboard_button_request_users),	attribute), 257 last_name (aiogram.types.shared_user.SharedUser at-
138 Key Duilder (alass in sizensus fam stears radio) 516	tribute), 205
KeyBuilder (class in aiogram.fsm.storage.redis), 516 keywords (aiogram.methods.set_sticker_keywords.SetStick	
attribute), 301	last_name (aiogram.utils.web_app.WebAppUser at-
keywords (aiogram.types.input_sticker.InputSticker at-	tribute), 561
tribute), 264	last_synchronization_error_date
KGS (aiogram.enums.currency.Currency attribute), 462	(aiogram.types.webhook_info.WebhookInfo
KICKED (aiogram.enums.chat_member_status.ChatMember	
attribute), 458	latitude (aiogram.methods.edit_message_live_location.EditMessageLivel
KRW (aiogram.enums.currency.Currency attribute), 462	attribute), 425
KZT (aiogram.enums.currency.Currency attribute), 462	<pre>latitude (aiogram.methods.send_location.SendLocation</pre>
L	latitude (aiogram.methods.send_venue.SendVenue at-
label (aiogram.types.labeled_price.LabeledPrice	tribute), 387
attribute), 279	<pre>latitude (aiogram.types.inline_query_result_location.InlineQueryResultL attribute), 245</pre>
LabeledPrice (class in aiogram.types.labeled_price), 279	latitude (aiogram.types.inline_query_result_venue.InlineQueryResultVen
language (aiogram.types.message_entity.MessageEntity	attribute), 251
attribute), 192	latitude (aiogram.types.input_location_message_content.InputLocationM
language_code (aiogram.methods.delete_my_commands attribute), 326	latitude (aiogram.types.input_venue_message_content.InputVenueMessage
language_code (aiogram.methods.get_my_commands.Get attribute), 343	
language_code (aiogram.methods.get_my_description.Ge	thyDescription
attribute), 345 language_code (aiogram.methods.get_my_name.GetMyN	ane leave() (alogram.types.chat.Chat method), 37
attribute), 346	LeaveChat (class in aiogram.methods.leave chat), 350

LEFT (aiogram.enums.chat_member_status.ChatMemberStatus)	atus attribute), 216
attribute), 458	location (aiogram.types.external_reply_info.ExternalReplyInfo
LEFT_CHAT_MEMBER (aiogram.enums.content_type.Content	
attribute), 460	<pre>location (aiogram.types.inline_query.InlineQuery at-</pre>
<pre>left_chat_member (aiogram.types.message.Message</pre>	tribute), 217
attribute), 147	location (aiogram.types.message.Message attribute),
<pre>length(aiogram.methods.send_video_note.SendVideoNote</pre>	
attribute), 393	location (<i>aiogram.types.venue.Venue attribute</i>), 210
<pre>length (aiogram.types.message_entity.MessageEntity at- tribute), 192</pre>	Location (class in aiogram.types.location), 140 login_url (aiogram.types.inline_keyboard_button.InlineKeyboardButton
length (aiogram.types.video_note.VideoNote attribute),	attribute), 126
213	LoginUrl (<i>class in aiogram.types.login_url</i>), 140
limit (aiogram.methods.get_updates.GetUpdates at-	
tribute), 452	longitude (aiogram.methods.edit_message_live_location.EditMessageLiv
limit (aiogram.methods.get_user_profile_photos.GetUser_	
attribute), 349	longitude (aiogram.methods.send_location.SendLocation
link_preview_options	attribute), 375
(aiogram.methods.edit_message_text.EditMessag	ellengitude (aiogram.methods.send_venue.SendVenue at-
attribute), 430	tribute), 387
link_preview_options	$\verb longitude (a iogram.types.inline_query_result_location.InlineQueryResul$
(aiogram.methods.send_message.SendMessage	attribute), 245
attribute), 380	longitude (aiogram.types.inline_query_result_venue.InlineQueryResultVenue)
link_preview_options	attribute), 251
(aiogram.types.external_reply_info.ExternalRepl_ attribute), 117	ylpgngitude (aiogram.types.input_location_message_content.InputLocation attribute), 260
link_preview_options	longitude (aiogram.types.input_venue_message_content.InputVenueMess
(aiogram.types.input_text_message_content.Inpu	
attribute), 262	longitude (aiogram.types.location.Location attribute),
link_preview_options	140
(aiogram.types.message.Message attribute), 146	Μ
<pre>linked_chat_id (aiogram.types.chat.Chat attribute),</pre>	MAD (aiogram.enums.currency.Currency attribute), 462
32	<pre>magic_data (aiogram.filters.magic_data.MagicData at-</pre>
LinkPreviewOptions (class in	tribute), 492
aiogram.types.link_preview_options), 139 live_period(aiogram.methods.send_location.SendLocation)	magic_result (aiogram.filters.command.CommandObject ion attribute), 486
attribute), 375	MagicData (<i>class in aiogram.filters.magic_data</i>), 492
	.haline_QueryBers()/Lawyram.client.session.base.BaseSession
attribute), 246	method), 13
<pre>live_period(aiogram.types.input_location_message_com</pre>	
attribute), 260	tribute), 468
live_period (aiogram.types.location.Location at- tribute), 140	MARKDOWN_V2 (aiogram.enums.parse_mode.ParseMode attribute), 468
LKR (aiogram.enums.currency.Currency attribute), 462	MASK (aiogram.enums.sticker_type.StickerType attribute),
LOCATION (aiogram.enums.content_type.ContentType at-	469
tribute), 459	mask_position (aiogram.methods.set_sticker_mask_position.SetStickerM
LOCATION (aiogram.enums.inline_query_result_type.Inline attribute), 465	
location (aiogram.types.business_location.BusinessLocat	mask_position (aiogram.types.input_sticker.InputSticker
attribute), 26	
location (aiogram.types.chat.Chat attribute), 32	mask_position (<i>aiogram.types.sticker.Sticker attribute</i>), 265
location (alogram.types.chat_location.ChatLocation	MaskPosition (class in aiogram.types.mask_position),
attribute), 87	264
location (aiogram.types.chosen_inline_result.ChosenInlin	

MaskPositionPoint (class in		tribute), 486
aiogram.enums.mask_position_point), 466	mention	_html() (aiogram.types.user.User method),
<pre>max_connections(aiogram.methods.set_webhook.SetWeb</pre>	bhook	208
attribute), 454 max_connections (aiogram.types.webhook_info.Webhool		_markdown() (aiogram.types.user.User method), 208
attribute), 287		ed (aiogram.filters.command.CommandObject
<pre>max_quantity (aiogram.types.keyboard_button_request_u</pre>		
attribute), 138		ltton (aiogram.methods.set_chat_menu_button.SetChatMenuButto
<pre>max_tip_amount (aiogram.methods.create_invoice_link.C</pre>		
attribute), 445 max_tip_amount (aiogram.methods.send_invoice.SendInv		ton (class in aiogram.types.menu_button), 141 tonCommands (class in
attribute), 448		aiogram.types.menu_button_commands),
<pre>max_tip_amount (aiogram.types.input_invoice_message_</pre>	content.In	
attribute), 259		tonDefault (class in
MaybeInaccessibleMessage (class in		aiogram.types.menu_button_default), 142
aiogram.types.maybe_inaccessible_message),	MenuBut	
141	Manag	aiogram.enums.menu_button_type), 467
<pre>md_text (aiogram.types.message.Message property), 150</pre>	мепивит	tonWebApp (class in aiogram.types.menu_button_web_app), 143
MDL (aiogram.enums.currency.Currency attribute), 462	MESSAGE	(aiogram.enums.update_type.UpdateType at-
media (aiogram.methods.edit_message_media.EditMessag		tribute), 470
attribute), 427		$(aiogram.handlers.callback_query.CallbackQueryHandler)$
$\texttt{media}(a iogram. methods. send_media_group. SendMediaGiagnam. methods. send_media_group. SendMediaGiagnam. sendMedia$		property), 544
attribute), 377		(aiogram.types.business_intro.BusinessIntro at-
<pre>media(aiogram.types.input_media_animation.InputMedia attribute), 128</pre>		e (ribute), 25 e (aiogram.types.callback_query.CallbackQuery
media (aiogram.types.input_media_audio.InputMediaAudi		attribute), 28
attribute), 130		(aiogram.types.passport_element_error_data_field.PassportEleme
<pre>media(aiogram.types.input_media_document.InputMedia)</pre>	Document	attribute), 271
attribute), 131		e (aiogram.types.passport_element_error_file.PassportElementErro
media (aiogram.types.input_media_photo.InputMediaPhot		attribute), 271
attribute), 132 media(aiogram.types.input_media_video.InputMediaVideo		<pre>(aiogram.types.passport_element_error_files.PassportElementErr attribute), 272</pre>
attribute), 133		airroule), 272 (aiogram.types.passport_element_error_front_side.PassportEleme
media_group_id (aiogram.types.message.Message at-	meeeuge	attribute), 273
tribute), 146	message	(aiogram.types.passport_element_error_reverse_side.PassportEle
MediaGroupBuilder (class in		attribute), 274
aiogram.utils.media_group), 573	-	(aiogram.types.passport_element_error_selfie.PassportElementEr
MEMBER (aiogram.enums.chat_member_status.ChatMember_ attribute), 458		attribute), 275 (aiogram.types.passport_element_error_translation_file.Passport.
member_limit (aiogram.methods.create_chat_invite_link.		
attribute), 319		(aiogram.types.passport_element_error_translation_files.Passpor
<pre>member_limit(aiogram.methods.edit_chat_invite_link.Edit_chat_invite_link.edit_chat_</pre>		
attribute), 327		e (aiogram.types.passport_element_error_unspecified.PassportElen
member_limit (<i>aiogram.types.chat_invite_link.ChatInvite</i>		attribute), 277
<i>attribute</i>), 50 member_status_changed		e (aiogram.types.update.Update attribute), 284 e (class in aiogram.types.message), 143
(aiogram.filters.chat_member_updated.ChatMen		
attribute), 487		attribute), 32
MemoryStorage (class in aiogram.fsm.storage.memory),	message	_auto_delete_time
515		(aiogram.types.message_auto_delete_timer_changed.MessageAut
MENTION (aiogram.enums.message_entity_type.MessageEn		attribute), 191
attribute), 467 mention (aiogram.filters.command.CommandObject at-	MESSAGE	C_AUTO_DELETE_TIMER_CHANGED (aiogram.enums.content_type.ContentType
meneron (mogram.jmers.commana.commana00jeci al-		(anogram.enums.comem_type.comenttype

mossago	<pre>attribute), 460 _auto_delete_timer_changed</pre>		_ids(aiogram.methods.forward_messages.ForwardMessages attribute), 333
messaye	(aiogram.types.message.Message attribute),	message_	_ids(aiogram.types.business_messages_deleted.BusinessMessage
message		MESSAGE_	attribute), 26 _REACTION (aiogram.enums.update_type.UpdateType
message	attribute), 315 _id(aiogram.methods.delete_message.DeleteMess	me ssage_	
message	attribute), 421 _id(aiogram.methods.edit_message_caption.EditM		tribute), 285 gREAGTION_COUNT
message	attribute), 423 _id(aiogram.methods.edit_message_live_location		(aiogram.enums.update_type.UpdateType avaiintPoAition
-	attribute), 425	message_	_reaction_count
	_id(aiogram.methods.edit_message_media.EditM attribute), 427	_	285
message	_id(aiogram.methods.edit_message_reply_markup attribute), 428		stagaR4pliolyharkutypes.input_text_message_content.InputTextMessa attribute), 262
message	_id(aiogram.methods.edit_message_text.EditMess attribute), 430	-	_thread_id (aiogram.methods.close_forum_topic.CloseForumTop attribute), 313
message		Messag ge_	_thread_id (aiogram.methods.copy_message.CopyMessage attribute), 315
message		mæss <i>bl</i> ge/	Shires d_id(aiogram.methods.copy_messages.CopyMessages attribute), 317
message		Message_	_thread_id (aiogram.methods.delete_forum_topic.DeleteForumT attribute), 324
message		mes sage_	_thread_id(aiogram.methods.edit_forum_topic.EditForumTopic attribute), 328
message		messalle	adribute), 328 attribute), 328 attribute), 332
message			satyaliseLicktaiongram.methods.forward_messages.ForwardMessa
	attribute), 431		attribute), 333
message	_1d (aiogram.methods.stop_poll.StopPoll attribute), 433		_thread_id (aiogram.methods.reopen_forum_topic.ReopenForum attribute), 356
message	_id(aiogram.methods.unpin_chat_message.Unpin attribute), 419	-	attribute), 361
message		npolska konge_	_thread_id(aiogram.methods.send_audio.SendAudio attribute), 364
message		nilels stage (<pre>utility action (aiogram.methods.send_chat_action.SendChatAction attribute), 367</pre>
message		message_	_thread_id (aiogram.methods.send_contact.SendContact attribute), 368
message		message_	_thread_id (aiogram.methods.send_dice.SendDice attribute), 370
message		mæssæigėn	Chargend_id (aiogram.methods.send_document.SendDocument attribute), 373
message		melskægea	ga RaeadioinC(aingl/phaneet hods.send_game.SendGame attribute), 439
message			ctinute,,439
maccago	attribute), 196		attribute), 448
	attribute), 203		_thread_id (aiogram.methods.send_location.SendLocation attribute), 375
message	_ids (aiogram.methods.copy_messages.CopyMess attribute), 317		_thread_id(aiogram.methods.send_media_group.SendMediaGraattribute), 378
message		message_	_thread_id (aiogram.methods.send_message.SendMessage attribute), 380

	SendPl	Tag RATE_TO_CHAT_ID (aiogram.enums.content_type.ContentType
attribute), 382		attribute), 460
<pre>message_thread_id(aiogram.methods.send_poll.Ser attribute), 385</pre>	ndPoli	higrate_to_chat_id (aiogram.types.message.Message attribute), 148
	Sonds	uinforme , 146 uinforme te_to_chat_id(aiogram.types.response_parameters.ResponsePara
attribute), 297	Senus	attribute), 204
	SendVi	mime_type (aiogram.types.animation.Animation at-
attribute), 387		tribute), 18
message_thread_id(aiogram.methods.send_video.S	SendVi	
attribute), 390		nime_type (aiogram.types.document.Document at-
<pre>message_thread_id(aiogram.methods.send_video_r</pre>		
attribute), 393		nime_type (aiogram.types.inline_query_result_document.InlineQueryResu
<pre>message_thread_id(aiogram.methods.send_voice.Set)</pre>		
attribute), 396		nime_type(aiogram.types.inline_query_result_video.InlineQueryResultVideo.
<pre>message_thread_id(aiogram.methods.unpin_all_for</pre>		
attribute), 417		nime_type (aiogram.types.video.Video attribute), 211
<pre>message_thread_id(aiogram.types.forum_topic.For</pre>	rumTog	nime_type (aiogram.types.voice.Voice attribute), 214
attribute), 120	1	INT (aiogram.enums.currency.Currency attribute), 462
<pre>message_thread_id (aiogram.types.message.Messa</pre>	age 1	nodel_computed_fields
attribute), 145		(aiogram.filters.callback_data.CallbackData
MessageAutoDeleteTimerChanged (class	in	attribute), 493
aiogram.types.message_auto_delete_timer_c	chang e	nd)del_computed_fields
191		(aiogram.methods.add_sticker_to_set.AddStickerToSet
MessageEntity (class	in	attribute), 289
aiogram.types.message_entity), 192	r	nodel_computed_fields
MessageEntityType (class	in	$(a iogram.methods.answer_callback_query.AnswerCallbackQuery.answ$
aiogram.enums.message_entity_type), 467		attribute), 307
MessageId (class in aiogram.types.message_id), 193		nodel_computed_fields
MessageOrigin (class	in	(aiogram.methods.answer_inline_query.AnswerInlineQuery
aiogram.types.message_origin), 193		attribute), 434
MessageOriginChannel (class	in 1	nodel_computed_fields
aiogram.types.message_origin_channel),		(aiogram.methods.answer_pre_checkout_query.AnswerPreChecko
193		attribute), 442
MessageOriginChat (class	<i>in</i> r	<pre>nodel_computed_fields</pre>
aiogram.types.message_origin_chat), 194		(aiogram.methods.answer_shipping_query.AnswerShippingQuery
MessageOriginHiddenUser (class	in	attribute), 443
aiogram.types.message_origin_hidden_user)), r	<pre>nodel_computed_fields</pre>
195		(aiogram.methods.answer_web_app_query.AnswerWebAppQuery
MessageOriginType (class	in	attribute), 436
aiogram.enums.message_origin_type), 468		nodel_computed_fields
MessageOriginUser (class	in	(aiogram.methods.approve_chat_join_request.ApproveChatJoinR
aiogram.types.message_origin_user), 195		attribute), 308 model_computed_fields
MessageReactionCountUpdated (class		(aiogram.methods.ban_chat_member.BanChatMember
aiogram.types.message_reaction_count_upde	uieu),	attribute), 309
MessageReactionUpdated (class	in ,	nodel_computed_fields
aiogram.types.message_reaction_updated),	111 1	(aiogram.methods.ban_chat_sender_chat.BanChatSenderChat
196		attribute), 311
MIGRATE_FROM_CHAT_ID	,	nodel_computed_fields
(aiogram.enums.content_type.ContentType	1	(aiogram.methods.close.Close attribute),
attribute), 460		312
migrate_from_chat_id	r	nodel_computed_fields
(aiogram.types.message.Message attribut		(aiogram.methods.close_forum_topic.CloseForumTopic
148	/ ,	attribute), 313

<pre>model_computed_fields</pre>		computed_fields
(aiogram.methods.close_general_forum_topic.Cl	loseGenera	a \Eioogmifiopnie thods.edit_forum_topic.EditForumTopic
attribute), 314		attribute), 328
<pre>model_computed_fields</pre>	model_c	computed_fields
(aiogram.methods.copy_message.CopyMessage		(aiogram.methods.edit_general_forum_topic.EditGeneralForumT
attribute), 316		attribute), 329
<pre>model_computed_fields</pre>		computed_fields
(aiogram.methods.copy_messages.CopyMessages	\$	$(a iogram.methods.edit_message_caption.EditMessageCaption$
attribute), 318		attribute), 423
model_computed_fields		computed_fields
	eChatInvit	elaingram.methods.edit_message_live_location.EditMessageLiveL
attribute), 319		attribute), 425
<pre>model_computed_fields</pre>		computed_fields
	rumTopic	(aiogram.methods.edit_message_media.EditMessageMedia
attribute), 320		attribute), 427
<pre>model_computed_fields</pre>		computed_fields
	oiceLink	(aiogram.methods.edit_message_reply_markup.EditMessageRepl
attribute), 446		attribute), 428
<pre>model_computed_fields</pre>		computed_fields
	eNewSticke	er(Suitogram.methods.edit_message_text.EditMessageText
attribute), 290		attribute), 430
<pre>model_computed_fields</pre>		computed_fields
	clineChats	IdiniRgquuesstmethods.export_chat_invite_link.ExportChatInviteLink
attribute), 321		attribute), 330
model_computed_fields		computed_fields
(aiogram.methods.delete_chat_photo.DeleteChat	Photo	(aiogram.methods.forward_message.ForwardMessage
attribute), 322	modolo	attribute), 332
model_computed_fields		computed_fields e (Sieo gram.methods.forward_messages.ForwardMessages
attribute), 323	Chaistick	attribute), 333
model_computed_fields	modolo	computed_fields
		(aiogram.methods.get_business_connection.GetBusinessConnecti
attribute), 324	иттори	attribute), 334
model_computed_fields	model c	computed_fields
(aiogram.methods.delete_message.DeleteMessage		(aiogram.methods.get_chat.GetChat_attribute),
attribute), 421	c	335
model_computed_fields	model c	computed_fields
		(aiogram.methods.get_chat_administrators.GetChatAdministrator
attribute), 422	ges	attribute), 336
model_computed_fields	model c	computed_fields
		n(ds iogram.methods.get_chat_member.GetChatMember
attribute), 326	iyeonina	attribute), 337
<pre>model_computed_fields</pre>	model c	computed_fields
		ohusogram.methods.get_chat_member_count.GetChatMemberCour
attribute), 291	Shekeri K	attribute), 338
model_computed_fields	model c	computed_fields
(aiogram.methods.delete_sticker_set.DeleteSticke		(aiogram.methods.get_chat_menu_button.GetChatMenuButton
attribute), 292		attribute), 339
model_computed_fields	model c	computed_fields
(aiogram.methods.delete_webhook.DeleteWebhoo		(aiogram.methods.get_custom_emoji_stickers.GetCustomEmojiSti
attribute), 450		attribute), 293
model_computed_fields	model c	computed_fields
(aiogram.methods.edit_chat_invite_link.EditChat		-
attribute), 327		340

<pre>model_computed_fields</pre>	<pre>model_computed_fields</pre>
(aiogram.methods.get_forum_topic_icon_stickers	s.GetForum(kapig:kannshitkeds .reopen_forum_topic.ReopenForumTopic
attribute), 341	attribute), 356
<pre>model_computed_fields</pre>	<pre>model_computed_fields</pre>
(aiogram.methods.get_game_high_scores.GetGa	meHighScowwinggram.methods.reopen_general_forum_topic.ReopenGeneralH
attribute), 438	attribute), 357
<pre>model_computed_fields</pre>	<pre>model_computed_fields</pre>
(aiogram.methods.get_me.GetMe attribute), 342	(aiogram.methods.replace_sticker_in_set.ReplaceStickerInSet attribute), 295
model_computed_fields	model_computed_fields
(aiogram.methods.get_my_commands.GetMyCom	
attribute), 343	attribute), 358
<pre>model_computed_fields</pre>	<pre>model_computed_fields</pre>
(aiogram.methods.get_my_default_administrator	_rights.Get(MidgefanlmAdhidissteatcheRights_invite_link.RevokeChatInviteLink
attribute), 344	attribute), 360
<pre>model_computed_fields</pre>	<pre>model_computed_fields</pre>
(aiogram.methods.get_my_description.GetMyDes	scription (aiogram.methods.send_animation.SendAnimation
attribute), 345	attribute), 362
<pre>model_computed_fields</pre>	<pre>model_computed_fields</pre>
(aiogram.methods.get_my_name.GetMyName	(aiogram.methods.send_audio.SendAudio
attribute), 346	attribute), 365
<pre>model_computed_fields</pre>	<pre>model_computed_fields</pre>
	MyShortDecarigtion.methods.send_chat_action.SendChatAction
attribute), 347	attribute), 367
<pre>model_computed_fields</pre>	<pre>model_computed_fields</pre>
(aiogram.methods.get_sticker_set.GetStickerSet	(aiogram.methods.send_contact.SendContact
attribute), 294	attribute), 368
<pre>model_computed_fields</pre>	<pre>model_computed_fields</pre>
(aiogram.methods.get_updates.GetUpdates	(aiogram.methods.send_dice.SendDice at-
attribute), 452	<pre>tribute), 371 model_computed_fields</pre>
model_computed_fields	rChatBoost{aiogram.methods.send_document.SendDocument
attribute), 348	attribute), 373
model_computed_fields	model_computed_fields
	serProfileP(uitogram.methods.send_game.SendGame
attribute), 348	attribute), 439
model_computed_fields	model_computed_fields
-	kInfo (aiogram.methods.send_invoice.SendInvoice
attribute), 453	attribute), 448
model_computed_fields	model_computed_fields
-	deGeneralF(ariangifamimethods.send_location.SendLocation
attribute), 349	attribute), 375
model_computed_fields	model_computed_fields
(aiogram.methods.leave_chat.LeaveChat	(aiogram.methods.send_media_group.SendMediaGroup
attribute), 350	attribute), 378
model_computed_fields	model_computed_fields
(aiogram.methods.log_out.LogOut attribute),	(aiogram.methods.send_message.SendMessage
351	attribute), 380
<pre>model_computed_fields</pre>	<pre>model_computed_fields</pre>
(aiogram.methods.pin_chat_message.PinChatMe	
attribute), 352	attribute), 382
model_computed_fields	<pre>model_computed_fields</pre>
(aiogram.methods.promote_chat_member.Promo	teChatMem(lociogram.methods.send_poll.SendPoll at-
attribute), 354	tribute), 385

<pre>model_computed_fields</pre>	model_	computed_fields
$(a iogram.methods.send_sticker.SendSticker$		(aiogram.methods.set_my_name.SetMyName
attribute), 297		attribute), 410
<pre>model_computed_fields</pre>	model_	computed_fields
(aiogram.methods.send_venue.SendVenue		(aiogram.methods.set_my_short_description.SetMyShortDescript
attribute), 388		attribute), 411
<pre>model_computed_fields</pre>	model_	computed_fields
(aiogram.methods.send_video.SendVideo		(aiogram.methods.set_passport_data_errors.SetPassportDataErro
attribute), 391		attribute), 456
<pre>model_computed_fields</pre>		computed_fields
(aiogram.methods.send_video_note.SendVideoNo attribute), 393	ote	(aiogram.methods.set_sticker_emoji_list.SetStickerEmojiList attribute), 299
<pre>model_computed_fields</pre>	model_	computed_fields
(aiogram.methods.send_voice.SendVoice attribute), 396		(aiogram.methods.set_sticker_keywords.SetStickerKeywords attribute), 300
<pre>model_computed_fields</pre>	model_	computed_fields
(aiogram.methods.set_chat_administrator_custor	n_title.Se	tQlaiao&daminisethavalsCsues_cstnidliehe_mask_position.SetStickerMaskPosit
attribute), 398		attribute), 301
<pre>model_computed_fields</pre>	model_	computed_fields
(aiogram.methods.set_chat_description.SetChatL attribute), 399	Descriptio	n(aiogram.methods.set_sticker_position_in_set.SetStickerPositionIn attribute), 302
model_computed_fields	model (computed_fields
		tt @n iogram.methods.set_sticker_set_thumbnail.SetStickerSetThumbi
attribute), 400	ши	attribute), 304
model_computed_fields	model d	computed_fields
		msaiogram.methods.set_sticker_set_title.SetStickerSetTitle
attribute), 401		attribute), 305
<pre>model_computed_fields</pre>	model_	computed_fields
(aiogram.methods.set_chat_photo.SetChatPhoto		(aiogram.methods.set_webhook.SetWebhook
attribute), 402		attribute), 454
<pre>model_computed_fields</pre>		computed_fields
(aiogram.methods.set_chat_sticker_set.SetChatSt attribute), 403	tickerSet	(aiogram.methods.stop_message_live_location.StopMessageLiveL attribute), 431
<pre>model_computed_fields</pre>	model_	computed_fields
(aiogram.methods.set_chat_title.SetChatTitle attribute), 404		(aiogram.methods.stop_poll.StopPoll attribute), 433
<pre>model_computed_fields</pre>	model_0	computed_fields
		il.(Suit Guston Hethojal Stickear S <u>e</u> t Flaut <u>n</u> them Her. Unban Chat Member
attribute), 298	_	attribute), 413
<pre>model_computed_fields</pre>	model_	computed_fields
(aiogram.methods.set_game_score.SetGameScor		(aiogram.methods.unban_chat_sender_chat.UnbanChatSenderCh
attribute), 441		attribute), 414
<pre>model_computed_fields</pre>	model_	computed_fields
(aiogram.methods.set_message_reaction.SetMess	sageReact	tidmiogram.methods.unhide_general_forum_topic.UnhideGeneralFo
attribute), 405		attribute), 415
<pre>model_computed_fields</pre>		computed_fields
(aiogram.methods.set_my_commands.SetMyCom attribute), 407	mands	(aiogram.methods.unpin_all_chat_messages.UnpinAllChatMessages, attribute), 416
model_computed_fields	model d	computed_fields
		et /hjiDgfault/hdthivistnap in <u>Right</u> forum_topic_messages.UnpinAllFor
attribute), 408	U	<i>attribute</i>), 417
<pre>model_computed_fields</pre>	model_	computed_fields
(aiogram.methods.set_my_description.SetMyDes attribute), 409	cription	(aiogram.methods.unpin_all_general_forum_topic_messages.Unp attribute), 418

<pre>model_computed_fields</pre>	(aiogram.types.business_intro.BusinessIntro
(aiogram.methods.unpin_chat_message.UnpinCh	
attribute), 419	<pre>model_computed_fields</pre>
<pre>model_computed_fields</pre>	(aiogram.types.business_location.BusinessLocation
(aiogram.methods.upload_sticker_file.UploadStic	
attribute), 306	<pre>model_computed_fields</pre>
model_computed_fields	(aiogram.types.business_messages_deleted.BusinessMessagesDel
(aiogram.types.animation.Animation attribute),	attribute), 26
17	model_computed_fields
<pre>model_computed_fields (aiogram.types.audio.Audio</pre>	(aiogram.types.business_opening_hours.BusinessOpeningHours attribute), 27
<pre>model_computed_fields</pre>	<pre>model_computed_fields</pre>
(aiogram.types.birthdate.Birthdate attribute), 19	(aiogram.types.business_opening_hours_interval.BusinessOpenir attribute), 27
<pre>model_computed_fields</pre>	<pre>model_computed_fields</pre>
(aiogram.types.bot_command.BotCommand attribute), 19	(aiogram.types.callback_game.CallbackGame attribute), 287
model_computed_fields	model_computed_fields
(aiogram.types.bot_command_scope.BotComman	
attribute), 20	attribute), 28
model_computed_fields	<pre>model_computed_fields (aiogram.types.chat.Chat at-</pre>
	dministrato rsiBiate GommandScopeAllChatAdministrators
attribute), 20	<pre>model_computed_fields</pre>
<pre>model_computed_fields</pre>	(aiogram.types.chat_administrator_rights.ChatAdministratorRights.
(aiogram.types.bot_command_scope_all_group_	
attribute), 21	<pre>model_computed_fields</pre>
model_computed_fields	(aiogram.types.chat_boost.ChatBoost at-
(aiogram.types.bot_command_scope_all_private	
attribute), 21	model_computed_fields
<pre>model_computed_fields</pre>	(aiogram.types.chat_boost_added.ChatBoostAdded
(aiogram.types.bot_command_scope_chat.BotCo	
attribute), 22	<pre>model_computed_fields</pre>
<pre>model_computed_fields</pre>	(aiogram.types.chat_boost_removed.ChatBoostRemoved
(aiogram.types.bot_command_scope_chat_admir	nistrators.Botfibuta).phlScopeChatAdministrators
attribute), 22	model_computed_fields
<pre>model_computed_fields</pre>	(aiogram.types.chat_boost_source.ChatBoostSource
(aiogram.types.bot_command_scope_chat_memb	per.BotCom attmillStw)p&ChatMember
attribute), 23	<pre>model_computed_fields</pre>
<pre>model_computed_fields</pre>	(aiogram.types.chat_boost_source_gift_code.ChatBoostSourceGi
(aiogram.types.bot_command_scope_default.Bot	Command San pib Duffy, AT
attribute), 23	<pre>model_computed_fields</pre>
<pre>model_computed_fields</pre>	(aiogram.types.chat_boost_source_giveaway.ChatBoostSourceGi
(aiogram.types.bot_description.BotDescription	attribute), 48
attribute), 24	<pre>model_computed_fields</pre>
<pre>model_computed_fields</pre>	(aiogram.types.chat_boost_source_premium.ChatBoostSourcePre
(aiogram.types.bot_name.BotName attribute),	attribute), 48
24	<pre>model_computed_fields</pre>
<pre>model_computed_fields</pre>	(aiogram.types.chat_boost_updated.ChatBoostUpdated
(aiogram.types.bot_short_description.BotShortD	escription attribute), 49
attribute), 24	model_computed_fields
model_computed_fields	(aiogram.types.chat_invite_link.ChatInviteLink
$(aiogram.types.business_connection.BusinessContextContex$	
attribute), 25	<pre>model_computed_fields</pre>
<pre>model_computed_fields</pre>	(aiogram.types.chat_join_request.ChatJoinRequest

attribute), 81	<pre>model_computed_fields</pre>
<pre>model_computed_fields</pre>	(aiogram.types.error_event.ErrorEvent at-
(aiogram.types.chat_location.ChatLocation	tribute), 539
attribute), 87	<pre>model_computed_fields</pre>
<pre>model_computed_fields</pre>	(aiogram.types.external_reply_info.ExternalReplyInfo
(aiogram.types.chat_member.ChatMember	attribute), 118
attribute), 88	model_computed_fields (<i>aiogram.types.file.File at-</i>
<pre>model_computed_fields</pre>	tribute), 119
attribute), 89	(aiogram.types.force_reply.ForceReply at-
model_computed_fields	tribute), 120
(aiogram.types.chat_member_banned.ChatMem	
attribute), 90	(aiogram.types.forum_topic.ForumTopic
<pre>model_computed_fields</pre>	attribute), 120
(aiogram.types.chat_member_left.ChatMember]	
attribute), 90	(aiogram.types.forum_topic_closed.ForumTopicClosed
<pre>model_computed_fields</pre>	attribute), 120
(aiogram.types.chat_member_member.ChatMen	nbmoldelnbeomputed_fields
attribute), 91	(aiogram.types.forum_topic_created.ForumTopicCreated
<pre>model_computed_fields</pre>	attribute), 121
(aiogram.types.chat_member_owner.ChatMemb	
attribute), 91	(aiogram.types.forum_topic_edited.ForumTopicEdited
<pre>model_computed_fields</pre>	attribute), 121
(aiogram.types.chat_member_restricted.ChatMe	
attribute), 92	(aiogram.types.forum_topic_reopened.ForumTopicReopened
<pre>model_computed_fields</pre>	attribute), 122
(atogram.types.cnat_member_updated.ChatMen attribute), 106	nbmo@pdlatedmputed_fields (aiogram.types.game.Game attribute), 287
model_computed_fields	model_computed_fields
(aiogram.types.chat_permissions.ChatPermissio	-
attribute), 113	attribute), 288
<pre>model_computed_fields</pre>	model_computed_fields
(aiogram.types.chat_photo.ChatPhoto at-	(aiogram.types.general_forum_topic_hidden.GeneralForumTopic
tribute), 113	attribute), 122
<pre>model_computed_fields</pre>	<pre>model_computed_fields</pre>
(aiogram.types.chat_shared.ChatShared	(aiogram.types.general_forum_topic_unhidden.GeneralForumTopic
attribute), 114	attribute), 122
<pre>model_computed_fields</pre>	<pre>model_computed_fields</pre>
(aiogram.types.chosen_inline_result.ChosenInli. attribute), 216	neResult (aiogram.types.giveaway.Giveaway attribute), 123
<pre>model_computed_fields</pre>	<pre>model_computed_fields</pre>
(aiogram.types.contact.Contact attribute), 115	(aiogram.types.giveaway_completed.GiveawayCompleted attribute), 123
<pre>model_computed_fields (aiogram.types.dice.Dice at-</pre>	<pre>model_computed_fields</pre>
tribute), 115	(aiogram.types.giveaway_created.GiveawayCreated
<pre>model_computed_fields</pre>	attribute), 124
(aiogram.types.document.Document attribute),	
116	(aiogram.types.giveaway_winners.GiveawayWinners
<pre>model_computed_fields</pre>	attribute), 124
$(aiogram.types.encrypted_credentials.Encrypted$	
attribute), 267	(aiogram.types.inaccessible_message.InaccessibleMessage
model_computed_fields	attribute), 125
(aiogram.types.encrypted_passport_element.En	
attribute), 268	$(a iogram.types.in line_keyboard_button.In lineKeyboardButton$

attribute		attribute), 246
model_computed		<pre>model_computed_fields</pre>
-		eyboardMathiqgram.types.inline_query_result_mpeg4_gif.InlineQueryResult
attribute		attribute), 248
model_computed		<pre>model_computed_fields</pre>
-	n.types.inline_query.InlineQuery	(aiogram.types.inline_query_result_photo.InlineQueryResultPhot
attribute		attribute), 250
model_computed	n.types.inline_query_result.InlineQueryK	<pre>model_computed_fields Result (aiogram.types.inline_query_result_venue.InlineQueryResultVenu</pre>
attribute		attribute), 252
model_computed		model_computed_fields
-		eQueryResultArgicle.types.inline_query_result_video.InlineQueryResultVide
attribute		attribute), 254
model_computed		model_computed_fields
-		QueryResultatingiam.types.inline_query_result_voice.InlineQueryResultVoice
attribute		z attribute), 255
model_computed		<pre>model_computed_fields</pre>
(aiograi	n.types.inline_query_result_cached_audi	io.InlineQu (a)Rescult.CypelseiteLind ioquery_results_button.InlineQueryResultsB
attribute	?), 223	attribute), 256
model_computed	_fields	<pre>model_computed_fields</pre>
(aiograi	n.types.inline_query_result_cached_doci	ıment.Inlin @QingryRe_typhtSanped_Doortana <u>n</u> tnessage_content.InputContactMes
attribute		attribute), 257
model_computed		<pre>model_computed_fields</pre>
		nlineQuery RaisightandypdGif iput_invoice_message_content.InputInvoiceMess
attribute		attribute), 259
<pre>model_computed</pre>		<pre>model_computed_fields</pre>
(aiograi attribute		g4_gif.Inlin (eQugnxyRayphtCimphat<u>d</u>Mpatj#Gif nessage_content.InputLocationM attribute), 260
model_computed	_fields	<pre>model_computed_fields</pre>
(aiograi	n.types.inline_query_result_cached_phot	o.InlineQu ¢ayðæsult£yndæilffhø<u>t</u>o nedia.InputMedia
attribute		attribute), 128
model_computed		<pre>model_computed_fields</pre>
		er.InlineQu(enigResult©pek.edfuir_henedia_animation.InputMediaAnimation
attribute		attribute), 129
model_computed		<pre>model_computed_fields</pre>
		o.InlineQu & wiRgsauhtGywelwe.dWidt onedia_audio.InputMediaAudio
attribute		attribute), 130
model_computed		<pre>model_computed_fields</pre>
-		e.InlineQue(wiRgsultGyphsdWpide_media_document.InputMediaDocument
attribute		<pre>attribute), 131 model_computed_fields</pre>
model_computed		model_computed_fletas eQueryRes(di6gmtmctypes.input_media_photo.InputMediaPhoto
attribute		attribute), 132
model_computed		model_computed_fields
		lineQueryRasiolgEmaitypenstinput_media_video.InputMediaVideo
attribute		attribute), 133
model_computed		model_computed_fields
-		QueryResultGiangueum.types.input_message_content.InputMessageContent
attribute		attribute), 261
model_computed		model_computed_fields
-		eryResultGlfaiogram.types.input_sticker.InputSticker
attribute		attribute), 264
model_computed		<pre>model_computed_fields</pre>
(aiograi	n.types.inline_query_result_location.Inli	neQueryRefa ltlgoantityp es.input_text_message_content.InputTextMessageCo

attribute), 262	149
<pre>model_computed_fields</pre>	<pre>model_computed_fields</pre>
(aiogram.types.input_venue_message_content.Inp	putVenueM (siagediontypet s.message_auto_delete_timer_changed.MessageAut
attribute), 262	attribute), 192
model_computed_fields	model_computed_fields
(aiogram.types.invoice.Invoice attribute), 278	(aiogram.types.message_entity.MessageEntity attribute), 192
<pre>model_computed_fields</pre>	<pre>model_computed_fields</pre>
(aiogram.types.keyboard_button.KeyboardButton attribute), 134	a (aiogram.types.message_id.MessageId at- tribute), 193
<pre>model_computed_fields</pre>	<pre>model_computed_fields</pre>
	oardButton RallFype n.types.message_origin.MessageOrigin
attribute), 135	attribute), 193
<pre>model_computed_fields</pre>	<pre>model_computed_fields</pre>
attribute), 137	eyboardBut (airBeque.styfelsu tnessage_origin_channel.MessageOriginChannel attribute), 194
<pre>model_computed_fields</pre>	<pre>model_computed_fields</pre>
attribute), 137	eyboardBut (airRgqme.st\$pss message_origin_chat.MessageOriginChat attribute), 194
<pre>model_computed_fields</pre>	<pre>model_computed_fields</pre>
	KeyboardBu (niongPreum.tspldson æssage_origin_hidden_user.MessageOriginHida
attribute), 138	attribute), 195
<pre>model_computed_fields</pre>	<pre>model_computed_fields</pre>
(aiogram.types.labeled_price.LabeledPrice attribute), 279	(aiogram.types.message_origin_user.MessageOriginUser attribute), 195
<pre>model_computed_fields</pre>	<pre>model_computed_fields</pre>
attribute), 139	<i>wOptions</i> (aiogram.types.message_reaction_count_updated.MessageReaction_attribute), 196
<pre>model_computed_fields</pre>	<pre>model_computed_fields</pre>
(aiogram.types.location.Location attribute), 140	(aiogram.types.message_reaction_updated.MessageReactionUpdattribute), 197
<pre>model_computed_fields</pre>	<pre>model_computed_fields</pre>
(aiogram.types.login_url.LoginUrl attribute), 141	(aiogram.types.order_info.OrderInfo attribute), 279
<pre>model_computed_fields</pre>	<pre>model_computed_fields</pre>
(aiogram.types.mask_position.MaskPosition attribute), 264	(aiogram.types.passport_data.PassportData attribute), 269
<pre>model_computed_fields</pre>	model_computed_fields
attribute), 141	ybeInacces(xibdeMuenstygees.passport_element_error.PassportElementError attribute), 270
<pre>model_computed_fields</pre>	<pre>model_computed_fields</pre>
(aiogram.types.menu_button.MenuButton attribute), 142	(aiogram.types.passport_element_error_data_field.PassportElem attribute), 270
<pre>model_computed_fields</pre>	<pre>model_computed_fields</pre>
(aiogram.types.menu_button_commands.MenuBu attribute), 142	uttonComm(midgram.types.passport_element_error_file.PassportElementErro attribute), 271
<pre>model_computed_fields</pre>	<pre>model_computed_fields</pre>
(aiogram.types.menu_button_default.MenuButtor attribute), 142	<i>Default</i> (aiogram.types.passport_element_error_files.PassportElementErr attribute), 272
<pre>model_computed_fields</pre>	<pre>model_computed_fields</pre>
(aiogram.types.menu_button_web_app.MenuButt attribute), 143	tonWebApp(aiogram.types.passport_element_error_front_side.PassportElem attribute), 273
<pre>model_computed_fields</pre>	<pre>model_computed_fields</pre>
(aiogram.types.message.Message attribute),	(aiogram.types.passport_element_error_reverse_side.PassportEle

attribute), 274	model_computed_fields
<pre>model_computed_fields</pre>	$(aiogram.types.response_parameters.ResponseParameters$
(aiogram.types.passport_element_error_selfie.Pa	
attribute), 275	model_computed_fields
model_computed_fields	(aiogram.types.sent_web_app_message.SentWebAppMessage
(aiogram.types.passport_element_error_translate	
attribute), 276	<pre>model_computed_fields</pre>
model_computed_fields	(aiogram.types.shared_user.SharedUser
(aiogram.types.passport_element_error_translati	<pre>ion_piles.PassportagementErrorTranslationFiles model_computed_fields</pre>
<i>attribute</i>), 276 model_computed_fields	(aiogram.types.shipping_address.ShippingAddress
(aiogram.types.passport_element_error_unspecij	
attribute), 277	model_computed_fields
model_computed_fields	(aiogram.types.shipping_option.ShippingOption
(aiogram.types.passport_file.PassportFile	attribute), 282
attribute), 278	model_computed_fields
model_computed_fields	(aiogram.types.shipping_query.ShippingQuery
(aiogram.types.photo_size.PhotoSize attribute),	attribute), 282
197	<pre>model_computed_fields (aiogram.types.sticker.Sticker</pre>
<pre>model_computed_fields (aiogram.types.poll.Poll at-</pre>	attribute), 265
tribute), 198	<pre>model_computed_fields</pre>
<pre>model_computed_fields</pre>	(aiogram.types.sticker_set.StickerSet attribute),
(aiogram.types.poll_answer.PollAnswer at-	267
tribute), 199	<pre>model_computed_fields (aiogram.types.story.Story</pre>
<pre>model_computed_fields</pre>	attribute), 205
	model_computed_fields
tribute), 199	(aiogram.types.successful_payment.SuccessfulPayment
model_computed_fields	attribute), 283
(aiogram.types.pre_checkout_query.PreCheckout attribute), 280	(aiogram.types.switch_inline_query_chosen_chat.SwitchInlineQu
model_computed_fields	attribute), 206
(aiogram.types.proximity_alert_triggered.Proxim	
attribute), 200	(aiogram.types.text_quote.TextQuote attribute),
model_computed_fields	207
(aiogram.types.reaction_count.ReactionCount	<pre>model_computed_fields</pre>
attribute), 200	(aiogram.types.update.Update attribute),
<pre>model_computed_fields</pre>	285
(aiogram.types.reaction_type.ReactionType	<pre>model_computed_fields (aiogram.types.user.User at-</pre>
attribute), 200	tribute), 208
<pre>model_computed_fields</pre>	<pre>model_computed_fields</pre>
	ctionTypeC(ustogrEmaji pes.user_chat_boosts.UserChatBoosts
attribute), 201	attribute), 209
model_computed_fields	model_computed_fields
(aiogram.types.reaction_type_emoji.ReactionTyp	
attribute), 201	attribute), 209
model_computed_fields	<pre>model_computed_fields yboardMarNaipgram.types.user_shared.UserShared</pre>
attribute), 202	attribute), 209
model_computed_fields	model_computed_fields
	yboardRem(wiogram.types.users_shared.UsersShared
attribute), 203	attribute), 210
model_computed_fields	<pre>model_computed_fields (aiogram.types.venue.Venue</pre>
(aiogram.types.reply_parameters.ReplyParameters	
attribute), 204	<pre>model_computed_fields (aiogram.types.video.Video</pre>

method), 307 attribute), 211 model_computed_fields model_post_init() (aiogram.methods.answer_inline_query.AnswerInlin (aiogram.types.video_chat_ended.VideoChatEnded method), 435 attribute), 212 model_post_init() (aiogram.methods.answer_pre_checkout_query.Answ model_computed_fields method), 442 (aiogram.types.video_chat_participants_invited.VindedUngBstiliaipint(S) (winggram.methods.answer_shipping_query.AnswerS) method), 443 attribute), 212 model_post_init() (aiogram.methods.answer_web_app_query.AnswerW model_computed_fields (aiogram.types.video_chat_scheduled.VideoChatScheduled method), 436 model_post_init() (aiogram.methods.approve_chat_join_request.Appro attribute), 213 model_computed_fields method), 308 (aiogram.types.video_chat_started.VideoChatStartadel_post_init() (aiogram.methods.ban_chat_member.BanChatMemb method), 310 attribute), 213 model_computed_fields model_post_init() (aiogram.methods.ban_chat_sender_chat.BanChatSet (aiogram.types.video_note.VideoNote method), 311 attribute), 213 model_post_init() (aiogram.methods.close.Close model_computed_fields (aiogram.types.voice.Voice method), 312 attribute), 214 model_post_init() (aiogram.methods.close_forum_topic.CloseForumTo model_computed_fields method), 313 (aiogram.types.web_app_data.WebAppData model_post_init() (aiogram.methods.close_general_forum_topic.Close attribute), 214 method), 314 model_computed_fields model_post_init() (aiogram.methods.copy_message.CopyMessage (aiogram.types.web_app_info.WebAppInfo method), 316 attribute), 215 model_post_init() (aiogram.methods.copy messages.CopyMessages model_computed_fields method), 318 (aiogram.types.webhook_info.WebhookInfo model_post_init() (aiogram.methods.create_chat_invite_link.CreateChat_invite_link.Crea attribute), 286 method), 319 model_computed_fields model_post_init() (aiogram.methods.create_forum_topic.CreateForum') (aiogram.types.write_access_allowed.WriteAccessAllowed_method), 320 attribute), 215 model_post_init() (aiogram.methods.create_invoice_link.CreateInvoice model_computed_fields method), 446 (aiogram.utils.web_app.WebAppChat model_post_init() (aiogram.methods.create_new_sticker_set.CreateNew attribute), 562 method), 290 model_computed_fields model_post_init() (aiogram.methods.decline_chat_join_request.Decline (aiogram.utils.web_app.WebAppInitData method), 321 attribute), 560 model_post_init() (aiogram.methods.delete_chat_photo.DeleteChatPho model_computed_fields method), 322 (aiogram.utils.web_app.WebAppUser model_post_init() (aiogram.methods.delete_chat_sticker_set.DeleteChat attribute), 561 method), 323 model_config (aiogram.utils.web_app.WebAppChat atmodel_post_init() (aiogram.methods.delete_forum_topic.DeleteForumT tribute), 562 method), 324 model_config (aiogram.utils.web_app.WebAppInitData model_post_init() (aiogram.methods.delete_message.DeleteMessage method), 421 attribute), 560 model_config (aiogram.utils.web_app.WebAppUser atmodel_post_init() (aiogram.methods.delete_messages.DeleteMessages tribute), 561 method), 422 model_fields (aiogram.utils.web_app.WebAppChat atmodel_post_init() (aiogram.methods.delete_my_commands.DeleteMyC tribute), 563 method), 326 model_fields (aiogram.utils.web_app.WebAppInitData model_post_init() (aiogram.methods.delete_sticker_from_set.DeleteStic attribute), 560 *method*), 291 model_fields (aiogram.utils.web_app.WebAppUser atmodel_post_init() (aiogram.methods.delete_sticker_set.DeleteStickerSe method), 292 tribute), 562 model_post_init() (aiogram.methods.add_sticker_to_sembddestigkasEcSurit() (aiogram.methods.delete_webhook.DeleteWebhook method), 289 method), 451 model_post_init() (aiogram.methods.answer_callback_quady Apposer Cialiba () Quiergram.methods.edit_chat_invite_link.EditChatInv

	<i>method</i>), 327	<i>method</i>), 294
model_	_post_init() (aiogram.methods.edit_forum_topic.EndidEdry	<pre>moExpianit() (aiogram.methods.get_updates.GetUpdates</pre>
	<i>method</i>), 328	<i>method</i>), 452
model_	_post_init() (aiogram.methods.edit_general_forumodoplc_	EdstGeineintlep(uuingopn:methods.get_user_chat_boosts.GetUserCha
	method), 329	<i>method</i>), 348
model		MessageCuplioniogram.methods.get_user_profile_photos.GetUserH
	method), 423	<i>method</i>), 348
model		pdstillithistoge Liviogramtinethods.get_webhook_info.GetWebhookInf
mouci_	method), 425	method), 453
model		fasst_gither() (aiogram.methods.hide_general_forum_topic.HideG
mouci_	method), 427	method), 349
model	_post_init() (aiogram.methods.edit_message_repl	
mouer_	method), 428	method), 350
model_	post_init() (aiogram.methods.edit_message_text_foldes/leg	
	<i>method</i>), 430	<i>method</i>), 351
model_		postChadin(i)(landgram.methods.pin_chat_message.PinChatMessag
	<i>method</i>), 330	method), 352
model_		boss vagnit() (aiogram.methods.promote_chat_member.PromoteCl
	<i>method</i>), 332	<i>method</i>), 355
model_	_post_init() (aiogram.methods.forward_messagesnfodedar	d dfæssigast () (aiogram.methods.reopen_forum_topic.ReopenForum
	<i>method</i>), 333	<i>method</i>), 356
model_	_post_init() (aiogram.methods.get_business_conmaothen_G	astation
	<i>method</i>), 334	<i>method</i>), 357
model_	_post_init()(aiogram.methods.get chat.GetChat model_)	<pre>post_init() (aiogram.methods.replace_sticker_in_set.ReplaceStic</pre>
	method), 335	method), 295
model		etGhatihalini(i)s(naiogram.methods.restrict_chat_member.RestrictCha
	method), 336	method), 358
model		Montbanit() (aiogram.methods.revoke_chat_invite_link.RevokeCh
mouci_	method), 337	method), 360
modol		weinou), 500
mouer_		
	method), 338	method), 362
model_	_post_init() (aiogram.methods.get_chat_menu_butwodeGet	
	<i>method</i>), 339	<i>method</i>), 365
model_		66 8tCristional Description Spicken methods.send_chat_action.SendChatAction
	<i>method</i>), 293	<i>method</i>), 367
model_	_post_init() (<i>aiogram.methods.get_file.GetFile</i> model_j	
	<i>method</i>), 340	<i>method</i>), 368
model_	_post_init() (aiogram.methods.get_forum_topic_imod_slic	borstGatfive(i)n(liapignlaronnStatthoods.send_dice.SendDice
	<i>method</i>), 341	<i>method</i>), 371
model_	_post_init() (aiogram.methods.get_game_high_scmodeGeg	Cost <u>e</u> HighS Cowwogram.methods.send_document.SendDocument
	<i>method</i>), 438	<i>method</i>), 373
model_	_post_init() (aiogram.methods.get_me.GetMe model_j	<pre>post_init() (aiogram.methods.send game.SendGame</pre>
	method), 342	method), 439
model		Sostminds () (aiogram.methods.send_invoice.SendInvoice
model	<i>method</i>), 343	<i>method</i>), 448
model_	method), 343 post_init() (aiogram.methods.get_my_default_admoidestra	method), 448 gostrighist@iMidgefanlaAdhidisstemubrRightion.SendLocation
	method), 343 post_init() (aiogram.methods.get_my_default_admoidestragestection), 344	method), 448 gostrighist@MidgefauluAdhidisstenubrRightion.SendLocation method), 376
	<pre>method), 343 post_init() (aiogram.methods.get_my_default_admoidestrag method), 344 post_init() (aiogram.methods.get_my_descriptiom6&eMy</pre>	method), 448 gostrighist@MydgefauluAdhudisstenubrRightion.SendLocation method), 376 gosscription() (aiogram.methods.send_media_group.SendMediaGra
model_	<pre>method), 343 post_init() (aiogram.methods.get_my_default_admoidestrag method), 344 post_init() (aiogram.methods.get_my_descriptiomothedMy method), 345</pre>	method), 448 aos_trighistGetMidgefanluAdhuidisstenub_fRightis on.SendLocation method), 376 Josst_riptiou() (aiogram.methods.send_media_group.SendMediaGra method), 378
model_	<pre>method), 343 post_init() (aiogram.methods.get_my_default_admoidestrag method), 344 post_init() (aiogram.methods.get_my_descriptiomodestrag method), 345 post_init() (aiogram.methods.get_my_name.GetModestrag)</pre>	method), 448 aost_righistGetMidGefanluAdhuidisstenub_Rightis on.SendLocation method), 376 Desst_riptiou() (aiogram.methods.send_media_group.SendMediaGra method), 378 post_init() (aiogram.methods.send_message.SendMessage
model_ model_	<pre>method), 343 post_init() (aiogram.methods.get_my_default_admoidestrag method), 344 post_init() (aiogram.methods.get_my_descriptiomodedMy method), 345 post_init() (aiogram.methods.get_my_name.GetModeling method), 346</pre>	method), 448 aost_righist@atMigBefanluAdhuidisstenub_fRightis on.SendLocation method), 376 fDesst_ripticu() (aiogram.methods.send_media_group.SendMediaGra method), 378 post_init() (aiogram.methods.send_message.SendMessage method), 380
model_ model_	<pre>method), 343 post_init() (aiogram.methods.get_my_default_admoidestray method), 344 post_init() (aiogram.methods.get_my_descriptiomofdedMy method), 345 post_init() (aiogram.methods.get_my_name.GetModdedMy method), 346 post_init() (aiogram.methods.get_my_short_descuoodedAnt)</pre>	method), 448 aos <u>t</u> ri żhi st G at Mi d igefanluAdhuidisstento <u>r</u> Rightio n.SendLocation method), 376 fossc <u>r</u> ipuiou () (aiogram.methods.send_media_group.SendMediaGra method), 378 post_init() (aiogram.methods.send_message.SendMessage method), 380 fossMyShirtD(xaicigtion .methods.send_photo.SendPhoto
model_ model_ model_	<pre>method), 343 post_init() (aiogram.methods.get_my_default_admoidestrag method), 344 post_init() (aiogram.methods.get_my_descriptiomodedMy method), 345 post_init() (aiogram.methods.get_my_name.GetModeling method), 346</pre>	<pre>method), 448 method), 448 method), 376 method), 376 method), 376 method), 378 method), 378 method), 378 method), 380 method), 380 method), 380 method), 380 method), 382</pre>

<i>method</i>), 385	<i>method</i>), 305
<pre>model_post_init() (aiogram.methods.send_sticker.SendSmodel_</pre>	<pre>post_init() (aiogram.methods.set_webhook.SetWebhook</pre>
method), 297	<i>method</i>), 454
<pre>model_post_init() (aiogram.methods.send_venue.SendVmodel_</pre>	<pre>post_init() (aiogram.methods.stop message live location.StopM</pre>
method), 388	<i>method</i>), 431
<pre>model_post_init() (aiogram.methods.send_video.SendVinladel_</pre>	
method), 391	<i>method</i>), 433
<pre>model_post_init() (aiogram.methods.send_video_note.SmodElde</pre>	
<i>method</i>), 393	method), 413
<pre>model_post_init() (aiogram.methods.send_voice.SendVoicelel_</pre>	
method), 396	method), 414
<pre>model_post_init() (aiogram.methods.set_chat_administmode_lc_</pre>	
method), 398	method), 415
<pre>model_post_init() (aiogram.methods.set_chat_descriptionededC)</pre>	
method), 399	method), 416
<pre>model_post_init() (aiogram.methods.set_chat_menu_buttodeSet</pre>	
method), 400	method), 417
model_post_init() (aiogram.methods.set_chat_permissioncdsar(
method), 401	method), 418
<pre>model_post_init() (aiogram.methods.set_chat_photo.SetCodePh</pre>	
<i>method</i>), 402	<i>method</i>), 419
<pre>model_post_init() (aiogram.methods.set_chat_sticker_smosterChat_sticker_sticker_sticker_sticker_stick</pre>	
<i>method</i>), 403	method), 306
<pre>model_post_init() (aiogram.methods.set_chat_title.SetChadEile</pre>	
<i>method</i>), 404	<i>method</i>), 18
<pre>model_post_init() (aiogram.methods.set_custom_emoji_modedr_</pre>	po <u>s_th</u> imba(d).SetCust oinEnanjiStpelsearSdiTlAundlio ail
<i>method</i>), 299	method), 18
<pre>model_post_init() (aiogram.methods.set_game_score.SetGdeles)</pre>	<pre>posst_init() (aiogram.types.birthdate.Birthdate</pre>
<i>method</i>), 441	method), 19
<pre>model_post_init() (aiogram.methods.set_message_reactiondsatt)</pre>	possugiRia(f)(miogram.types.bot_command.BotCommand
<i>method</i>), 406	method), 19
<pre>model_post_init() (aiogram.methods.set_my_commandsmodelAy)</pre>	porstuainist () (aiogram.types.bot_command_scope.BotCommandSo
<i>method</i>), 407	<i>method</i>), 20
<pre>model_post_init() (aiogram.methods.set_my_default_admadestra</pre>	pos <u>trightistSe)MyiDefaulttAydesi.bistratamRightds</u> scope_all_chat_admit
method), 409	method), 20
<pre>model_post_init() (aiogram.methods.set_my_descriptionsSetEXYy_</pre>	
<i>method</i>), 410	<i>method</i>), 21
<pre>model_post_init() (aiogram.methods.set_my_name.SetMyotherhe_</pre>	
<i>method</i>), 410	<i>method</i>), 21
<pre>model_post_init() (aiogram.methods.set_my_short_desaniptiedn_</pre>	
method), 412	method), 22
<pre>model_post_init() (aiogram.methods.set_passport_datamedels.</pre>	
method), 456	method), 22
<pre>model_post_init() (aiogram.methods.set_sticker_emoji_hiotl@a/s</pre>	
method), 300	method), 23
<pre>model_post_init() (aiogram.methods.set_sticker_keywomdodSelSy d_b_b_200</pre>	
method), 300	method), 23
<pre>model_post_init() (aiogram.methods.set_sticker_mask_modulan</pre>	
method), 301	method), 24
<pre>model_post_init() (aiogram.methods.set_sticker_positiomode_ke</pre>	
method), 302	method), 24
<pre>model_post_init() (aiogram.methods.set_sticker_set_thumoderali).</pre>	
<i>method</i>), 304	method), 24
<pre>model_post_init() (aiogram.methods.set_sticker_set_tithmSdreStic</pre>	poss et Title() (aiogram.types.business_connection.BusinessConnec

method), 25	<i>method</i>), 93
<pre>model_post_init()</pre>	(aiogram.types.business_intro.Businessbuelo_post_init() (aiogram.types.chat_member_updated.ChatMember
method), 25	<i>method</i>), 106
<pre>model_post_init()</pre>	(aiogram.types.business_location.BusindesLapasioninit() (aiogram.types.chat_permissions.ChatPermissions
<i>method</i>), 26	<i>method</i>), 113
, .	(aiogram.types.business_messages_dmlodell_BpsistessiMessagesIDebatedtypes.chat_photo.ChatPhoto
method), 26	method), 113
, · ·	(aiogram.types.business_opening_homodeleilsiposstQpirriitg)H(airegram.types.chat_shared.ChatShared
method), 27	method), 114
	(aiogram.types.business_opening_homosdeiltepossltBistints(SQppingrghlaypelnthosedn_inline_result.ChosenInlineRe
method), 27	method), 216
, · ·	(aiogram.types.callback_game.Callbacdchampost_init() (aiogram.types.contact.Contact
method), 287	
	(aiogram.types.callback_query.Callb mcdQile_post_init() (aiogram.types.dice.Dice method),
method), 28	115
	(aiogram.types.chat.Chatmethod), model_post_init() (aiogram.types.document.Document
32	<i>method</i>), 116
<pre>model_post_init() method), 45</pre>	(aiogram.types.chat_administrator_rigbde:Chpotsdministratordentials.EncryptedCreation method), 267
<pre>model_post_init()</pre>	(aiogram.types.chat_boost.ChatBoostmodel_post_init() (aiogram.types.encrypted_passport_element.Encrypt
method), 45	<i>method</i>), 268
, · ·	(aiogram.types.chat_boost_added.Chaddedstadstadstadstadstadstadstadstadstadsta
<i>method</i>), 46	method), 539
, · ·	(aiogram.types.chat_boost_removed.@bduBoopsoBamoned () (aiogram.types.external_reply_info.ExternalReplyInf
method), 46	method), 118
	(aiogram.types.chat_boost_source.ChagBebspostceinit() (aiogram.types.file.File method),
method), 47	119
	(aiogram.types.chat_boost_source_giftodedlendbattBinitSO)(aeOgiftContlepes.force_reply.ForceReply
method), 47	<i>method</i>), 120
<pre>model_post_init()</pre>	(aiogram.types.chat_boost_source_ginedwdy_fclastBinistS6)u(ca6ginamstypes.forum_topic.ForumTopic
<i>method</i>), 48	<i>method</i>), 120
<pre>model_post_init()</pre>	(aiogram.types.chat_boost_source_preodal_fbstBinst\$6)/aeiBgraniutypes.forum_topic_closed.ForumTopicClos
method), 48	<i>method</i>), 120
<pre>model_post_init()</pre>	(aiogram.types.chat_boost_updated.GladBbopdstdated t() (aiogram.types.forum_topic_created.ForumTopicCre
method), 49	<i>method</i>), 121
<pre>model_post_init()</pre>	(aiogram.types.chat_invite_link.ChathodeeLipost_init() (aiogram.types.forum_topic_edited.ForumTopicEdite
method), 49	<i>method</i>), 121
, .	(aiogram.types.chat_join_request.ChaddenRepart_init() (aiogram.types.forum_topic_reopened.ForumTopicRe
<i>method</i>), 81	method), 122
, .	(aiogram.types.chat_location.ChatLomotleil_post_init() (aiogram.types.game.Game
method), 87	method), 287
, .	(aiogram.types.chat_member.ChatMandderl_post_init() (aiogram.types.game_high_score.GameHighScore
method), 88	(alogram.types.chai_member.chaimeinghscore method), 288
	(aiogram.types.chat_member_admini wodw) , GostMembt (Administratorypes.general_forum_topic_hidden.General
method), 89	method), 122
-	(aiogram.types.chat_member_banneduGdalMpodserBainted) (aiogram.types.general_forum_topic_unhidden.Gene
method), 90	<i>method</i>), 122
-	(aiogram.types.chat_member_left.Ch ut)de hb poke ftinit() (aiogram.types.giveaway.Giveaway
method), 90	<i>method</i>), 123
-	$(aiogram.types.chat_member_member_member_member_indentifier (aiogram.types.giveaway_completed.GiveawayCompleted.Giveaw$
method), 91	<i>method</i>), 123
-	$(aiogram.types.chat_member_owner.{\tt mbdeW} epdst \underline{Oirnier}() (aiogram.types.giveaway_created.GiveawayCreated) (aiogram.types.giveaway_created) (aiogram.types) (a$
method), 91	<i>method</i>), 124
model post init()	(aiogram.types.chat_member_restrictaddenaty6staberthes(r)(teidogram.types.giveaway_winners.GiveawayWinners

m	ethod), 12	24 method	1), 257
model_pos	t_init() (aiogram.types.inaccessible_message ulude de .poiste <u>M</u> a	<pre>site@ (aiogram.types.input_invoice_message_content.Input</pre>
	ethod), 12		
model_pos	t_init() (aiogram.types.inline keyboard butt mobelingKeyboim	dB(the aiogram.types.input_location_message_content.Input_location_message
-	ethod), 12		
	, .) (aiogram.types.inline_keyboard_mar knqd&i l <u>inpeKstyb</u> in	
	ethod), 12		
	, .		it() (aiogram.types.input_media_animation.InputMediaA
	ethod, 2		
	, .		<pre>htt() (aiogram.types.input_media_audio.InputMediaAudio</pre>
	ethod, 2		
			r yReJulaiogiche n.types.input_media_document.InputMediaD
	ethod), 22		
	, .		Resultating in m.types.input_media_photo.InputMediaPhoto
	ethod, 22		
	, .		i), 192 i irt QukajaBexultCyndreilApud<u>io</u>nedia_video.InputMediaVideo
	ethod, 2		
			t iltijue@ugryRe.typteSaapedDocsonge t_content.InputMessage
	ethod), 22		
-		() (aiogram.types.inline_query_result_cmolled_gifstrline	
	ethod), 2		
-			i f.tx(I):(eQugercyRespl:tCimphet<u>dMp</u>keg4Gsis fage_content.InputTex
	ethod), 2		
-			int Que ay Bescult Cypelse il Photoenue_message_content. Input
	ethod), 2		
		() (aiogram.types.inline_query_result_c mobled_spicst er. in	
	ethod), 2		
		() (aiogram.types.inline_query_result_c mobled_ypdsc.<u>l</u>nb	
	ethod), 2		
model_pos	t_init(() (aiogram.types.inline_query_result_c molled_poise.Irih	nieQ)(wiRgsaltGyphsdWybaard_button_poll_type.Keyboard
m	ethod), 2.	37 method	<i>l</i>), 135
model_pos	t_init(() (aiogram.types.inline_query_result_c madeA_pbistQim	n i:yR∂s(di6gmtmct ypes.keyboard_button_request_chat.Keybo
m	ethod), 2.	39 method	<i>l</i>), 137
model_pos	t_init() (aiogram.types.inline_query_result_dwodede_pdshina@	hiver/yRasinolgDoncutypenst keyboard_button_request_user.Keybo
m	ethod), 24	41 method	<i>l</i>), 137
model_pos	t_init() (aiogram.types.inline_query_result_g node.l nl pasQ uein	x Ref)l(Giangu am.types.keyboard_button_request_users.Keyb
m	ethod), 24	42 method	1), 138
model_pos	t_init() (aiogram.types.inline_query_result_gij6theline@useryRe	stall@lfaiogram.types.labeled price.LabeledPrice
	ethod), 2		
			aryRefultIgoantinpes.link_preview_options.LinkPreviewOp
-	ethod), 24		
	, .	() (aiogram.types.inline_query_result_model_gifbhlinia	
-	ethod, 24		
	, .	() (aiogram.types.inline_query_result_phode_InfposQuar	
-	ethod), 2:		
		() (aiogram.types.inline_query_result_v node_Infjoeg uar	
-	ethod, 2		
			r), 204 v Re{})({&idgr am.types.maybe_inaccessible_message.Maybe.
-			
	ethod, 2		
-		() (aiogram.types.inline_query_result_vmddeln/pasQuery	
	ethod, 2		
			xiyRes(disBnanntypes.menu_button_commands.MenuButton
	<i>ethod</i>), 2:		
mode1_pos	τ_init() (aiogram.types.input_contact_messageodentepo.shpat	<mark>ርድርሲን (thilessanget</mark> Sp esemt enu_button_default.MenuButtonDe

	method), 14	2 198	
model_	_post_init()	(aiogram.types.menu_button_web_appoWehuBastonW	ibApp(aiogram.types.poll_answer.PollAnswer
	method), 14		
model_	_post_init()	(aiogram.types.message.Message model_post_in	it() (aiogram.types.poll_option.PollOption
	method), 14	9 method	<i>I</i>), 199
model_	_post_init()	(aiogram.types.message_auto_delete_modelchoose_did	žlæss)ugañagaaDre.lypeTsinner<u>C</u>hangod t_query.PreCheckoutQua
	method), 19	2 method	<i>I</i>), 280
model_	_post_init()		it() (aiogram.types.proximity_alert_triggered.ProximityA
	method), 19		
model_		(aiogram.types.message_id.MessageIndodel_post_in	
	method), 19		
model_		(aiogram.types.message_origin.Messmed@ligpost_in	
	method), 19		
model_			int())d(wiegram.types.reaction_type_custom_emoji.Reaction_
	method), 19		
model_			<pre>html() (aiogram.types.reaction_type_emoji.ReactionTypeEn</pre>
	method), 19		
model_			Origin(Hilaghanhitsepes.reply_keyboard_markup.ReplyKeybo
madal	method), 19		
moder_	_post_init() method), 19		<pre>ide() (aiogram.types.reply_keyboard_remove.ReplyKeyboa 0, 203</pre>
model	,,		sageRaiogonhaypeblpelpledparameters.ReplyParameters
mouer_	method), 19		
model	, · ·		int fightlipgated.types.response_parameters.ResponseParan
mouci_	method), 19		
model	· · · ·		it() (aiogram.types.sent_web_app_message.SentWebApp
	method), 27		
model_	· · · ·	(aiogram.types.passport_data.Passpanddeda_post_in	
	method), 26		
model_	_post_init()	(aiogram.types.passport_element_ermodels.posEleim	ant Entoniogram.types.shipping_address.ShippingAddress
	method), 27	0 method	(), 281
model_	_post_init()	(aiogram.types.passport_element_ermodedtap@std_Pa	isp6) (ElegranEtypeDatipFielg_option.ShippingOption
	method), 27		
model_	_post_init()	(aiogram.types.passport_element_ermodele_posspoint	Elt MentleogonFiltypes.shipping_query.ShippingQuery
	method), 27		
model_	-	(aiogram.types.passport_element_ermoodfeles.prostport	
	method), 27		
model_	-	(aiogram.types.passport_element_ermodfebnposide_Pa	
	method), 27		
model_	-	(aiogram.types.passport_element_ermodedvepros_tiden	
modol	method), 27		1), 205 iEl@n(enioErnonSylfes .successful_payment.SuccessfulPayme
moder_	_post_init() method), 27		
model			ij, 200 ie:RosspiogaEdentypeEssoitFra<u>n</u>islinionFidey_ chosen_chat.Sw
mouci_	_pose_infic() method), 27		
model		(aiogram.types.passport_element_ermoderluplasionif	
mouci_	method), 27		
model		(aiogram.types.passport_element_ermodelspecified.f	
	method), 27		
model_		(aiogram.types.passport_file.Passportfcidel_post_in	
-	method), 27		······································
model_			it() (aiogram.types.user_chat_boosts.UserChatBoosts
	method), 19		
model_	_post_init()	(aiogram.types.poll.Poll method), model_post_in	it() (aiogram.types.user_profile_photos.UserProfilePhoto

method), 209 aiogram.enums.message_entity_type, 467 model_post_init() (aiogram.types.user_shared.UserShared aiogram.enums.message_origin_type,468 method), 209 aiogram.enums.parse_mode, 468 model_post_init() (aiogram.types.users_shared.UsersSharedaiogram.enums.passport_element_error_type, method), 210 468 model_post_init() aiogram.enums.poll_type, 469 (aiogram.types.venue.Venue method). 211 aiogram.enums.reaction_type_type, 469 model_post_init() aiogram.enums.sticker_format, 469 (aiogram.types.video.Video method), 211 aiogram.enums.sticker_type, 469 model_post_init() (aiogram.types.video_chat_ended.VideoClaivEgraml.enums.topic_icon_color, 470 method), 212 aiogram.enums.update_type, 470 model_post_init() (aiogram.types.video_chat_participants_inaiteghtinleefdepRintspättsInvited method), 212 aiogram.handlers.callback_query, 544 model_post_init() (aiogram.types.video_chat_scheduled.Vidæid/gut/Sm/mett/hd/ds.add_sticker_to_set, 288 aiogram.methods.answer_callback_query, method), 213 model_post_init() (aiogram.types.video_chat_started.VideoChatStarted aiogram.methods.answer_inline_query, 433 method), 213 model_post_init() (aiogram.types.video_note.VideoNote aiogram.methods.answer_pre_checkout_query, method), 213 442 model_post_init() (aiogram.types.voice.Voice aiogram.methods.answer_shipping_query, method), 214 443 model_post_init() (aiogram.types.web_app_data.WebAppDataiogram.methods.answer_web_app_query, 436 aiogram.methods.approve_chat_join_request, *method*), 214 model_post_init() (aiogram.types.web_app_info.WebAppInfo 308 method), 215 aiogram.methods.ban_chat_member, 309 model_post_init() (aiogram.types.webhook_info.WebhookInfaiogram.methods.ban_chat_sender_chat, 311 method), 286 aiogram.methods.close, 312 model_post_init() (aiogram.types.write_access_allowed.WriteAlogaxanalhuenteabds.close_forum_topic, 313 method), 215 aiogram.methods.close_general_forum_topic, model_post_init() (aiogram.utils.web_app.WebAppChat 314 *method*), 563 aiogram.methods.copy_message, 315 model_post_init() (aiogram.utils.web_app.WebAppInitData aiogram.methods.copy_messages, 317 aiogram.methods.create_chat_invite_link, method), 560 319 model_post_init() (aiogram.utils.web_app.WebAppUser method), 562 aiogram.methods.create_forum_topic, 320 module aiogram.methods.create_invoice_link,444 aiogram.dispatcher.flags, 542 aiogram.methods.create_new_sticker_set, aiogram.enums.bot_command_scope_type, 457 289 aiogram.enums.chat_action, 457 aiogram.methods.decline_chat_join_request, aiogram.enums.chat_boost_source_type, 458 321 aiogram.enums.chat_member_status, 458 aiogram.methods.delete_chat_photo, 322 aiogram.methods.delete_chat_sticker_set, aiogram.enums.chat_type, 459 aiogram.enums.content_type, 459 323 aiogram.methods.delete_forum_topic, 324 aiogram.enums.currency, 461 aiogram.enums.dice_emoji, 464 aiogram.methods.delete_message, 420 aiogram.enums.encrypted_passport_element, aiogram.methods.delete_messages, 422 464 aiogram.methods.delete_my_commands, 325 aiogram.enums.inline_query_result_type, aiogram.methods.delete_sticker_from_set, 465 291 aiogram.enums.input_media_type, 465 aiogram.methods.delete_sticker_set, 292 aiogram.enums.keyboard_button_poll_type_type, aiogram.methods.delete_webhook,450 aiogram.methods.edit_chat_invite_link, 466 aiogram.enums.mask_position_point, 466 326 aiogram.enums.menu_button_type, 467 aiogram.methods.edit_forum_topic, 328

aiogram.methods.edit_general_forum_topic, aiogram.methods.revoke_chat_invite_link, 359 329 aiogram.methods.edit_message_caption, 423 aiogram.methods.send_animation, 361 aiogram.methods.edit_message_live_location, aiogram.methods.send_audio, 363 424 aiogram.methods.send_chat_action, 366 aiogram.methods.edit_message_media, 426 aiogram.methods.send_contact, 368 aiogram.methods.edit_message_reply_markup, aiogram.methods.send_dice,370 428 aiogram.methods.send_document, 372 aiogram.methods.edit_message_text, 429 aiogram.methods.send_game, 438 aiogram.methods.export_chat_invite_link, aiogram.methods.send_invoice, 447 330 aiogram.methods.send_location, 375 aiogram.methods.forward_message, 331 aiogram.methods.send_media_group, 377 aiogram.methods.forward_messages, 333 aiogram.methods.send_message, 379 aiogram.methods.get_business_connection, aiogram.methods.send_photo, 381 334 aiogram.methods.send_poll,384 aiogram.methods.get_chat, 335 aiogram.methods.send_sticker, 296 aiogram.methods.send_venue, 387 aiogram.methods.get_chat_administrators, aiogram.methods.send_video, 390 336 aiogram.methods.send_video_note, 392 aiogram.methods.get_chat_member, 337 aiogram.methods.get_chat_member_count, aiogram.methods.send_voice, 395 338 aiogram.methods.set_chat_administrator_custom_title, aiogram.methods.get_chat_menu_button, 339 397 aiogram.methods.get_custom_emoji_stickers, aiogram.methods.set_chat_description, 399 aiogram.methods.set_chat_menu_button, 400 293 aiogram.methods.get_file, 340 aiogram.methods.set_chat_permissions, 401 aiogram.methods.get_forum_topic_icon_stickers,aiogram.methods.set_chat_photo, 402 341 aiogram.methods.set_chat_sticker_set, 403 aiogram.methods.get_game_high_scores, 437 aiogram.methods.set_chat_title,404 aiogram.methods.get_me, 342 aiogram.methods.set_custom_emoji_sticker_set_thumbnail aiogram.methods.get_my_commands, 343 298 aiogram.methods.get_my_default_administrator_raighgram.methods.set_game_score, 440 344 aiogram.methods.set_message_reaction, 405 aiogram.methods.get_my_description, 345 aiogram.methods.set_my_commands, 407 aiogram.methods.get_my_name, 346 aiogram.methods.set_my_default_administrator_rights, aiogram.methods.get_my_short_description, 408 aiogram.methods.set_my_description, 409 347 aiogram.methods.get_sticker_set, 294 aiogram.methods.set_my_name, 410 aiogram.methods.get_updates, 451 aiogram.methods.set_my_short_description, aiogram.methods.get_user_chat_boosts, 347 411 aiogram.methods.get_user_profile_photos, aiogram.methods.set_passport_data_errors, 348 455 aiogram.methods.get_webhook_info,453 aiogram.methods.set_sticker_emoji_list, aiogram.methods.hide_general_forum_topic, 299 349 aiogram.methods.set_sticker_keywords, 300 aiogram.methods.leave_chat,350 aiogram.methods.set_sticker_mask_position, aiogram.methods.log_out, 351 301aiogram.methods.pin_chat_message, 352 aiogram.methods.set_sticker_position_in_set, aiogram.methods.promote_chat_member, 353 302 aiogram.methods.reopen_forum_topic, 356 aiogram.methods.set_sticker_set_thumbnail, aiogram.methods.reopen_general_forum_topic, 303 357 aiogram.methods.set_sticker_set_title, aiogram.methods.replace_sticker_in_set, 305 295 aiogram.methods.set_webhook, 453 aiogram.methods.restrict_chat_member, 358

aiogram.methods.stop_message_live_location, aiogram.types.chat_boost_source_gift_code, 431 47 aiogram.methods.stop_poll, 432 aiogram.types.chat_boost_source_giveaway, aiogram.methods.unban_chat_member, 412 47 aiogram.methods.unban_chat_sender_chat, aiogram.types.chat_boost_source_premium, 414 48 aiogram.methods.unhide_general_forum_topic, aiogram.types.chat_boost_updated, 49 aiogram.types.chat_invite_link,49 415 aiogram.methods.unpin_all_chat_messages, aiogram.types.chat_join_request, 50 aiogram.types.chat_location,87 416 aiogram.methods.unpin_all_forum_topic_messagesaiogram.types.chat_member,87 417 aiogram.types.chat_member_administrator, aiogram.methods.unpin_all_general_forum_topic_messages, aiogram.types.chat_member_banned,90 418 aiogram.methods.unpin_chat_message, 419 aiogram.types.chat_member_left,90 aiogram.methods.upload_sticker_file, 306 aiogram.types.chat_member_member,91 aiogram.types.animation,17 aiogram.types.chat_member_owner,91 aiogram.types.audio, 18 aiogram.types.chat_member_restricted, 92 aiogram.types.birthdate, 19 aiogram.types.chat_member_updated,93 aiogram.types.bot_command, 19 aiogram.types.chat_permissions, 112 aiogram.types.bot_command_scope, 20 aiogram.types.chat_photo,113 aiogram.types.bot_command_scope_all_chat_administgrammorsypes.chat_shared, 114 20 aiogram.types.chosen_inline_result, 215 aiogram.types.bot_command_scope_all_group_chatsiogram.types.contact, 114 aiogram.types.dice, 115 21 aiogram.types.bot_command_scope_all_private_chamicsgram.types.document, 116 21 aiogram.types.encrypted_credentials, 267 aiogram.types.bot_command_scope_chat, 22 aiogram.types.encrypted_passport_element, aiogram.types.bot_command_scope_chat_administrator 22 aiogram.types.error_event, 539 aiogram.types.bot_command_scope_chat_member, aiogram.types.external_reply_info, 117 23 aiogram.types.file, 119 aiogram.types.bot_command_scope_default, aiogram.types.force_reply, 119 aiogram.types.forum_topic, 120 23 aiogram.types.bot_description, 24 aiogram.types.forum_topic_closed, 120 aiogram.types.bot_name, 24 aiogram.types.forum_topic_created, 121 aiogram.types.bot_short_description, 24 aiogram.types.forum_topic_edited, 121 aiogram.types.business_connection, 25 aiogram.types.forum_topic_reopened, 122 aiogram.types.business_intro,25 aiogram.types.game, 287 aiogram.types.business_location, 26 aiogram.types.game_high_score, 288 aiogram.types.business_messages_deleted, aiogram.types.general_forum_topic_hidden, 26 122 aiogram.types.business_opening_hours, 27 aiogram.types.general_forum_topic_unhidden, aiogram.types.business_opening_hours_interval, 122 27 aiogram.types.giveaway, 122 aiogram.types.callback_game, 287 aiogram.types.giveaway_completed, 123 aiogram.types.callback_query, 28 aiogram.types.giveaway_created, 124 aiogram.types.chat, 29 aiogram.types.giveaway_winners, 124 aiogram.types.chat_administrator_rights, aiogram.types.inaccessible_message, 125 43 aiogram.types.inline_keyboard_button, 125 aiogram.types.chat_boost,45 aiogram.types.inline_keyboard_markup, 127 aiogram.types.chat_boost_added, 46 aiogram.types.inline_query, 216 aiogram.types.chat_boost_removed, 46 aiogram.types.inline_query_result, 218 aiogram.types.chat_boost_source, 47

aiogram.types.inline_query_result_article, aiogram.types.input_media_video, 132 aiogram.types.input_message_content, 261 219 aiogram.types.input_sticker, 263 aiogram.types.inline_query_result_audio, 220 aiogram.types.input_text_message_content, aiogram.types.inline_query_result_cached_audio, 261221 aiogram.types.input_venue_message_content, aiogram.types.inline_query_result_cached_document,262 223 aiogram.types.invoice, 278 aiogram.types.inline_query_result_cached_gif, aiogram.types.keyboard_button, 134 225 aiogram.types.keyboard_button_poll_type, aiogram.types.inline_query_result_cached_mpeg4_gifl,35 aiogram.types.keyboard_button_request_chat, 227 aiogram.types.inline_query_result_cached_photo, 135 aiogram.types.keyboard_button_request_user, 229 aiogram.types.inline_query_result_cached_sticker, 137 231 aiogram.types.keyboard_button_request_users, aiogram.types.inline_query_result_cached_video, 138 aiogram.types.labeled_price, 279 233 aiogram.types.inline_query_result_cached_voiceaiogram.types.link_preview_options, 139 236 aiogram.types.location, 140 aiogram.types.inline_query_result_contact, aiogram.types.login_url,140 aiogram.types.mask_position, 264 238 aiogram.types.maybe_inaccessible_message, aiogram.types.inline_query_result_document, 239 141 aiogram.types.inline_query_result_game, aiogram.types.menu_button, 141 241 aiogram.types.menu_button_commands, 142 aiogram.types.inline_query_result_gif, aiogram.types.menu_button_default, 142 242 aiogram.types.menu_button_web_app, 143 aiogram.types.inline_query_result_location, aiogram.types.message, 143 244 aiogram.types.message_auto_delete_timer_changed, aiogram.types.inline_query_result_mpeg4_gif, 191 246 aiogram.types.message_entity, 192 aiogram.types.inline_query_result_photo, aiogram.types.message_id, 193 249 aiogram.types.message_origin, 193 aiogram.types.inline_query_result_venue, aiogram.types.message_origin_channel, 193 251 aiogram.types.message_origin_chat, 194 aiogram.types.inline_query_result_video, aiogram.types.message_origin_hidden_user, 252 195 aiogram.types.inline_query_result_voice, aiogram.types.message_origin_user, 195 aiogram.types.message_reaction_count_updated, 254 aiogram.types.inline_query_results_button, 196 256 aiogram.types.message_reaction_updated, aiogram.types.input_contact_message_content, 196 257 aiogram.types.order_info, 279 aiogram.types.input_file, 127 aiogram.types.passport_data, 269 aiogram.types.passport_element_error, 269 aiogram.types.input_invoice_message_content, 257aiogram.types.passport_element_error_data_field, aiogram.types.input_location_message_content, 270 260 aiogram.types.passport_element_error_file, aiogram.types.input_media, 128 271 aiogram.types.input_media_animation, 128 aiogram.types.passport_element_error_files, aiogram.types.input_media_audio, 129 272 aiogram.types.input_media_document, 130 aiogram.types.passport_element_error_front_side, aiogram.types.input_media_photo, 132 273

```
aiogram.types.passport_element_error_reverse_saidegram.types.web_app_data, 214
                                                    aiogram.types.web_app_info, 215
    273
aiogram.types.passport_element_error_selfie,
                                                    aiogram.types.webhook_info, 286
    274
                                                    aiogram.types.write_access_allowed, 215
aiogram.types.passport_element_error_transition_afighe_m.types.birthdate.Birthdate attribute), 19
                                               MOUTH (aiogram.enums.mask position point.MaskPositionPoint
    275
aiogram.types.passport_element_error_translation_fithelsute), 466
    276
                                               mpeg4_duration(aiogram.types.inline_query_result_mpeg4_gif.InlineQu
aiogram.types.passport_element_error_unspecified, attribute), 248
    277
                                               mpeg4_file_id(aiogram.types.inline_query_result_cached_mpeg4_gif.In
aiogram.types.passport_file,278
                                                        attribute), 229
aiogram.types.photo_size, 197
                                               MPEG4_GIF (aiogram.enums.inline_query_result_type.InlineQueryResultTy
aiogram.types.poll, 198
                                                        attribute), 465
aiogram.types.poll_answer, 199
                                               mpeg4_height (aiogram.types.inline_query_result_mpeg4_gif.InlineQuery
aiogram.types.poll_option, 199
                                                        attribute), 248
aiogram.types.pre_checkout_query, 280
                                               mpeg4_url (aiogram.types.inline_query_result_mpeg4_gif.InlineQueryRes
aiogram.types.proximity_alert_triggered,
                                                        attribute), 248
    199
                                               mpeg4_width(aiogram.types.inline_query_result_mpeg4_gif.InlineQueryH
aiogram.types.reaction_count, 200
                                                        attribute), 248
aiogram.types.reaction_type, 200
                                               MUR (aiogram.enums.currency.Currency attribute), 462
aiogram.types.reaction_type_custom_emoji,
                                               MVR (aiogram.enums.currency.Currency attribute), 462
                                               MXN (aiogram.enums.currency.Currency attribute), 463
    201
aiogram.types.reaction_type_emoji, 201
                                               MY_CHAT_MEMBER (aiogram.enums.update_type.UpdateType
aiogram.types.reply_keyboard_markup, 202
                                                        attribute), 471
aiogram.types.reply_keyboard_remove, 203
                                               my_chat_member
                                                                 (aiogram.types.update.Update
                                                                                              at-
aiogram.types.reply_parameters, 203
                                                        tribute), 285
aiogram.types.response_parameters, 204
                                               MYR (aiogram.enums.currency.Currency attribute), 463
aiogram.types.sent_web_app_message, 263
                                               MZN (aiogram.enums.currency.Currency attribute), 463
aiogram.types.shared_user, 205
                                               Ν
aiogram.types.shipping_address, 281
aiogram.types.shipping_option, 281
                                               name (aiogram.methods.add_sticker_to_set.AddStickerToSet
aiogram.types.shipping_query, 282
                                                        attribute), 289
aiogram.types.sticker, 265
                                               name (aiogram.methods.create chat invite link.CreateChatInviteLink
aiogram.types.sticker_set,266
                                                        attribute), 319
aiogram.types.story, 205
                                                name (aiogram.methods.create_forum_topic.CreateForumTopic
aiogram.types.successful_payment, 283
                                                        attribute), 320
aiogram.types.switch_inline_query_chosen_chame (aiogram.methods.create_new_sticker_set.CreateNewStickerSet
    206
                                                        attribute), 290
aiogram.types.text_quote, 207
                                               name (aiogram.methods.delete_sticker_set.DeleteStickerSet
aiogram.types.update, 284
                                                        attribute), 292
aiogram.types.user, 207
                                               name (aiogram.methods.edit_chat_invite_link.EditChatInviteLink
aiogram.types.user_chat_boosts, 209
                                                        attribute), 327
aiogram.types.user_profile_photos, 209
                                               name (aiogram.methods.edit_forum_topic.EditForumTopic
aiogram.types.user_shared, 209
                                                        attribute), 328
aiogram.types.users_shared, 210
                                               name (aiogram.methods.edit_general_forum_topic.EditGeneralForumTopic
aiogram.types.venue, 210
                                                        attribute), 329
aiogram.types.video,211
                                               name (aiogram.methods.get_sticker_set.GetStickerSet at-
aiogram.types.video_chat_ended, 212
                                                        tribute), 294
aiogram.types.video_chat_participants_invitede (aiogram.methods.replace_sticker_in_set.ReplaceStickerInSet
    212
                                                        attribute), 295
aiogram.types.video_chat_scheduled, 212
                                               name (aiogram.methods.set_custom_emoji_sticker_set_thumbnail.SetCustor
aiogram.types.video_chat_started, 213
                                                        attribute), 298
aiogram.types.video_note, 213
                                                          (aiogram.methods.set_my_name.SetMyName
                                               name
aiogram.types.voice, 214
                                                        attribute), 410
```

<pre>name (aiogram.methods.set_sticker_set_thumbnail.SetSticke attribute), 303</pre>	e rSexThhadompiloto (aiogram.types.message.Message at- tribute), 147
<pre>name (aiogram.methods.set_sticker_set_title.SetStickerSetT attribute), 305</pre>	iMEW_CHAT_TITLE (aiogram.enums.content_type.ContentType attribute), 460
<pre>name (aiogram.types.bot_name.BotName attribute), 24</pre>	<pre>new_chat_title (aiogram.types.message.Message at-</pre>
name (aiogram.types.chat_invite_link.ChatInviteLink at- tribute), 49	tribute), 147 new_reaction (aiogram.types.message_reaction_updated.MessageReaction)
name (aiogram.types.forum_topic.ForumTopic attribute), 120	attribute), 197 next_offset (aiogram.methods.answer_inline_query.AnswerInlineQuery
<pre>name (aiogram.types.forum_topic_created.ForumTopicCreated.For</pre>	
<pre>name (aiogram.types.forum_topic_edited.ForumTopicEdited attribute), 121</pre>	
name (aiogram.types.order_info.OrderInfo attribute), 279	
name (aiogram.types.sticker_set.StickerSet attribute), 266	
<pre>need_email(aiogram.methods.create_invoice_link.Create</pre>	0
need_email(aiogram.methods.send_invoice.SendInvoice attribute), 449	tribute), 451
<pre>need_email (aiogram.types.input_invoice_message_contex attribute), 259</pre>	nb Fusel Wabge Massing and Gassiget user_profile_photos.GetUserProfilePhotos attribute), 348
need_name (aiogram.methods.create_invoice_link.CreateIn attribute), 446	ⁿ <i>G</i> ffS <i>k</i> ^{<i>i</i>} ^{<i>k</i>} (aiogram.types.inline_query.InlineQuery at- tribute), 216
need_name (aiogram.methods.send_invoice.SendInvoice attribute), 449	offset (aiogram.types.message_entity.MessageEntity at- tribute), 192
	t. GRAUHO getife.Methods.Gnstlett_pre_checkout_query.AnswerPreCheckoutQue
need_phone_number(aiogram.methods.create_invoice_lin	attribute), 442 ⁿ &K(&UgHttm:MeHiblis.answer_shipping_query.AnswerShippingQuery
attribute), 446	attribute), 443
<pre>need_phone_number (aiogram.methods.send_invoice.Send attribute), 449</pre>	dureichat_member (aiogram.types.chat_member_updated.ChatMemberUp attribute), 94
	3516<u>0</u>46864700H(Av0jGeMessus Message_reaction_updated.MessageReactic attribute), 197
need_shipping_address	
(aiogram.methods.create_invoice_link.CreateInvo	
attribute), 446 need_shipping_address	one_time_keyboard (aiogram.types.reply_keyboard_markup.ReplyKeybo attribute), 202
(aiogram.methods.send_invoice.SendInvoice attribute), 449	only_if_banned (aiogram.methods.unban_chat_member.UnbanChatMem attribute), 413
need_shipping_address (aiogram.types.input_invoice_message_content.li	only_new_members (aiogram.types.giveaway.Giveaway nputInvoiceMessageContent
attribute), 260 needs_repainting(aiogram.methods.create_new_sticker	only_new_members(aiogram.types.giveaway_winners.GiveawayWinners set.CreateWstickerSet
attribute), 290	open_period (aiogram.methods.send_poll.SendPoll at-
needs_repainting (aiogram.types.sticker.Sticker attribute), 266	tribute), 385
	open_period (aiogram.types.poll.Poll attribute), 198 GpetMang_etGPsetadogram.types.business_opening_hours.BusinessOpening
attribute), 94	attribute). 27
NEW_CHAT_MEMBERS (aiogram.enums.content_type.Content attribute), 460	<pre>Compening_minute (aiogram.types.business_opening_hours_interval.Busine attribute), 27</pre>
new_chat_members (aiogram.types.message.Message attribute), 147	option_ids (aiogram.types.poll_answer.PollAnswer at-
NEW_CHAT_PHOTO (aiogram.enums.content_type.ContentTy	tribute), 199
attribute), 460	Poptions (aiogram.methods.send_poll.SendPoll at- tribute), 385

options (aiogram.types.poll.Poll attribute), 198 order_info(aiogram.types.pre_checkout_query.PreCheck	parse_mode (aiogram.types.inline_query_result_mpeg4_gif.InlineQueryRe
attribute), 280	parse_mode (aiogram.types.inline_query_result_photo.InlineQueryResultF
order_info(aiogram.types.successful_payment.Successfu	
attribute), 283	$\verb"parse_mode" (a iogram.types.inline_query_result_video.InlineQueryResultWideo.InlineQuer$
OrderInfo (class in aiogram.types.order_info), 279	attribute), 254
attribute), 117	nparse_mode(aiogram.types.inline_query_result_voice.InlineQueryResultV attribute), 255
P	parse_mode (aiogram.types.input_media_animation.InputMediaAnimation attribute), 129
PAB (aiogram.enums.currency.Currency attribute), 463	<pre>parse_mode(aiogram.types.input_media_audio.InputMediaAudio attribute), 130</pre>
pack() (aiogram.filters.callback_data.CallbackData method), 492	parse_mode (aiogram.types.input_media_document.InputMediaDocument
parse_mode (<i>aiogram.methods.copy_message.CopyMessa</i>	
attribute), 316	<pre>parse_mode(aiogram.types.input_media_photo.InputMediaPhoto</pre>
parse_mode(aiogram.methods.edit_message_caption.Edit	Message Capitribute), 132
attribute), 423 parse_mode (aiogram.methods.edit_message_text.EditMes	parse_mode (aiogram.types.input_media_video.InputMediaVideo
attribute), 430	parse_mode (aiogram.types.input_text_message_content.InputTextMessage
parse_mode (aiogram.methods.send_animation.SendAnimatio	ation attribute), 262
attribute), 362	parse_webapp_init_data() (in module
parse_mode (aiogram.methods.send_audio.SendAudio	aiogram.utils.web_app), 559
attribute), 364	ParseMode (class in aiogram.enums.parse_mode), 468 PANSPORT (aiogram.enums.encrypted_passport_element.EncryptedPassport_
attribute), 373	attribute), 464
<pre>parse_mode(aiogram.methods.send_message.SendMessag attribute), 380</pre>	PASSPORT_DATA (aiogram.enums.content_type.ContentType attribute), 460
parse_mode (aiogram.methods.send_photo.SendPhoto	passport_data (aiogram.types.message.Message
attribute), 382	attribute), 148
parse_mode (aiogram.methods.send_video.SendVideo attribute), 391	PASSPORT_REGISTRATION (aiogram.enums.encrypted_passport_element.EncryptedPassport_
parse_mode (aiogram.methods.send_voice.SendVoice	attribute), 465
attribute), 396	PassportData (class in aiogram.types.passport_data),
<pre>parse_mode(aiogram.types.inline_query_result_audio.Inl</pre>	ineQueryReMutAudio
attribute), 221	PassportElementError (class in
<pre>parse_mode(aiogram.types.inline_query_result_cached_a attribute), 223</pre>	269
parse_mode (aiogram.types.inline_query_result_cached_d attribute), 225	Bannan:Infine (atoparte FRE Still Caster and Caster) alogram.types.passport_element_error_data_field),
<pre>parse_mode(aiogram.types.inline_query_result_cached_g</pre>	if.InlineQuer9ResultCachedGif PassportElementErrorFile (class in
<pre>parse_mode(aiogram.types.inline_query_result_cached_n</pre>	npeg4_gif. HinneONetworestateDetteeMp ugy4Gafror_file), 271
<pre>parse_mode(aiogram.types.inline_query_result_cached_p</pre>	Rozennant Eleveretennou (class in
attribute), 231	aiogram.types.passport_element_error_files),
<pre>parse_mode(aiogram.types.inline_query_result_cached_v</pre>	PassportElementErrorFrontSide (class in
<pre>parse_mode(aiogram.types.inline_query_result_cached_v</pre>	oice.Inline GastyReSyncsagssport_Ce lement_error_front_side), 273
<pre>parse_mode(aiogram.types.inline_query_result_document</pre>	PARAPORE FURNING FOR THE PARAPORE SIDE (class in
attribute), 241	aiogram.types.passport_element_error_reverse_side),
<pre>parse_mode(aiogram.types.inline_query_result_gif.Inline attribute), 244</pre>	QueryResuttGif PassportElementErrorSelfie (class in
<i>anniome), 2</i> 77	aiogram.types.passport_element_error_selfie),

274		tribute), 114
	nhone r	uumber (aiogram.types.encrypted_passport_element.EncryptedPass
aiogram.types.passport_element_error_translation	on_file),	attribute), 268
275	phone_r	number (aiogram.types.inline_query_result_contact.InlineQueryRe
PassportElementErrorTranslationFiles (class in		attribute), 238
alogram.types.passport_element_error_translatio 276	o <i>ip<u>h</u>jola</i> e),r	<pre>number (aiogram.types.input_contact_message_content.InputConta attribute), 257</pre>
PassportElementErrorType(classin	phone_r	number (aiogram.types.order_info.OrderInfo at-
aiogram.enums.passport_element_error_type),		tribute), 279
468	PhoneNu	umber (class in aiogram.utils.formatting), 571
·	РНОТО	(aiogram.enums.content_type.ContentType at-
aiogram.types.passport_element_error_unspecifi		tribute), 459
277	PHOTO (a	uiogram.enums.inline_query_result_type.InlineQueryResultType
PassportFile (class in aiogram.types.passport_file),		attribute), 465
278		uogram.enums.input_media_type.InputMediaType
pattern (aiogram.filters.exception.ExceptionMessageFilte		attribute), 466
attribute), 495	photo	(aiogram.methods.send_photo.SendPhoto at-
pay (aiogram.types.inline_keyboard_button.InlineKeyboard		tribute), 382
attribute), 126	photo	(aiogram.methods.set_chat_photo.SetChatPhoto
payload (aiogram.methods.create_invoice_link.CreateInvo		attribute), 402
attribute), 445		niogram.types.chat.Chat attribute), 30
payload (aiogram.methods.send_invoice.SendInvoice at- tribute), 448	photo	(aiogram.types.chat_shared.ChatShared at- tribute), 114
<pre>payload (aiogram.types.input_invoice_message_content.lr attribute), 259</pre>	1 polntortxoo(i.a	<pre>ieidessagefassureternal_reply_info.ExternalReplyInfo attribute), 117</pre>
PEN (aiogram.enums.currency.Currency attribute), 463	photo (a	niogram.types.game.Game attribute), 287
pending_join_request_count		niogram.types.message.Message attribute), 146
(aiogram.types.chat_invite_link.ChatInviteLink attribute), 50	photo	(aiogram.types.shared_user.SharedUser at- tribute), 205
pending_update_count	photo_f	file_id(aiogram.types.inline_query_result_cached_photo.InlineQ
(aiogram.types.webhook_info.WebhookInfo	• –	<i>attribute</i>), 231
attribute), 286	photo_h	neight (aiogram.methods.create_invoice_link.CreateInvoiceLink
performer (aiogram.methods.send_audio.SendAudio at-	-	attribute), 446
tribute), 365	photo_h	neight (aiogram.methods.send_invoice.SendInvoice
performer (aiogram.types.audio.Audio attribute), 18		attribute), 449
<pre>performer (aiogram.types.inline_query_result_audio.Inlin attribute), 221</pre>	1 pDotry R	asiughtu(hiogram.types.inline_query_result_photo.InlineQueryResu attribute), 250
performer (aiogram.types.input_media_audio.InputMedia	<i>lA</i> pladitoo_ h	neight (aiogram.types.input_invoice_message_content.InputInvoic
attribute), 130	-	attribute), 259
	stphoEbat	Silzenbeiogram.methods.create_invoice_link.CreateInvoiceLink
attribute), 358	-	attribute), 446
permissions (aiogram.methods.set_chat_permissions.Setwattribute), 401	Chlackorns	attribute), 448
permissions (aiogram.types.chat.Chat attribute), 31	photo s	size(aiogram.types.input_invoice_message_content.InputInvoiceM
personal_chat (<i>aiogram.types.chat.Chat attribute</i>), 30	p110 c0_c	attribute), 259
	edentrot. E	and (ypitegiPunsspothElkswentste_invoice_link.CreateInvoiceLink
attribute), 464		<i>attribute</i>), 446
PHONE_NUMBER (aiogram.enums.encrypted_passport_elem	e ph5to rw	
<i>attribute</i>), 465	-1	attribute), 448
PHONE_NUMBER (aiogram.enums.message_entity_type.Mess	a gladzoo ity	(Fypeaiogram.types.inline_query_result_photo.InlineQueryResultPh
attribute), 467	androt-	attribute), 249
attribute), 368		<pre>arl (aiogram.types.input_invoice_message_content.InputInvoiceMe attribute), 259</pre>
phone_number (aiogram.types.contact.Contact at-	photo_u	arl (aiogram.utils.web_app.WebAppChat at-

tribute), 562 attribute), 281	
photo_url (aiogram.utils.web_app.WebAppUser at- PRE (aiogram.enums.message_entity_type.MessageEntityType	
tribute), 562 attribute), 467	
photo_width(aiogram.methods.create_invoice_link.CreataInec(ickIsinkn aiogram.utils.formatting), 572	
attribute), 446 PRE_CHECKOUT_QUERY (aiogram.enums.update_type.UpdateT	ype
photo_width(aiogram.methods.send_invoice.SendInvoice attribute), 471	
attribute), 448 pre_checkout_query (aiogram.types.update.Update at-	
photo_width(aiogram.types.inline_query_result_photo.InlineQueryReibudtePh@to5	
attribute), 249 pre_checkout_query_id	
photo_width(aiogram.types.input_invoice_message_content.InputIn(wiwgMnssagthOdstantswer_pre_checkout_query.Answ attribute), 259 attribute), 442	verPreCheck
photos (aiogram.types.user_profile_photos.UserProfilePhoRrseCheckoutQuery (class in	
attribute), 209 aiogram.types.pre_checkout_query), 280	
PhotoSize (class in aiogram.types.photo_size), 197 prefer_large_media(aiogram.types.link_preview_options.link_prev	LinkPreviewC
PHP (aiogram.enums.currency.Currency attribute), 463attribute), 139	
<pre>pin() (aiogram.types.message.Message method), 190 prefer_small_media(aiogram.types.link_preview_options.link_</pre>	LinkPreviewC
pin_message() (aiogram.types.chat.Chat method), 38 attribute), 139	
PinChatMessage(classinprefix(aiogram.filters.command.CommandObjectat-	
aiogram.methods.pin_chat_message), 352 tribute), 486	
PINNED_MESSAGE (aiogram.enums.content_type.ContentTypPREMIUM (aiogram.enums.chat_boost_source_type.ChatBoost_ attribute), 460 attribute), 458	SourceType
pinned_message (aiogram.types.chat.Chat attribute), premium_animation (aiogram.types.sticker.Sticker at-	
31 <i>tribute</i>), 265	
<pre>pinned_message (aiogram.types.message.Message at- premium_subscription_month_count</pre>	
tribute), 148 (aiogram.types.giveaway.Giveaway attribute),	
PKR (aiogram.enums.currency.Currency attribute), 463 123	
PLN (<i>aiogram.enums.currency.Currency attribute</i>), 463 premium_subscription_month_count	
point (aiogram.types.mask_position.MaskPosition at- tribute), 264 (aiogram.types.giveaway_winners.GiveawayWinners.	5
POLL (aiogram.enums.content_type.ContentType at- prepare_value() (aiogram.client.session.base.BaseSession tribute), 459 method), 14	
POLL (aiogram.enums.update_type.UpdateType at- prices (aiogram.methods.create_invoice_link.CreateInvoiceI	ink
tribute), 471 attribute), 445	лик
poll(aiogram.types.external_reply_info.ExternalReplyInfoprices (aiogram.methods.send_invoice.SendInvoice at-	
attribute), 118 tribute), 448	(musica) Magag
poll (aiogram.types.message.Message attribute), 147 prices (aiogram.types.input_invoice_message_content.Input] poll (aiogram.types.update.Update attribute), 285 attribute), 259	nvoicemessu
poll (aiogram.types.update.Update attribute), 285attribute), 259Poll (class in aiogram.types.poll), 198prices (aiogram.types.shipping_option.ShippingOption	
POLL_ANSWER (aiogram.enums.update_type.UpdateType attribute), 282	
attribute), 471 attribute), 47	
poll_answer (aiogram.types.update.Update attribute), 459	
285 prize_description (aiogram.types.giveaway.Giveaway	
poll_id (aiogram.types.poll_answer.PollAnswer at- attribute), 123	
tribute), 199 prize_description (aiogram.types.giveaway_winners.Give	awavWinners
PollAnswer (<i>class in aiogram.types.poll_answer</i>), 199 <i>attribute</i>), 125	
PollOption (class in aiogram.types.poll_option), 199 profile_accent_color_id (aiogram.types.chat.Chat	
PollType (class in aiogram.enums.poll_type), 469 attribute), 31	
position (aiogram.methods.set_sticker_position_in_set.SetStickEinPersitianRepsitianRepsitianCustom_emoji_id	
attribute), 302 (aiogram.types.chat.Chat attribute), 31	
position (aiogram.types.game_high_score.GameHighScorpromote() (aiogram.types.chat.Chat method), 39	
attribute), 288 PromoteChatMember (class in	
position (aiogram.types.text_quote.TextQuote at- tribute), 207 aiogram.methods.promote_chat_member), 353	
post_code (aiogram.types.shipping_address.ShippingAddressotect_content (aiogram.methods.copy_message.CopyMe	ssage

attribute), 316	provider_token (aiogram.types.input_invoice_message_content.InputInv
protect_content (<i>aiogram.methods.copy_me</i>	
attribute), 318	proximity_alert_radius
attribute), 332	message.ForwardMes (age gram.methods.edit_message_live_location.EditMessageLiveL attribute), 425
protect_content (aiogram.methods.forward_	
attribute), 333	(aiogram.methods.send_location.SendLocation
<pre>protect_content (aiogram.methods.send_ani attribute), 362</pre>	proximity_alert_radius
protect_content (<i>aiogram.methods.send_aud</i>	
attribute), 365	attribute), 246
protect_content (aiogram.methods.send_cor	
attribute), 369	(aiogram.types.input_location_message_content.InputLocationMe
protect_content (aiogram.methods.send_dic	
attribute), 371	proximity_alert_radius
	cument.SendDocument(aiogram.types.location.Location attribute),
attribute), 373	140
protect_content (aiogram.methods.send_gan	
attribute), 439	(aiogram.enums.content_type.ContentType
protect_content (aiogram.methods.send_inv	
attribute), 449	<pre>proximity_alert_triggered ation.SendLocation (aiogram.types.message.Message attribute),</pre>
attribute), 376	148
protect_content (aiogram.methods.send_methods.	
attribute), 378	aiogram.types.proximity_alert_triggered),
protect_content (aiogram.methods.send_methods.sen	ssage.SendMessage 199 PYG (aiogram.enums.currency.Currency attribute), 463
<pre>protect_content(aiogram.methods.send_pho</pre>	oto.SendPhoRython Enhancement Proposals
attribute), 383	PEP 484 , 3
<pre>protect_content (aiogram.methods.send_pol</pre>	I.SendPoll PEP 492, 3
<pre>protect_content (aiogram.methods.send_stic</pre>	
attribute), 297	QAR (aiogram.enums.currency.Currency attribute), 463
attribute), 388	ue.SendVenuery (aiogram.types.chosen_inline_result.ChosenInlineResult attribute), 216
<pre>protect_content (aiogram.methods.send_vid</pre>	tribute). 216
<pre>protect_content(aiogram.methods.send_vid</pre>	eo_note.SendVidegNategram.types.switch_inline_query_chosen_chat.SwitchInlineQuery
attribute), 394	attribute), 206
<pre>protect_content (aiogram.methods.send_voi</pre>	ce.SendVoicquery_id (aiogram.utils.web_app.WebAppInitData at-
	tribute), 560 ice_link.Cre qtedsvqica Linkaiogram.methods.send_poll.SendPoll at-
attribute), 445	tribute), 384
provider_data(<i>aiogram.methods.send_invoid</i>	re.SendInvoiGuestion (aiogram.types.poll.Poll attribute), 198
attribute), 448	QUIZ (aiogram.enums.keyboard_button_poll_type_type.KeyboardButtonPol
<pre>provider_data(aiogram.types.input_invoice_</pre>	
attribute), 259	QUIZ (aiogram.enums.poll_type.PollType attribute), 469
provider_payment_charge_id	quote (aiogram.types.message.Message attribute), 145
attribute), 283	uccessfulPay and e (aiogram.types.reply_parameters.ReplyParameters attribute), 204
<pre>provider_token(aiogram.methods.create_inv</pre>	voice_link.CreateleveiseLinkes (aiogram.types.reply_parameters.ReplyParameters
attribute), 445	attribute), 204
<pre>provider_token(aiogram.methods.send_invo</pre>	ice.SendInvojcote_parse_mode (aiogram.types.reply_parameters.ReplyParameters
attribute), 448	attribute), 204

<pre>quote_position(aiogram.types.reply_parameters.Reply attribute), 204</pre>	Paregiestex () (aiogram.webhook.aiohttp_server.TokenBasedRequestHandler method), 501
R	REGULAR (aiogram.enums.keyboard_button_poll_type_type.KeyboardButtor attribute), 466
react() (aiogram.types.message.Message method), 191	REGULAR (aiogram.enums.poll_type.PollType attribute),
react() (alogram.rypes.message.message memoa), 191 reaction (alogram.methods.set_message_reaction.SetMes attribute), 406	
ReactionCount (class in	attribute), 469
aiogram.types.reaction_count), 200	<pre>remove_caption(aiogram.methods.copy_messages.CopyMessages</pre>
reactions (aiogram.types.message_reaction_count_upda attribute), 196	<pre>remove_date(aiogram.types.chat_boost_removed.ChatBoostRemoved</pre>
ReactionType (class in aiogram.types.reaction_type), 200	attribute), 46 remove_keyboard(aiogram.types.reply_keyboard_remove.ReplyKeyboard
ReactionTypeCustomEmoji (class in	attribute), 203
aiogram.types.reaction_type_custom_emoji), 201	REMOVED_CHAT_BOOST (<i>aiogram.enums.update_type.UpdateType</i> <i>attribute</i>), 471
ReactionTypeEmoji (class in	<pre>removed_chat_boost (aiogram.types.update.Update at-</pre>
aiogram.types.reaction_type_emoji), 201	tribute), 286
ReactionTypeType(classinaiogram.enums.reaction_type_type), 469	render() (aiogram.utils.formatting.Text method), 569 RENTAL_AGREEMENT (aiogram.enums.encrypted_passport_element.Encrypt
read() (aiogram.types.input_file.BufferedInputFile	attribute), 464
<i>method</i>), 127	ReopenForumTopic(classin
<pre>read() (aiogram.types.input_file.FSInputFile method),</pre>	aiogram.methods.reopen_forum_topic), 356
127	ReopenGeneralForumTopic (class in
<pre>read() (aiogram.types.input_file.InputFile method), 127</pre>	aiogram.methods.reopen_general_forum_topic),
<pre>read() (aiogram.types.input_file.URLInputFile method),</pre>	357
127	ReplaceStickerInSet (class in
receiver (aiogram.utils.web_app.WebAppInitData at- tribute), 560	aiogram.methods.replace_sticker_in_set), 295
RECORD_VIDE0 (aiogram.enums.chat_action.ChatAction attribute), 457	<pre>reply() (aiogram.types.message.Message method), 167 reply_animation() (aiogram.types.message.Message</pre>
record_video() (aiogram.utils.chat_action.ChatActionSciences and the class method), 557	
RECORD_VIDEO_NOTE (aiogram.enums.chat_action.ChatAc	
attribute), 458	reply_contact() (aiogram.types.message.Message
record_video_note()	<i>method</i>), 155
(aiogram.utils.chat_action.ChatActionSender class method), 557	reply_dice() (aiogram.types.message.Message method), 173
RECORD_VOICE (aiogram.enums.chat_action.ChatAction attribute), 458	<pre>reply_document() (aiogram.types.message.Message method), 157</pre>
record_voice() (aiogram.utils.chat_action.ChatActionSt class method), 557	
RED (aiogram.enums.topic_icon_color.TopicIconColor attribute), 470	reply_invoice() (aiogram.types.message.Message method), 160
RedisStorage (class in aiogram.fsm.storage.redis), 515	reply_location() (aiogram.types.message.Message
regexp_match (aiogram.filters.command.CommandObjec attribute), 486	
register() (aiogram.fsm.scene.SceneRegistry method),	attribute), 316
532	reply_markup(aiogram.methods.edit_message_caption.EditMessageCaption.edit_messageCapti
<pre>register() (aiogram.webhook.aiohttp_server.BaseReque method), 500</pre>	reply_markup(aiogram.methods.edit_message_live_location.EditMessage
<pre>register() (aiogram.webhook.aiohttp_server.SimpleReq</pre>	uestHandleattribute), 425
method), 500	reply_markup (aiogram.methods.edit_message_media.EditMessageMedia attribute), 427

<pre>reply_markup(aiogram.methods.edit_message_reply_ma</pre>	rknep Eylitharissong (Riply Muthups.inline_query_result_cached_sticker.InlineQ attribute), 233
	Arealy Inarkup (aiogram.types.inline_query_result_cached_video.InlineQu attribute), 235
reply_markup(aiogram.methods.send_animation.SendAn attribute), 362	<pre>simepiby_markup(aiogram.types.inline_query_result_cached_voice.InlineQu attribute), 237</pre>
reply_markup (aiogram.methods.send_audio.SendAudio attribute), 365	reply_markup (aiogram.types.inline_query_result_contact.InlineQueryRe. attribute), 239
reply_markup(aiogram.methods.send_contact.SendConte attribute), 369	acteply_markup(aiogram.types.inline_query_result_document.InlineQuery) attribute), 241
reply_markup (aiogram.methods.send_dice.SendDice attribute), 371	<pre>reply_markup(aiogram.types.inline_query_result_game.InlineQueryResu attribute), 242</pre>
attribute), 373	c napily_ markup(aiogram.types.inline_query_result_gif.InlineQueryResultC attribute), 244
reply_markup(aiogram.methods.send_game.SendGame attribute), 439	reply_markup(aiogram.types.inline_query_result_location.InlineQueryReattribute), 246
attribute), 449	careply_markup(aiogram.types.inline_query_result_mpeg4_gif.InlineQuery attribute), 248
<pre>reply_markup(aiogram.methods.send_location.SendLoca attribute), 376</pre>	<pre>utmeply_markup(aiogram.types.inline_query_result_photo.InlineQueryResu attribute), 250</pre>
reply_markup(aiogram.methods.send_message.SendMess attribute), 380	sargeply_markup(aiogram.types.inline_query_result_venue.InlineQueryResu attribute), 252
attribute), 383	reply_markup(aiogram.types.inline_query_result_video.InlineQueryResu attribute), 254
tribute), 386	<pre>reply_markup(aiogram.types.inline_query_result_voice.InlineQueryResu attribute), 256</pre>
reply_markup(aiogram.methods.send_sticker.SendSticker attribute), 297	tribute), 149
reply_markup(aiogram.methods.send_venue.SendVenue attribute), 388	(aiogram.types.message.Message method),
reply_markup(aiogram.methods.send_video.SendVideo attribute), 391	165 reply_parameters(aiogram.methods.copy_message.CopyMessage
<pre>reply_markup(aiogram.methods.send_video_note.SendVi attribute), 394</pre>	<i>ideoNote</i> attribute), 316 reply_parameters(aiogram.methods.send_animation.SendAnimation
reply_markup (aiogram.methods.send_voice.SendVoice attribute), 396	attribute), 362 reply_parameters (aiogram.methods.send_audio.SendAudio
reply_markup(aiogram.methods.stop_message_live_loca	tion.StopMattageLayeLovation
attribute), 432 reply_markup (aiogram.methods.stop_poll.StopPoll at-	reply_parameters (aiogram.methods.send_contact.SendContact attribute), 369
tribute), 433 reply_markup(aiogram.types.inline_query_result_article	reply_parameters(aiogram.methods.send_dice.SendDice a.InlineOue rstRebult Article
attribute), 219	reply_parameters (aiogram.methods.send_document.SendDocument
<pre>reply_markup(aiogram.types.inline_query_result_audio.</pre>	InlineQuergiNesonna), diogram.methods.send_game.SendGame
<pre>reply_markup(aiogram.types.inline_query_result_cached attribute), 223</pre>	d_audio.Inl itte@uete yR@S@ltCachedAudio reply_parameters(aiogram.methods.send_invoice.SendInvoice
reply_markup(aiogram.types.inline_query_result_cached	d_documentaltrliibuet@yertyResultCachedDocument
attribute), 225 reply_markup(aiogram.types.inline_query_result_cached	reply_parameters (aiogram.methods.send_location.SendLocation d_oif Inline@tmeibuResubilGachedGif
attribute), 227	reply_parameters (aiogram.methods.send_media_group.SendMediaGrou
reply_markup(aiogram.types.inline_query_result_cached	
attribute), 229	reply_parameters (aiogram.methods.send_message.SendMessage
<pre>reply_markup(aiogram.types.inline_query_result_cached</pre>	a_photo.ini mergume eyk@smiCacheaPhoto reply_parameters(aiogram.methods.send_photo.SendPhoto

attribute), 383 reply_to_message_id reply_parameters (aiogram.methods.send_poll.SendPoll (aiogram.methods.send_photo.SendPhoto attribute), 385 attribute), 383 reply_parameters(*aiogram.methods.send_sticker.SendStickeply_to_message_id* attribute), 297 (aiogram.methods.send_poll.SendPoll atreply_parameters (aiogram.methods.send venue.SendVenue tribute), 386 reply_to_message_id attribute), 388 reply_parameters (aiogram.methods.send_video.SendVideo (aiogram.methods.send_sticker.SendSticker attribute), 391 attribute), 297 reply_parameters(aiogram.methods.send_video_note.Semdpideotbotmessage_id attribute), 394 (aiogram.methods.send_venue.SendVenue reply_parameters (aiogram.methods.send_voice.SendVoice attribute), 388 attribute), 396 reply_to_message_id reply_photo() (aiogram.methods.send_video.SendVideo (aiogram.types.message.Message method), 168 attribute), 391 reply_poll() (aiogram.types.message.Message reply_to_message_id (aiogram.methods.send_video_note.SendVideoNote method), 170 reply_sticker() (aiogram.types.message.Message attribute), 394 method), 174 reply_to_message_id reply_to_message (aiogram.types.message.Message (aiogram.methods.send_voice.SendVoice attribute), 145 attribute), 396 reply_to_message_id reply_to_story (aiogram.types.message.Message at-(aiogram.methods.copy_message.CopyMessage tribute), 146 attribute), 316 reply_venue() (aiogram.types.message.Message method), 176 reply_to_message_id (aiogram.methods.send_animation.SendAnimationreply_video() (aiogram.types.message.Message attribute), 362 method), 178 reply_to_message_id reply_video_note() (aiogram.types.message.Message (aiogram.methods.send_audio.SendAudio method), 180 attribute), 365 reply_voice() (aiogram.types.message.Message reply_to_message_id method), 182 (class (aiogram.methods.send_contact.SendContact ReplyKeyboardBuilder in aiogram.utils.keyboard), 550 attribute), 369 ReplyKeyboardMarkup reply_to_message_id (class in (aiogram.methods.send_dice.SendDice ataiogram.types.reply_keyboard_markup), tribute), 371 202 reply_to_message_id ReplyKeyboardRemove (class in (aiogram.methods.send_document.SendDocument aiogram.types.reply_keyboard_remove), 203 attribute), 374 ReplyParameters (class in reply_to_message_id aiogram.types.reply_parameters), 203 (aiogram.methods.send game.SendGame request_chat (aiogram.types.keyboard button.KeyboardButton attribute), 439 attribute), 134 request_contact(aiogram.types.keyboard_button.KeyboardButton reply_to_message_id (aiogram.methods.send_invoice.SendInvoice attribute), 134 attribute), 449 request_id (aiogram.types.chat_shared.ChatShared atreply_to_message_id tribute), 114 (aiogram.methods.send_location.SendLocation request_id(aiogram.types.keyboard_button_request_chat.KeyboardButto attribute), 376 attribute), 136 reply_to_message_id request_id(aiogram.types.keyboard_button_request_user.KeyboardButto (aiogram.methods.send_media_group.SendMediaGroup attribute), 137 attribute), 378 request_id (aiogram.types.keyboard_button_request_users.KeyboardButt attribute), 138 reply_to_message_id (aiogram.methods.send message.SendMessage request_id (aiogram.types.user_shared.UserShared atattribute), 380 tribute), 209

<pre>request_id (aiogram.types.users_shared.UsersShared</pre>	attribute), 216
attribute), 210	results (aiogram.methods.answer_inline_query.AnswerInlineQuery
<pre>request_location(aiogram.types.keyboard_button.Keyl</pre>	
attribute), 134	<pre>retake() (aiogram.fsm.scene.SceneWizard method), 534</pre>
	u sært.KeybbundRuitogrRequestel.sesp onse_parameters.ResponseParameters
attribute), 138	attribute), 204
	RENERSE&EDBILITiogRaquent@hstpassport_element_error_type.PassportEle
attribute), 137	attribute), 468
request_photo(aiogram.types.keyboard_button_request_ attribute), 139	_ ixxve.Kxy b ouddRaitogrRequypte U. xen xrypted_passport_element.EncryptedPas attribute), 269
<pre>request_poll (aiogram.types.keyboard_button.Keyboard attribute), 134</pre>	Bratrooke_invite_link() (aiogram.types.chat.Chat method), 34
request_title(<i>aiogram.types.keyboard_button_request_attribute</i>), 137	_devoke_hwesdBg#on(RiggrestGhathods.ban_chat_member.BanChatMember attribute), 310
request_user (aiogram.types.keyboard_button.Keyboard	
attribute), 135	aiogram.methods.revoke_chat_invite_link),
request_username(<i>aiogram.types.keyboard_button_request_username</i>)	
attribute), 137	rights (aiogram.methods.set_my_default_administrator_rights.SetMyDefa
request_username(<i>aiogram.types.keyboard_button_request_username</i>)	
attribute), 139	RON (aiogram.enums.currency.Currency attribute), 463
	alBOBE (aiogram.enums.topic_icon_color.TopicIconColor
attribute), 134	attribute), 470
request_write_access	Router (class in aiogram.dispatcher.router), 475
(aiogram.types.login_url.LoginUrl attribute), 141	
reset_data_on_enter	row() (aiogram.utils.keyboard.ReplyKeyboardBuilder
(aiogram.fsm.scene.SceneConfig attribute),	<i>method</i>), 551
532	RSD (aiogram.enums.currency.Currency attribute), 463
reset_history_on_enter	RUB (aiogram.enums.currency.Currency attribute), 463
(aiogram.fsm.scene.SceneConfig attribute), 532	<pre>run_polling() (aiogram.dispatcher.dispatcher.Dispatcher method), 482</pre>
resize_keyboard(aiogram.types.reply_keyboard_marku attribute), 202	p.ReplyKeyboardMarkup S
<pre>resolve_bot() (aiogram.webhook.aiohttp_server.BaseRe</pre>	easested_graddse_webapp_init_data() (in module aiogram.utils.web_app), 559
resolve_bot() (aiogram.webhook.aiohttp_server.Simple	RequestHandlerenums currency Currency attribute) 463
method), 500	scale (aiogram.types.mask_position.MaskPosition at-
resolve_bot() (aiogram.webhook.aiohttp_server.TokenE	BasedRequestHandler64
<i>method</i>), 501	Scene (class in aiogram.fsm.scene), 530
resolve_used_update_types()	SceneConfig (class in aiogram.fsm.scene), 532
(aiogram.dispatcher.router.Router method),	SceneException, 540
476	SceneRegistry (class in aiogram.fsm.scene), 531
ResponseParameters (class in	ScenesManager (class in aiogram.fsm.scene), 532
aiogram.types.response_parameters), 204	SceneWizard (class in aiogram.fsm.scene), 533
RestartingTelegram, 540	scope (aiogram.methods.delete_my_commands.DeleteMyCommands
<pre>restrict() (aiogram.types.chat.Chat method), 41</pre>	attribute), 325
RestrictChatMember (class in	scope (aiogram.methods.get_my_commands.GetMyCommands
aiogram.methods.restrict_chat_member),	attribute), 343
358	scope(aiogram.methods.set my commands.SetMyCommands
RESTRICTED (aiogram.enums.chat_member_status.ChatMe	emberStatusuttribute), 407
attribute), 458	<pre>score (aiogram.methods.set_game_score.SetGameScore</pre>
<pre>result(aiogram.methods.answer_web_app_query.Answer</pre>	rWebAppQuativibute), 441
attribute), 436	<pre>score (aiogram.types.game_high_score.GameHighScore</pre>
<pre>result_id(aiogram.types.chosen_inline_result.ChosenIn</pre>	lineResult attribute), 288

<pre>secret (aiogram.types.encrypted_credentials.EncryptedC attribute), 267</pre>	Crestendients_chat (aiogram.types.message_origin_chat.MessageOriginChat attribute), 194
<pre>secret_token(aiogram.methods.set_webhook.SetWebho</pre>	oksender_chat_id(aiogram.methods.ban_chat_sender_chat.BanChatSende attribute), 311
SEK (aiogram.enums.currency.Currency attribute), 463	<pre>sender_chat_id(aiogram.methods.unban_chat_sender_chat.UnbanChat</pre>
<pre>selective (aiogram.types.force_reply.ForceReply</pre>	attribute), 414
attribute), 120	<pre>sender_user(aiogram.types.message_origin_user.MessageOriginUser</pre>
selective (aiogram.types.reply_keyboard_markup.Reply	
attribute), 202	<pre>sender_user_name (aiogram.types.message_origin_hidden_user.Message</pre>
selective (aiogram.types.reply_keyboard_remove.Reply.	
attribute), 203	SendGame (class in aiogram.methods.send_game), 438
SELFIE (aiogram.enums.passport_element_error_type.Pa.	ss Sertellaniointer (allispein aiogram.methods.send_invoice),
attribute), 468	447
<pre>selfie(aiogram.types.encrypted_passport_element.Encr</pre>	yp 8edBisspatiEle ment (class in
attribute), 269	aiogram.methods.send_location), 375
<pre>send_copy() (aiogram.types.message.Message method),</pre>	SendMediaGroup (class in
184	aiogram.methods.send_media_group), 377
send_email_to_provider	SendMessage (class in aiogram.methods.send_message),
(aiogram.methods.create_invoice_link.CreateInv	
attribute), 446	SendPhoto (class in aiogram.methods.send_photo), 381
<pre>send_email_to_provider</pre>	SendPoll (class in aiogram.methods.send_poll), 384
(aiogram.methods.send_invoice.SendInvoice	SendSticker (class in aiogram.methods.send_sticker),
attribute), 449	296
send_email_to_provider	SendVenue (class in aiogram.methods.send_venue), 387
(aiogram.types.input_invoice_message_content	In Satid Vaide Maskassei Gantegnam. methods. send_video), 390
attribute), 260	SendVideoNote (class in
<pre>send_phone_number_to_provider</pre>	aiogram.methods.send_video_note), 392
(aiogram.methods.create_invoice_link.CreateInv	vo&end#loice (class in aiogram.methods.send_voice), 395
attribute), 446	SentWebAppMessage (class in
<pre>send_phone_number_to_provider</pre>	aiogram.types.sent_web_app_message), 263
(aiogram.methods.send_invoice.SendInvoice	<pre>set_administrator_custom_title()</pre>
attribute), 449	(aiogram.types.chat.Chat method), 39
<pre>send_phone_number_to_provider</pre>	<pre>set_data() (aiogram.fsm.scene.SceneWizard method),</pre>
(aiogram.types.input_invoice_message_content.	InputInvoiceM&ssageContent
attribute), 260	<pre>set_data() (aiogram.fsm.storage.base.BaseStorage</pre>
SendAnimation (class in	<i>method</i>), 517
aiogram.methods.send_animation), 361	<pre>set_description() (aiogram.types.chat.Chat method),</pre>
SendAudio (class in aiogram.methods.send_audio), 363	42
SendChatAction (class in	<pre>set_locale() (aiogram.utils.i18n.middleware.FSMI18nMiddleware</pre>
aiogram.methods.send_chat_action), 366	method), 554
<pre>SendContact (class in aiogram.methods.send_contact),</pre>	<pre>set_name (aiogram.types.sticker.Sticker attribute), 265</pre>
368	<pre>set_permissions() (aiogram.types.chat.Chat method),</pre>
SendDice (class in aiogram.methods.send_dice), 370	39
SendDocument (class in	<pre>set_photo() (aiogram.types.chat.Chat method), 43</pre>
aiogram.methods.send_document), 372	<pre>set_position_in_set() (aiogram.types.sticker.Sticker</pre>
SENDER (aiogram.enums.chat_type.ChatType attribute),	method), 266
459	<pre>set_state() (aiogram.fsm.storage.base.BaseStorage</pre>
<pre>sender_boost_count (aiogram.types.message.Message</pre>	<i>method</i>), 516
attribute), 145	<pre>set_sticker_set() (aiogram.types.chat.Chat method),</pre>
sender_business_bot	37
(aiogram.types.message.Message attribute),	<pre>set_title() (aiogram.types.chat.Chat method), 42</pre>
145	SetChatAdministratorCustomTitle (class in
<pre>sender_chat (aiogram.types.message.Message at- tribute), 145</pre>	aiogram.methods.set_chat_administrator_custom_title), 397

SetChatDescription aiogram.methods.set_cha	(class t_description), 399		<pre>setup() (aiogram.utils.i18n.middleware.I18nMiddleware method), 555</pre>	
SetChatMenuButton aiogram.methods.set_cha	(class t_menu_button),	in	SetWebhook (class in aiogram.methods.set_webhook), 453	
400			SGD (aiogram.enums.currency.Currency attribute), 463	
SetChatPermissions	(class	in	SharedUser (class in aiogram.types.shared_user), 205	
aiogram.methods.set_cha	t permissions),		shifted_id (aiogram.types.chat.Chat property), 33	
401			shipping_address(<i>aiogram.types.order_info.OrderInfo</i>	
SetChatPhoto (cl	lass	in	attribute), 279	
aiogram.methods.set_cha			shipping_address (<i>aiogram.types.shipping_query.Shipping</i>	ngOuerv
-	(class	in	attribute), 282	ngguery
aiogram.methods.set_cha	· · · · · · · · · · · · · · · · · · ·		<pre>shipping_option_id (aiogram.types.pre_checkout_query</pre>	ProCheckout
-	ass	in	attribute), 280	.1 recheckowy
		ın		Suggessful Dave
aiogram.methods.set_cha			shipping_option_id (aiogram.types.successful_payment.	.SuccessjuiPayn
SetCustomEmojiStickerSetThu		in		
aiogram.methods.set_cus 298	tom_emoji_sticker_		_the hipping_options (aiogram.methods.answer_shipping_quarteribute), 443	
	ass	in	SHIPPING_QUERY (aiogram.enums.update_type.UpdateType	е
aiogram.methods.set_gan	ne_score), 440		attribute), 471	
SetMessageReaction	(class	in	<pre>shipping_query (aiogram.types.update.Update at-</pre>	
aiogram.methods.set_mes	ssage_reaction),		tribute), 285	
405	-		<pre>shipping_query_id(aiogram.methods.answer_shipping_</pre>	query.AnswerSi
SetMyCommands (c	lass	in	attribute), 443	
aiogram.methods.set_my_			ShippingAddress (class in	
SetMyDefaultAdministratorRi		in	aiogram.types.shipping_address), 281	
aiogram.methods.set_my_				
408			aiogram.types.shipping_option), 281	
SotMulloccription	(class	ın	ShippingQuery (class in	
	•			
aiogram.methods.set_my_	description), 409		aiogram.types.shipping_query), 282	
aiogram.methods.set_my_ SetMyName (class in aiogram.m 410	description), 409	ıe),	<pre>aiogram.types.shipping_query), 282 short_description(aiogram.methods.set_my_short_desc attribute), 411</pre>	
aiogram.methods.set_my_ SetMyName (class in aiogram.m 410 SetMyShortDescription	_description), 409 ethods.set_my_nam (class	ne), in	<pre>aiogram.types.shipping_query), 282 short_description(aiogram.methods.set_my_short_desc</pre>	
aiogram.methods.set_my_ SetMyName (class in aiogram.m 410	_description), 409 ethods.set_my_nam (class	ne), in	<pre>aiogram.types.shipping_query), 282 short_description(aiogram.methods.set_my_short_desc attribute), 411</pre>	
aiogram.methods.set_my_ SetMyName (class in aiogram.me 410 SetMyShortDescription aiogram.methods.set_my_ 411	description), 409 eethods.set_my_nam (class _short_description)	ne), in ,	<pre>aiogram.types.shipping_query), 282 short_description (aiogram.methods.set_my_short_desc</pre>	n.BotShortDesci
aiogram.methods.set_my_ SetMyName (class in aiogram.m 410 SetMyShortDescription aiogram.methods.set_my_ 411 SetPassportDataErrors	description), 409 ethods.set_my_nam (class _short_description) (class	in in , in	<pre>aiogram.types.shipping_query), 282 short_description (aiogram.methods.set_my_short_desc</pre>	n.BotShortDesc inkPreviewOptic
aiogram.methods.set_my_ SetMyName (class in aiogram.me 410 SetMyShortDescription aiogram.methods.set_my_ 411 SetPassportDataErrors aiogram.methods.set_pas	description), 409 ethods.set_my_nam (class _short_description) (class	in in , in	<pre>aiogram.types.shipping_query), 282 short_description(aiogram.methods.set_my_short_desc</pre>	n.BotShortDesci inkPreviewOptic
aiogram.methods.set_my_ SetMyName (class in aiogram.methods.set_my_ 410 SetMyShortDescription aiogram.methods.set_my_ 411 SetPassportDataErrors aiogram.methods.set_pass 455	description), 409 ethods.set_my_nam (class _short_description) (class sport_data_errors)	ne), in , in	<pre>aiogram.types.shipping_query), 282 short_description(aiogram.methods.set_my_short_desc</pre>	n.BotShortDesc inkPreviewOptio swerCallbackQu
aiogram.methods.set_my_ SetMyName (class in aiogram.methods.set_my_ 410 SetMyShortDescription aiogram.methods.set_my_ 411 SetPassportDataErrors aiogram.methods.set_pass 455 SetStickerEmojiList	_description), 409 eethods.set_my_nam (class _short_description) (class sport_data_errors) (class	ne), in , in	<pre>aiogram.types.shipping_query), 282 short_description(aiogram.methods.set_my_short_desc</pre>	n.BotShortDesc inkPreviewOptio swerCallbackQu
aiogram.methods.set_my_ SetMyName (class in aiogram.me 410 SetMyShortDescription aiogram.methods.set_my_ 411 SetPassportDataErrors aiogram.methods.set_pas 455 SetStickerEmojiList aiogram.methods.set_stic	_description), 409 eethods.set_my_nam (class _short_description) (class sport_data_errors) (class	ne), in , in	<pre>aiogram.types.shipping_query), 282 short_description(aiogram.methods.set_my_short_desc</pre>	n.BotShortDesc inkPreviewOptio swerCallbackQu
aiogram.methods.set_my_ SetMyName (class in aiogram.methods.set_my_ 410 SetMyShortDescription aiogram.methods.set_my_ 411 SetPassportDataErrors aiogram.methods.set_pass 455 SetStickerEmojiList aiogram.methods.set_stic 299	description), 409 eethods.set_my_nam (class _short_description) (class esport_data_errors) (class ker_emoji_list),	in , in , in	aiogram.types.shipping_query), 282 short_description (aiogram.methods.set_my_short_desc attribute), 411 short_description (aiogram.types.bot_short_description attribute), 24 show_above_text (aiogram.types.link_preview_options.Li attribute), 139 show_alert (aiogram.methods.answer_callback_query.Ans attribute), 307 show_alert (aiogram.utils.callback_answer.CallbackAnsw property), 565 SimpleI18nMiddleware (class in	n.BotShortDesc inkPreviewOptio swerCallbackQu
aiogram.methods.set_my_ SetMyName (class in aiogram.methods.set_my_ 410 SetMyShortDescription aiogram.methods.set_my_ 411 SetPassportDataErrors aiogram.methods.set_pass 455 SetStickerEmojiList aiogram.methods.set_stic 299 SetStickerKeywords	description), 409 eethods.set_my_nam (class _short_description) (class sport_data_errors) (class ker_emoji_list), (class	ne), in , in	aiogram.types.shipping_query), 282 short_description (aiogram.methods.set_my_short_desc attribute), 411 short_description (aiogram.types.bot_short_description attribute), 24 show_above_text (aiogram.types.link_preview_options.Li attribute), 139 show_alert (aiogram.methods.answer_callback_query.Ans attribute), 307 show_alert (aiogram.utils.callback_answer.CallbackAnsw property), 565 SimpleI18nMiddleware (class in aiogram.utils.i18n.middleware), 553	n.BotShortDescr inkPreviewOptio swerCallbackQu
aiogram.methods.set_my_ SetMyName (class in aiogram.methods.set_my_ 410 SetMyShortDescription aiogram.methods.set_my_ 411 SetPassportDataErrors aiogram.methods.set_pass 455 SetStickerEmojiList aiogram.methods.set_stic 299 SetStickerKeywords aiogram.methods.set_stic	description), 409 eethods.set_my_nam (class _short_description) (class sport_data_errors) (class ker_emoji_list), (class	in , in , in	aiogram.types.shipping_query), 282 short_description (aiogram.methods.set_my_short_desc attribute), 411 short_description (aiogram.types.bot_short_description attribute), 24 show_above_text (aiogram.types.link_preview_options.Li attribute), 139 show_alert (aiogram.methods.answer_callback_query.Ans attribute), 307 show_alert (aiogram.utils.callback_answer.CallbackAnsw property), 565 SimpleI18nMiddleware (class in aiogram.utils.i18n.middleware), 553 SimpleRequestHandler (class in	n.BotShortDescr inkPreviewOptio swerCallbackQu
aiogram.methods.set_my_ SetMyName (class in aiogram.methods.set_my_ 410 SetMyShortDescription aiogram.methods.set_my_ 411 SetPassportDataErrors aiogram.methods.set_pass 455 SetStickerEmojiList aiogram.methods.set_stic 299 SetStickerKeywords aiogram.methods.set_stic 300	description), 409 eethods.set_my_nam (class _short_description) (class rsport_data_errors) (class ker_emoji_list), (class ker_keywords),	in , in , in in	aiogram.types.shipping_query), 282 short_description(aiogram.methods.set_my_short_desc attribute), 411 short_description(aiogram.types.bot_short_description attribute), 24 show_above_text(aiogram.types.link_preview_options.Li attribute), 139 show_alert(aiogram.methods.answer_callback_query.Ans attribute), 307 show_alert(aiogram.utils.callback_answer.CallbackAnsw property), 565 SimpleI18nMiddleware(class in aiogram.utils.i18n.middleware), 553 SimpleRequestHandler(class in aiogram.webhook.aiohttp_server), 500	n.BotShortDesc inkPreviewOptio swerCallbackQu
aiogram.methods.set_my_ SetMyName (class in aiogram.methods.set_my_ 410 SetMyShortDescription aiogram.methods.set_my_ 411 SetPassportDataErrors aiogram.methods.set_pass 455 SetStickerEmojiList aiogram.methods.set_stic 299 SetStickerKeywords aiogram.methods.set_stic 300 SetStickerMaskPosition	description), 409 eethods.set_my_nam (class _short_description) (class sport_data_errors) (class ker_emoji_list), (class ker_keywords), (class	in in in in in in	aiogram.types.shipping_query), 282 short_description (aiogram.methods.set_my_short_desc attribute), 411 short_description (aiogram.types.bot_short_description attribute), 24 show_above_text (aiogram.types.link_preview_options.Li attribute), 139 show_alert (aiogram.methods.answer_callback_query.Ans attribute), 307 show_alert (aiogram.utils.callback_answer.CallbackAnsw property), 565 SimpleI18nMiddleware (class in aiogram.utils.i18n.middleware), 553 SimpleRequestHandler (class in aiogram.webhook.aiohttp_server), 500 SLOT_MACHINE (aiogram.enums.dice_emoji.DiceEmoji	n.BotShortDescr inkPreviewOptio swerCallbackQu
aiogram.methods.set_my_ SetMyName (class in aiogram.methods.set_my_ 410 SetMyShortDescription aiogram.methods.set_my_ 411 SetPassportDataErrors aiogram.methods.set_pass 455 SetStickerEmojiList aiogram.methods.set_stic 299 SetStickerKeywords aiogram.methods.set_stic 300 SetStickerMaskPosition aiogram.methods.set_stic	description), 409 eethods.set_my_nam (class _short_description) (class sport_data_errors) (class ker_emoji_list), (class ker_keywords), (class	in in in in in in	aiogram.types.shipping_query), 282 short_description (aiogram.methods.set_my_short_desc attribute), 411 short_description (aiogram.types.bot_short_description attribute), 24 show_above_text (aiogram.types.link_preview_options.Li attribute), 139 show_alert (aiogram.methods.answer_callback_query.Ans attribute), 307 show_alert (aiogram.utils.callback_answer.CallbackAnsw property), 565 SimpleI18nMiddleware (class in aiogram.utils.i18n.middleware), 553 SimpleRequestHandler (class in aiogram.webhook.aiohttp_server), 500 SLOT_MACHINE (aiogram.enums.dice_emoji.DiceEmoji attribute), 464	n.BotShortDescr inkPreviewOptio swerCallbackQu
aiogram.methods.set_my_ SetMyName (class in aiogram.methods.set_my_ 410 SetMyShortDescription aiogram.methods.set_my_ 411 SetPassportDataErrors aiogram.methods.set_pass 455 SetStickerEmojiList aiogram.methods.set_stic 300 SetStickerMaskPosition aiogram.methods.set_stic 301	_description), 409 eethods.set_my_nam (class _short_description) (class sport_data_errors) (class ker_emoji_list), (class ker_keywords), (class ker_mask_position)	in , , , in , in in ,	aiogram.types.shipping_query), 282 short_description (aiogram.methods.set_my_short_desc attribute), 411 short_description (aiogram.types.bot_short_description attribute), 24 show_above_text (aiogram.types.link_preview_options.Li attribute), 139 show_alert (aiogram.methods.answer_callback_query.Ans attribute), 307 show_alert (aiogram.utils.callback_answer.CallbackAnsw property), 565 SimpleI18nMiddleware (class in aiogram.utils.i18n.middleware), 553 SimpleRequestHandler (class in aiogram.webhook.aiohttp_server), 500 SLOT_MACHINE (aiogram.types.dice.DiceEmoji at-	n.BotShortDescr inkPreviewOptio swerCallbackQu
aiogram.methods.set_my_ SetMyName (class in aiogram.methods.set_my_ 410 SetMyShortDescription aiogram.methods.set_my_ 411 SetPassportDataErrors aiogram.methods.set_pass 455 SetStickerEmojiList aiogram.methods.set_stic 299 SetStickerKeywords aiogram.methods.set_stic 300 SetStickerMaskPosition aiogram.methods.set_stic 301 SetStickerPositionInSet	_description), 409 eethods.set_my_nam (class _short_description) (class sport_data_errors) (class ker_emoji_list), (class ker_keywords), (class ker_mask_position) (class	ne), in , in in in in in	aiogram.types.shipping_query), 282 short_description (aiogram.methods.set_my_short_desc attribute), 411 short_description (aiogram.types.bot_short_description attribute), 24 show_above_text (aiogram.types.link_preview_options.Li attribute), 139 show_alert (aiogram.methods.answer_callback_query.Ans attribute), 307 show_alert (aiogram.utils.callback_answer.CallbackAnsw property), 565 SimpleI18nMiddleware (class in aiogram.utils.i18n.middleware), 553 SimpleRequestHandler (class in aiogram.webhook.aiohttp_server), 500 SLOT_MACHINE (aiogram.types.dice_DiceEmoji at- tribute), 115	n.BotShortDescr inkPreviewOptio swerCallbackQu
aiogram.methods.set_my_ SetMyName (class in aiogram.methods.set_my_ 410 SetMyShortDescription aiogram.methods.set_my_ 411 SetPassportDataErrors aiogram.methods.set_pass 455 SetStickerEmojiList aiogram.methods.set_stic 300 SetStickerMaskPosition aiogram.methods.set_stic 301 SetStickerPositionInSet aiogram.methods.set_stic	_description), 409 eethods.set_my_nam (class _short_description) (class sport_data_errors) (class ker_emoji_list), (class ker_keywords), (class ker_mask_position) (class	ne), in , in in in in in	<pre>aiogram.types.shipping_query), 282 short_description(aiogram.methods.set_my_short_desc</pre>	n.BotShortDesc inkPreviewOptio swerCallbackQu
aiogram.methods.set_my_ SetMyName (class in aiogram.methods.set_my_ 410 SetMyShortDescription aiogram.methods.set_my_ 411 SetPassportDataErrors aiogram.methods.set_pass 455 SetStickerEmojiList aiogram.methods.set_stic 299 SetStickerKeywords aiogram.methods.set_stic 300 SetStickerMaskPosition aiogram.methods.set_stic 301 SetStickerPositionInSet aiogram.methods.set_stic 302	_description), 409 eethods.set_my_nam (class _short_description) (class rsport_data_errors) (class ker_emoji_list), (class ker_keywords), (class ker_mask_position) (class ker_position_in_set	ne), in , in in in in in in in , in	<pre>aiogram.types.shipping_query), 282 short_description(aiogram.methods.set_my_short_desc</pre>	n.BotShortDesc inkPreviewOptio swerCallbackQu
aiogram.methods.set_my_ SetMyName (class in aiogram.methods.set_my_ 410 SetMyShortDescription aiogram.methods.set_my_ 411 SetPassportDataErrors aiogram.methods.set_pass 455 SetStickerEmojiList aiogram.methods.set_stic 299 SetStickerKeywords aiogram.methods.set_stic 300 SetStickerMaskPosition aiogram.methods.set_stic 301 SetStickerPositionInSet aiogram.methods.set_stic 302 SetStickerSetThumbnail	_description), 409 eethods.set_my_nam (class _short_description) (class sport_data_errors) (class ker_emoji_list), (class ker_keywords), (class ker_mask_position) (class ker_position_in_set (class	ne), in , in in in in in in in t), in in	<pre>aiogram.types.shipping_query), 282 short_description (aiogram.methods.set_my_short_desc</pre>	n.BotShortDesc inkPreviewOptio swerCallbackQu
aiogram.methods.set_my_ SetMyName (class in aiogram.methods.set_my_ 410 SetMyShortDescription aiogram.methods.set_my_ 411 SetPassportDataErrors aiogram.methods.set_pass 455 SetStickerEmojiList aiogram.methods.set_stic 300 SetStickerKeywords aiogram.methods.set_stic 301 SetStickerPosition SetStickerPositionInSet aiogram.methods.set_stic 302 SetStickerSetThumbnail aiogram.methods.set_stic	_description), 409 eethods.set_my_nam (class _short_description) (class sport_data_errors) (class ker_emoji_list), (class ker_keywords), (class ker_mask_position) (class ker_position_in_set (class	ne), in , in in in in in in in t), in in	<pre>aiogram.types.shipping_query), 282 short_description (aiogram.methods.set_my_short_desc</pre>	n.BotShortDesc inkPreviewOptio swerCallbackQu
aiogram.methods.set_my_ SetMyName (class in aiogram.methods.set_my_ 410 SetMyShortDescription aiogram.methods.set_my_ 411 SetPassportDataErrors aiogram.methods.set_pass 455 SetStickerEmojiList aiogram.methods.set_stic 300 SetStickerMaskPosition aiogram.methods.set_stic 301 SetStickerPositionInSet aiogram.methods.set_stic 302 SetStickerSetThumbnail aiogram.methods.set_stic 303	_description), 409 eethods.set_my_nam (class _short_description) (class sport_data_errors) (class ker_emoji_list), (class ker_keywords), (class ker_mask_position) (class ker_position_in_set (class	ne), in , in in in in in in in t), in in	<pre>aiogram.types.shipping_query), 282 short_description (aiogram.methods.set_my_short_desc</pre>	n.BotShortDescr inkPreviewOptio swerCallbackQu
aiogram.methods.set_my_ SetMyName (class in aiogram.methods.set_my_ 410 SetMyShortDescription aiogram.methods.set_my_ 411 SetPassportDataErrors aiogram.methods.set_pass 455 SetStickerEmojiList aiogram.methods.set_stic 300 SetStickerKeywords aiogram.methods.set_stic 301 SetStickerPosition SetStickerPositionInSet aiogram.methods.set_stic 302 SetStickerSetThumbnail aiogram.methods.set_stic	_description), 409 eethods.set_my_nam (class _short_description) (class sport_data_errors) (class ker_emoji_list), (class ker_keywords), (class ker_mask_position) (class ker_position_in_set (class ker_set_thumbnail) (class	ne), in , in in in in in in in t), in in	<pre>aiogram.types.shipping_query), 282 short_description (aiogram.methods.set_my_short_desc</pre>	n.BotShortDescr inkPreviewOptio swerCallbackQu

<pre>source (aiogram.types.chat_boost.ChatBoost attribute),</pre>		aiogram.types.chat_mer attribute), 91	nber_member.ChatMe	emberMember
<pre>source(aiogram.types.chat_boost_removed.ChatBoostRen</pre>	n otæ ðtus (d		nber_owner.ChatMem	berOwner
source (aiogram.types.chat_boost_source_gift_code.Chath attribute), 47	BotatSiasi (a	, .	nber_restricted.ChatN	1emberRestricted
<pre>source(aiogram.types.chat_boost_source_giveaway.Chathered</pre>	BSTLISKER	ceGüorgaarayenums.conte	nt_type.ContentType	at-
attribute), 48 source (aiogram.types.chat_boost_source_premium.ChatE	BSTASKER		_query_result_type.Inl	ineQueryResultType
attribute), 48 source (aiogram.types.passport_element_error_data_field	l. Basiqkoert	-	_sticker_to_set.AddStic	ckerToSet
attribute), 270 source (aiogram.types.passport_element_error_file.Passpo		attribute), 289 EivogrEih emethods.delei	te_sticker_from_set.De	eleteStickerFromSet
attribute), 271 source (aiogram.types.passport_element_error_files.Passp		attribute), 291		
attribute), 272		attribute), 295	-	
<pre>source(aiogram.types.passport_element_error_front_side</pre>		E leniograEnnovelthodtsSida o tribute), 296	<i>1_sticker.SendSticker</i>	at-
<pre>source (aiogram.types.passport_element_error_reverse_si attribute), 274</pre>		(tEbgræmtEvetbucRlevæets<u>e</u> attribute), 299	Sidker_emoji_list.SetS	tickerEmojiList
source (aiogram.types.passport_element_error_selfie.Pass attribute), 275		(w ui Egman Selfet hods.set_s attribute), 300	sticker_keywords.SetSt	ickerKeywords
source (aiogram.types.passport_element_error_translation attribute), 275	nsfilecRers	<i>,,</i>	s tictken<u>F</u>iila sk_position.	SetStickerMaskPosit
source (aiogram.types.passport_element_error_translation attribute), 276	n <u>s</u> filæsklens	<i>,,</i>	stilekteo<u>n</u>Fileis tion_in_set	t.SetStickerPositionIr
<pre>source(aiogram.types.passport_element_error_unspecifie</pre>	dsRachperr	, .	wad_sticker_file.Upload	lStickerFile
attribute), 277 SPOILER (aiogram.enums.message_entity_type.MessageEn		· · ·	s intro RusinessIntro	at-
attribute), 467		tribute), 25		
<pre>Spoiler (class in aiogram.utils.formatting), 571 start_date(aiogram.types.video_chat_scheduled.VideoC</pre>	ThatSchedu			
attribute), 212 start_param (aiogram.utils.web_app.WebAppInitData	sticker	(alogram.types.i attribute), 263	nput_sticker.InputStick	<i>ker</i>
attribute), 561 start_parameter(aiogram.methods.send_invoice.SendIn		(aiogram.types.message (class in aiogram.types		46
attribute), 448 start_parameter (aiogram.types.inline_query_results_bi	sticker	_file_id(aiogram.typ	pes.inline_query_result	t_cached_sticker.Inlin
attribute), 256		_format (<i>aiogram.met</i> l		er set.CreateNewStic
start_parameter (aiogram.types.invoice.Invoice		attribute), 290		
attribute), 278		_format(<i>aiogram.metl</i>	ods.upload_sticker_fi	le. Upload Sticker File
<pre>start_polling() (aiogram.dispatcher.dispatcher.Dispatcher.Dispatcher.dispatcher.Dispatcher.dispatcher.Dispatcher.dispatcher.dispatcher.Dispatcher.dis</pre>		attribute), 306		
method), 482 state (aiogram.fsm.scene.SceneConfig attribute), 533		_set_name(<i>aiogram.m</i> <i>attribute</i>), 403	etnoas.set_cnat_sticke	er_set.SetChatSticker
state (alogram.types.shipping_address.ShippingAddress		_set_name (<i>aiogram.t</i> y	vnes chat Chat attribut	(P)
attribute), 281	Sticker_	32	pes.enai.enai annoai	c),
STATIC (aiogram.enums.sticker_format.StickerFormat attribute), 469		_type (aiogram.method attribute), 290	ls.create_new_sticker_	_set.CreateNewSticke
<pre>status (aiogram.types.chat_member_administrator.ChatM attribute), 88</pre>		n tryipæ a (ai ogram.types tribute), 266	sticker_set.StickerSet	at-
status (aiogram.types.chat_member_banned.ChatMember attribute), 90	r Buink ærl		(class format) 469	in
status (aiogram.types.chat_member_left.ChatMemberLef.		•	•	SreateNewStickerSet
attribute), 90		attribute), 290		

stickers (aiogram.types.sticker_set.StickerSet at-	attribute), 391
tribute), 266	<pre>supports_streaming(aiogram.types.input_media_video.InputMediaVide</pre>
StickerSet (class in aiogram.types.sticker_set), 266	attribute), 134
StickerType (class in aiogram.enums.sticker_type), 469	switch_inline_query
<pre>stop_live_location()</pre>	(aiogram.types.inline_keyboard_button.InlineKeyboardButton
(aiogram.types.message.Message method),	attribute), 126
189 <pre>stop_polling() (aiogram.dispatcher.dispatcher.Dispatch</pre>	<pre>switch_inline_query_chosen_chat</pre>
method), 482	attribute), 126
	<pre>switch_inline_query_current_chat</pre>
aiogram.methods.stop_message_live_location), 431	(aiogram.types.inline_keyboard_button.InlineKeyboardButton attribute), 126
StopPoll (<i>class in aiogram.methods.stop_poll</i>), 432	switch_pm_parameter
STORY (aiogram.enums.content_type.ContentType at- tribute), 459	(aiogram.methods.answer_inline_query.AnswerInlineQuery attribute), 435
<pre>story (aiogram.types.external_reply_info.ExternalReplyIn attribute), 118</pre>	<pre>fawitch_pm_text(aiogram.methods.answer_inline_query.AnswerInlineQu attribute), 435</pre>
story (aiogram.types.message.Message attribute), 146	SwitchInlineQueryChosenChat (class in
Story (class in aiogram.types.story), 205	aiogram.types.switch_inline_query_chosen_chat),
<pre>stream_content() (aiogram.client.session.base.BaseSes. method), 14</pre>	-
<pre>street_line1(aiogram.types.shipping_address.Shipping.</pre>	Address
attribute), 281	<pre>telegram_payment_charge_id</pre>
<pre>street_line2(aiogram.types.shipping_address.Shipping. attribute), 281</pre>	Address (aiogram.types.successful_payment.SuccessfulPayment attribute), 283
STRIKETHROUGH (aiogram.enums.message_entity_type.Me.	
attribute), 467	TelegramAPIServer (class in aiogram.client.telegram),
Strikethrough (class in aiogram.utils.formatting), 571	12
SUCCESSFUL_PAYMENT (aiogram.enums.content_type.Cont	
attribute), 460	TelegramConflictError, 540
	TelegramEntityTooLarge, 540
attribute), 148 SuccessfulPayment (class in	TelegramForbiddenError, 540
aiogram.types.successful_payment), 283	TelegramMigrateToChat, 540 TelegramNetworkError, 540
suggested_tip_amounts	TelegramNotFound, 540
(aiogram.methods.create_invoice_link.CreateInv	
attribute), 445	TelegramServerError, 540
suggested_tip_amounts	TelegramUnauthorizedError, 540
(aiogram.methods.send_invoice.SendInvoice	TEMPORARY_REGISTRATION
attribute), 448	(aiogram.enums.encrypted_passport_element.EncryptedPassport.
<pre>suggested_tip_amounts</pre>	attribute), 465
(aiogram.types.input_invoice_message_content.L attribute), 259	n pENI voi (aMgssageGantsn ontent_type.ContentType at- tribute), 459
SUPERGROUP (aiogram.enums.chat_type.ChatType attribute), 459	text (aiogram.filters.command.CommandObject prop- erty), 486
SUPERGROUP_CHAT_CREATED	text (aiogram.methods.answer_callback_query.AnswerCallbackQuery
(aiogram.enums.content_type.ContentType	attribute), 307
attribute), 460	<pre>text(aiogram.methods.edit_message_text.EditMessageText</pre>
<pre>supergroup_chat_created</pre>	attribute), 430
(aiogram.types.message.Message attribute), 147	text (aiogram.methods.send_message.SendMessage at- tribute), 380
<pre>supports_inline_queries (aiogram.types.user.User</pre>	text (aiogram.types.game.Game attribute), 287
attribute), 208	text (aiogram.types.inline_keyboard_button.InlineKeyboardButton
<pre>supports_streaming(aiogram.methods.send_video.Send</pre>	lVideo attribute), 126

text (aiogram.types.inline_query_results_button.InlineQue attribute), 256	e t.jnRmbrlasBu tto(naiogram.types.video_note.VideoNote at- tribute), 213
text (aiogram.types.keyboard_button.KeyboardButton attribute), 134	thumbnail_height (<i>aiogram.types.inline_query_result_article.InlineQuerattribute</i>), 220
text (aiogram.types.menu_button.MenuButton at- tribute), 142	thumbnail_height (aiogram.types.inline_query_result_contact.InlineQue attribute), 239
text (aiogram.types.menu_button_web_app.MenuButtonW attribute), 143	ethappbnail_height (aiogram.types.inline_query_result_document.InlineQ attribute), 241
text (aiogram.types.message.Message attribute), 146 text (aiogram.types.poll_option.PollOption attribute),	thumbnail_height (aiogram.types.inline_query_result_location.InlineQu attribute), 246
199 text (aiogram.types.text_quote.TextQuote attribute), 207	thumbnail_height (<i>aiogram.types.inline_query_result_venue.InlineQuery_attribute</i>), 252
text (aiogram.utils.callback_answer.CallbackAnswer property), 565	<pre>thumbnail_mime_type (aiogram.types.inline_query_result_gif.InlineQueryResultGif</pre>
Text (class in aiogram.utils.formatting), 569	attribute), 244 thumbnail_mime_type
288 TEXT_LINK (aiogram.enums.message_entity_type.Message)	(aiogram.types.inline_query_result_mpeg4_gif.InlineQueryResult
attribute), 467 TEXT_MENTION (aiogram.enums.message_entity_type.Mess	thumbnail_url (aiogram.types.inline_query_result_article.InlineQueryRe
attribute), 467 TextLink (class in aiogram.utils.formatting), 572	thumbnail_url (aiogram.types.inline_query_result_contact.InlineQueryR attribute), 239
TextMention (class in alogram.utils.formatting), 572 TextQuote (class in alogram.types.text_quote), 207	thumbnail_url (aiogram.types.inline_query_result_document.InlineQuer_ attribute), 241
THB (aiogram.enums.currency.Currency attribute), 463 thumbnail (aiogram.methods.send_animation.SendAnimat	thumbnail_url (aiogram.types.inline_query_result_gif.InlineQueryResult
attribute), 362 thumbnail (aiogram.methods.send_audio.SendAudio at-	thumbnail_url (aiogram.types.inline_query_result_location.InlineQueryl attribute), 246
tribute), 365 thumbhail (aiogram.methods.send_document.SendDocum	thumbnail_url (aiogram.types.inline_query_result_mpeg4_gif.InlineQuery_
attribute), 373	$\texttt{thumbnail_url}\ (a iogram.types.inline_query_result_photo.InlineQueryResul$
thumbnail (aiogram.methods.send_video.SendVideo at- tribute), 391	attribute), 249 thumbnail_url (aiogram.types.inline_query_result_venue.InlineQueryRes
thumbnail (aiogram.methods.send_video_note.SendVideo) attribute), 393	$\texttt{thumbnail_url} (a iogram.types.inline_query_result_video.InlineQueryResult$
thumbnail (aiogram.methods.set_sticker_set_thumbnail.Se attribute), 304	$\texttt{thumbnail_width} (a iogram.types.inline_query_result_article.InlineQuery_result_article.Inline_article.Inline_article.Inline_article.InlineQuery_result_article.Inline_article.Inline_article.InlineQuery_result_article.Inline_articl$
thumbnail (aiogram.types.animation.Animation at- tribute), 18	attribute), 220 thumbnail_width (aiogram.types.inline_query_result_contact.InlineQuer
	attribute), 239 thumbnail_width (aiogram.types.inline_query_result_document.InlineQu
	attribute), 241 attribute), 241 attribute), 246 attribute), 246 attribute), 246
	attribute), 246 Abadiaobnail_width (aiogram.types.inline_query_result_venue.InlineQuery)
<pre>attribute), 130 thumbnail(aiogram.types.input_media_document.InputMattribute), 131</pre>	attribute), 252 ethin@ozonwenthame(aiogram.types.business_opening_hours.BusinessOpening_hours) 27
thumbnail (aiogram.types.input_media_video.InputMedia	
attribute), 133 thumbnail (aiogram.types.sticker.Sticker attribute), 265	tribute), 452 title(aiogram.methods.create_invoice_link.CreateInvoiceLink attribute) 445
thumbnail (aiogram.types.sticker_set.StickerSet at- tribute), 267	attribute), 445 title(aiogram.methods.create_new_sticker_set.CreateNewStickerSet
thumbnail (aiogram.types.video.Video attribute), 211	attribute), 290

title		at-	attribute), 262
	tribute), 365		title (aiogram.types.invoice.Invoice attribute), 278
title	(aiogram.methods.send_invoice.SendInvoice tribute), 447	at-	title (aiogram.types.shipping_option.ShippingOption attribute), 281
title	(aiogram.methods.send_venue.SendVenue tribute), 387	at-	title (aiogram.types.sticker_set.StickerSet attribute), 266
title	(aiogram.methods.set_chat_title.SetChatTitle tribute), 404	at-	title (aiogram.types.venue.Venue attribute), 210 title (aiogram.utils.web_app.WebAppChat attribute),
title	(aiogram.methods.set_sticker_set_title.SetStick attribute), 305	erSet	
title	(aiogram.types.audio.Audio attribute), 18		TokenBasedRequestHandler (class in
	(aiogram.types.business_intro.BusinessIntro tribute), 25	at-	aiogram.webhook.aiohttp_server), 501
+:+10			
	(aiogram.types.chat.Chat attribute), 30	,	aiogram.enums.topic_icon_color), 470
title	(aiogram.types.chat_shared.ChatShared tribute), 114	at-	278
	(aiogram.types.game.Game attribute), 287		<pre>total_amount(aiogram.types.pre_checkout_query.PreCheckoutQuery</pre>
title	aiogram.types.inline_query_result_article.Inl	ineQı	
title(attribute), 219 (aiogram.types.inline_query_result_audio.Inlin	ıeQu	total_amount (aiogram.types.successful_payment.SuccessfulPayment eryResultAudiribute), 283
	attribute), 221		<pre>total_count (aiogram.types.reaction_count.ReactionCount</pre>
title((aiogram.types.inline_query_result_cached_do attribute), 225	сите	ent.InlineQuatryRusel];CaOhedDocument total_count(aiogram.types.user_profile_photos.UserProfilePhotos
+i+10((aiogram.types.inline_query_result_cached_gi	f Inlii	
LI LI C	attribute), 226		total_voter_count (aiogram.types.poll.Poll attribute),
title((aiogram.types.inline_query_result_cached_m	neo4	
ci ci ci c	attribute), 229	-150	translation(aiogram.types.encrypted_passport_element.EncryptedPass
title	(aiogram.types.inline_query_result_cached_ph	oto.I	
	attribute), 231		TRANSLATION_FILE (aiogram.enums.passport_element_error_type.Passpo
title	aiogram.types.inline_query_result_cached_vi	deo.Iı	
	attribute), 235		TRANSLATION_FILES (aiogram.enums.passport_element_error_type.Passp
title	aiogram.types.inline_query_result_cached_vo	ice.Ir	
	attribute), 237		traveler (aiogram.types.proximity_alert_triggered.ProximityAlertTrigger
title	aiogram.types.inline_query_result_document.	Inline	
	attribute), 240		TRY (aiogram.enums.currency.Currency attribute), 463
title((aiogram.types.inline_query_result_gif.InlineQ	Juery	RETDL(Gifgram.enums.currency.Currency attribute), 463
	attribute), 244		TWD (aiogram.enums.currency.Currency attribute), 463
title((aiogram.types.inline_query_result_location.Ir attribute), 246	ılineQ	Q txyp_yResultIgrantione thods.send_poll.SendPoll attribute), 385
title((aiogram.types.inline_query_result_mpeg4_gij attribute), 248	Inlin.	ne Opper(kRegrlaMpyg4Gli ßt_command_scope_all_chat_administrators.BotCor attribute), 20
title		1eQue	erty fres (ltiBkotum.types.bot_command_scope_all_group_chats.BotCommandS attribute), 21
title(ıeQu	attribute), 21 attribute), 21 attribute), 21
+i+10(o Ou	ery Result Kigham.types.bot_command_scope_chat.BotCommandScopeChat
	attribute), 254		attribute), 22
title((aiogram.types.inline_query_result_voice.Inlin attribute), 255	eQue	er yRpal(lallogica m.types.bot_command_scope_chat_administrators.BotComma attribute), 22
title		t.Inpı	ut Lypei(aMgrsunga)[pententt_ command_scope_chat_member.BotCommandSco attribute), 23
+i+1~	attribute), 258 (algoright transformedia, audio, InputMadia	1	attribute), 25 iotype(aiogram.types.bot_command_scope_default.BotCommandScopeDefa
(1116(attribute), 130	1/1UUl	attribute), 23
title		Innut	tV zypeMaisggan ampentchat.Chat attribute), 30

type (aiogram.types.encrypted_passport_element.EncryptedBppspor	(Elogwan t.types.menu_button.MenuButton at-
attribute), 268	tribute), 141
type (aiogram.types.inline_query_result_article.InlineQueryRpm(aA attribute), 219	agichen.types.menu_button_commands.MenuButtonCommands attribute), 142
type (aiogram.types.inline_query_result_audio.InlineQuerytRepaltation	adiam.types.menu_button_default.MenuButtonDefault
attribute), 221	attribute), 142
type (aiogram.types.inline_query_result_cached_audio.Inli nyQueen attribute), 222	ResaultCypebechendidbutton_web_app.MenuButtonWebApp attribute), 143
type (aiogram.types.inline_query_result_cached_document .typie eQa	uergyResul/p@cached&ugeurnenit y.MessageEntity at-
attribute), 225	tribute), 192
type (aiogram.types.inline_query_result_cached_gif.Inline@ypey@dia attribute), 226	sight@ndhpd&iffessage_origin_channel.MessageOriginChannel attribute), 193
type (aiogram.types.inline_query_result_cached_mpeg4_gif.jple(eq)	h gercyR#spltConelsed}#pegi{Ginf_ chat.MessageOriginChat
attribute), 229	attribute), 194
type (aiogram.types.inline_query_result_cached_photo.Inli nyQu (ai)	Result.Cypbsethelssag e_origin_hidden_user.MessageOriginHiddenU
attribute), 231	attribute), 195
type (aiogram.types.inline_query_result_cached_sticker.Inltyg@u(enig	o ßesultt§peh.ed&siclger_ origin_user.MessageOriginUser
attribute), 233	attribute), 195
type (aiogram.types.inline_query_result_cached_video.Inling@a(m)	RessauhtGypolex.dMisdport_element_error_data_field.PassportElementE
attribute), 234	attribute), 270
type (aiogram.types.inline_query_result_cached_voice.Inline@packagin	RgsultGyphsdWasiport_element_error_file.PassportElementErrorFil
attribute), 236	attribute), 271
type (aiogram.types.inline_query_result_contact.InlineQue types(di	<pre>Ggmtmctypes.passport_element_error_files.PassportElementErrorF</pre>
attribute), 238	attribute), 272
type (aiogram.types.inline_query_result_document.InlineQtypeRasia	<pre>olgEumatypestpassport_element_error_front_side.PassportElementE</pre>
attribute), 240	attribute), 273
type (aiogram.types.inline_query_result_game.InlineQueryRepal(Gia	<pre>mgwam.types.passport_element_error_reverse_side.PassportElemen</pre>
attribute), 242	attribute), 274
type (aiogram.types.inline_query_result_gif.InlineQueryRexypeIfaio	<pre>ogram.types.passport_element_error_selfie.PassportElementError.</pre>
attribute), 243	attribute), 275
type (aiogram.types.inline_query_result_location.InlineQuery_Re (all	<pre>bgoantitypes.passport_element_error_translation_file.PassportElem</pre>
attribute), 245	attribute), 275
type (aiogram.types.inline_query_result_mpeg4_gif.Inline@ayep@Reis	ndtMpay#&ipassport_element_error_translation_files.PassportEle
attribute), 247	attribute), 276
type (aiogram.types.inline_query_result_photo.InlineQuerytesea (util	<pre>bgtam.types.passport_element_error_unspecified.PassportElement</pre>
attribute), 249	attribute), 277
type (aiogram.types.inline_query_result_venue.InlineQuerytRepel(dda	<i>agream.types.poll.Poll attribute)</i> , 198
attribute), 251 type (a	iogram.types.reaction_count.ReactionCount at-
type (aiogram.types.inline_query_result_video.InlineQueryResultVid	d ex ibute), 200
attribute), 253 type (aio	ogram.types.reaction_type_custom_emoji.ReactionTypeCustomEmo
type (aiogram.types.inline_query_result_voice.InlineQueryResultVo	iættribute), 201
attribute), 255 type (aio	ogram.types.reaction_type_emoji.ReactionTypeEmoji
type (aiogram.types.input_media_animation.InputMediaAnimation	attribute), 201
attribute), 128 type (ai	ogram.types.sticker.Sticker attribute), 265
type (aiogram.types.input_media_audio.InputMediaAudio type (a attribute), 129	aiogram.utils.web_app.WebAppChat attribute), 562
type (aiogram.types.input_media_document.InputMediaDoTNPPING	(aiogram.enums.chat_action.ChatAction at-
attribute), 131	tribute), 457
<pre>type(aiogram.types.input_media_photo.InputMediaPhoto typing(</pre>	() (aiogram.utils.chat_action.ChatActionSender
	class method), 557
type (aiogram.types.input_media_video.InputMediaVideo TZS (aio attribute), 133	
type(aiogram.types.keyboard_button_poll_type.KeyboardButtonPol	lType
105	gram.enums.currency.Currency attribute), 463

UGX (aiogram.enums.currency.Currency attribute), 463 attribute), 90 unban() (aiogram.types.chat.Chat method), 41 until_date(aiogram.types.chat member restricted.ChatMemberRestricted unban_sender_chat() (aiogram.types.chat.Chat attribute), 93 method), 33 update (aiogram.types.error_event.ErrorEvent at-UnbanChatMember (class in tribute), 539 aiogram.methods.unban chat member), Update (class in aiogram.types.update), 284 412 update_data() (aiogram.fsm.scene.SceneWizard UnbanChatSenderChat method), 534 (class in aiogram.methods.unban_chat_sender_chat), update_data() (aiogram.fsm.storage.base.BaseStorage 414 method), 517 unclaimed_prize_count update_handler_flags() (aiogram.filters.base.Filter (aiogram.types.giveaway_completed.GiveawayCompleted method), 496 attribute), 123 update_id (aiogram.types.update.Update attribute), 284 UpdateType (class in aiogram.enums.update_type), 470 unclaimed_prize_count (aiogram.types.giveaway_winners.GiveawayWinn&pdateTypeLookupError, 286 attribute), 125 UPLOAD_DOCUMENT (aiogram.enums.chat_action.ChatAction UNDERLINE (aiogram.enums.message_entity_type.MessageEntityType attribute), 458 attribute), 467 upload_document()(aiogram.utils.chat action.ChatActionSender Underline (class in aiogram.utils.formatting), 571 class method), 557 UnhideGeneralForumTopic UPLOAD_PHOTO (aiogram.enums.chat action.ChatAction (class in aiogram.methods.unhide_general_forum_topic), attribute), 457 415 upload_photo() (aiogram.utils.chat_action.ChatActionSender UNKNOWN (aiogram.enums.content_type.ContentType atclass method), 557 tribute), 459 UPLOAD_VIDEO (aiogram.enums.chat action.ChatAction unpack() (aiogram.filters.callback_data.CallbackData attribute), 457 class method), 492 upload_video() (aiogram.utils.chat action.ChatActionSender unpin() (aiogram.types.message.Message method), 190 class method), 557 unpin_all_general_forum_topic_messages() UPLOAD_VIDEO_NOTE (aiogram.enums.chat_action.ChatAction (aiogram.types.chat.Chat method), 43 attribute), 458 unpin_all_messages() upload_video_note() (aiogram.types.chat.Chat method), 38 (aiogram.utils.chat_action.ChatActionSender unpin_message() (aiogram.types.chat.Chat method), class method), 557 UPLOAD_VOICE (aiogram.enums.chat_action.ChatAction 38 UnpinAllChatMessages attribute), 458 (class in aiogram.methods.unpin_all_chat_messages), upload_voice() (*aiogram.utils.chat action.ChatActionSender* 416 class method), 557 UnpinAllForumTopicMessages (class *in* UploadStickerFile (class in aiogram.methods.unpin_all_forum_topic_messages), aiogram.methods.upload_sticker_file), 306 417 URL (aiogram.enums.message_entity_type.MessageEntityType UnpinAllGeneralForumTopicMessages (class in attribute), 467 aiogram.methods.unpin all general forum topic unksraiges am.methods.answer callback query.AnswerCallbackQuery 418 attribute), 307 UnpinChatMessage (aiogram.methods.set webhook.SetWebhook (class in url ataiogram.methods.unpin_chat_message), 419 tribute), 454 unrestrict_boost_count (aiogram.types.chat.Chat url(aiogram.types.inline_keyboard_button.InlineKeyboardButton attribute), 32 attribute), 126 UNSPECIFIED (aiogram.enums.passport_element_error_typatPassportDie.typatPassportDie.typaery_result_article.InlineQueryResultArticle attribute), 469 attribute), 219 UnsupportedKeywordArgument, 540 url (aiogram.types.link_preview_options.LinkPreviewOptions until_date(aiogram.methods.ban_chat_member.BanChatMember_attribute), 139 attribute), 310 url (aiogram.types.login_url.LoginUrl attribute), 140 until_date(aiogram.methods.restrict_chat_member.RestrictChatMen(biogram.types.message_entity.MessageEntity attribute), 359 attribute), 192 until_date(aiogram.types.chat member banned.ChatMemaber(Biognedn.types.user.User property), 208

aiogram Documentation, Release 3.5.0

url (aiogram.types.web_app_info.WebAppInfo attribute), 215	<pre>user_chat_id(aiogram.types.chat_join_request.ChatJoinRequest</pre>
url (aiogram.types.webhook_info.WebhookInfo at- tribute), 286	user_id (aiogram.methods.add_sticker_to_set.AddStickerToSet attribute), 288
url (aiogram.utils.callback_answer.CallbackAnswer property), 566	user_id (aiogram.methods.approve_chat_join_request.ApproveChatJoinRe attribute), 308
Url (class in aiogram.utils.formatting), 570	user_id(aiogram.methods.ban_chat_member.BanChatMember
<pre>URLInputFile (class in aiogram.types.input_file), 127,</pre>	attribute), 309
474	user_id(aiogram.methods.create_new_sticker_set.CreateNewStickerSet
USD (aiogram.enums.currency.Currency attribute), 463	attribute), 290
use_independent_chat_permissions	user_id (aiogram.methods.decline_chat_join_request.DeclineChatJoinRed
(aiogram.methods.restrict_chat_member.Restrict attribute), 358	user_id (aiogram.methods.get_chat_member.GetChatMember
use_independent_chat_permissions	attribute), 337
	Paseixsidu(aiogram.methods.get_game_high_scores.GetGameHighScores
attribute), 401	<i>attribute</i>), 438
USER (aiogram.enums.message_origin_type.MessageOrigin attribute), 468	Typer_id (aiogram.methods.get_user_chat_boosts.GetUserChatBoosts attribute), 348
user (aiogram.types.business_connection.BusinessConnection.Busines	tionser_id (aiogram.methods.get_user_profile_photos.GetUserProfilePhotos attribute), 348
attribute), 47	owssem_racGiftGgrden.methods.promote_chat_member.PromoteChatMember attribute), 354
<pre>user (aiogram.types.chat_boost_source_giveaway.ChatBoo attribute), 48</pre>	p x/Sen_r_adGiviergwary n.methods.replace_sticker_in_set.ReplaceStickerInSet attribute), 295
<pre>user (aiogram.types.chat_boost_source_premium.ChatBoo attribute), 48</pre>	stSenrcied (aiognum.methods.restrict_chat_member.RestrictChatMember attribute), 358
user (aiogram.types.chat_member_administrator.ChatMen attribute), 88	n bsel adninistrator_custom_title.SetChatAd attribute), 398
user (aiogram.types.chat_member_banned.ChatMemberBa attribute), 90	msed_id(aiogram.methods.set_game_score.SetGameScore attribute), 440
user (aiogram.types.chat_member_left.ChatMemberLeft attribute), 90	user_id (aiogram.methods.set_passport_data_errors.SetPassportDataErro attribute), 456
user (aiogram.types.chat_member_member.ChatMemberMattribute), 91	<pre>tarsber_id (aiogram.methods.set_sticker_set_thumbnail.SetStickerSetThumbn attribute), 303</pre>
user (aiogram.types.chat_member_owner.ChatMemberOw attribute), 91	nuser_id (aiogram.methods.unban_chat_member.UnbanChatMember attribute), 413
user (aiogram.types.chat_member_restricted.ChatMember attribute), 92	Resenicied (aiogram.methods.upload_sticker_file.UploadStickerFile attribute), 306
user (aiogram.types.game_high_score.GameHighScore attribute), 288	user_id (aiogram.types.bot_command_scope_chat_member.BotCommand_ attribute), 23
user (aiogram.types.message_entity.MessageEntity at-	user_id (aiogram.types.contact.Contact attribute), 115
tribute), 192	user_id (aiogram.types.shared_user.SharedUser at-
user (aiogram.types.message_reaction_updated.MessageR	
attribute), 197	user_id (aiogram.types.user_shared.UserShared at-
user (aiogram.types.poll_answer.PollAnswer attribute), 199	tribute), 209 user_ids (aiogram.types.users_shared.UsersShared at-
user (aiogram.utils.web_app.WebAppInitData attribute),	tribute), 210
560	user_is_bot (aiogram.types.keyboard_button_request_user.KeyboardButton_request_user.Ke
User (class in aiogram.types.user), 207	attribute), 137
user_administrator_rights	user_is_bot (aiogram.types.keyboard_button_request_users.KeyboardBu
(aiogram.types.keyboard_button_request_chat.Ke	eyboardButaonRaque,stGRat
attribute), 137	user_is_premium (aiogram.types.keyboard_button_request_user.Keyboar
user_chat_id (aiogram.types.business_connection.Busine	
attribute), 25	user_is_premium(aiogram.types.keyboard_button_request_users.Keyboard_button_request_users

attribute), 138	attribute), 94
USER_SHARED (aiogram.enums.content_type.ContentType	
attribute), 461	tribute), 459
user_shared (aiogram.types.message.Message at- tribute), 150	VIDEO (<i>aiogram.enums.inline_query_result_type.InlineQueryResultType</i> <i>attribute</i>), 465
UserChatBoosts (class in	VIDEO (aiogram.enums.input_media_type.InputMediaType
aiogram.types.user_chat_boosts), 209	attribute), 466
username (<i>aiogram.types.chat.Chat attribute</i>), 30	VIDEO (aiogram.enums.sticker_format.StickerFormat at-
username (aiogram.types.chat_shared.ChatShared at-	tribute), 469
tribute), 114	video (aiogram.methods.send_video.SendVideo at-
username (aiogram.types.shared_user.SharedUser	tribute), 390
attribute), 205	<pre>video (aiogram.types.external_reply_info.ExternalReplyInfo</pre>
username (<i>aiogram.types.user.User attribute</i>), 207 username (<i>aiogram.utils.web_app.WebAppChat at-</i>	video (aiogram.types.message.Message attribute), 146
tribute), 562	Video (alogram.types.message.message antibute), 140 Video (class in alogram.types.video), 211
username (aiogram.utils.web_app.WebAppUser at-	
tribute), 561	attribute), 461
	video_chat_ended (aiogram.types.message.Message
aiogram.types.user_profile_photos), 209	attribute), 149
users (aiogram.types.users_shared.UsersShared at-	VIDEO_CHAT_PARTICIPANTS_INVITED
tribute), 210	(aiogram.enums.content_type.ContentType
users (aiogram.types.video_chat_participants_invited.Vid	
attribute), 212	video_chat_participants_invited
USERS_SHARED (aiogram.enums.content_type.ContentType attribute), 460	e (aiogram.types.message.Message attribute), 149
users_shared (aiogram.types.message.Message at-	VIDEO_CHAT_SCHEDULED
tribute), 148	(aiogram.enums.content_type.ContentType
UserShared (class in aiogram.types.user_shared), 209	attribute), 461
UsersShared (class in aiogram.types.users_shared), 210	
UTILITY_BILL (aiogram.enums.encrypted_passport_elem attribute), 464	149
UYU (aiogram.enums.currency.Currency attribute), 463	VIDEO_CHAT_STARTED (aiogram.enums.content_type.ContentType
UZS (aiogram.enums.currency.Currency attribute), 464	attribute), 461
V	<pre>video_chat_started (aiogram.types.message.Message</pre>
value (aiogram.types.dice.Dice attribute), 115	<pre>video_duration(aiogram.types.inline_query_result_video.InlineQueryRe</pre>
vcard (aiogram.methods.send_contact.SendContact at-	attribute), 254
tribute), 368 vcard (aiogram.types.contact.Contact attribute), 115	<pre>video_file_id (aiogram.types.inline_query_result_cached_video.InlineQ attribute), 235</pre>
vcard (alogram types contact. Contact altroute), 115	uvidResultionship (aiogram.types.inline_query_result_video.InlineQueryResu
attribute), 238	attribute), 254
vcard (aiogram.types.input_contact_message_content.Inpu attribute), 257	uXCDEAcMAEssageicoranenums.content_type.ContentType attribute), 459
VENUE (<i>aiogram.enums.content_type.ContentType</i> at- tribute), 459	<pre>video_note(aiogram.methods.send_video_note.SendVideoNote</pre>
VENUE (aiogram.enums.inline_query_result_type.InlineQue attribute) 465	ery kdesalt nope (aiogram.types.external_reply_info.ExternalReplyInfo attribute), 118
venue (aiogram types external renty info External Rentyle	gideo_note (aiogram.types.message.Message attribute),
attribute), 118	146

via_bot(aiogram.types.message.Message attribute), 146 video_width(aiogram.types.inline_query_result_video.InlineQueryResult via_chat_folder_invite_link attribute), 254

aiogram.types.video chat ended), 212 web_app_name (aiogram.types.write access allowed.WriteAccessAllowed VideoChatParticipantsInvited (class in attribute). 215 aiogram.types.video chat participants invited), web_app_query_id(aiogram.methods.answer web app query.AnswerWe 212 attribute), 436 VideoChatScheduled (class in WebAppChat (class in aiogram.utils.web_app), 562 aiogram.types.video chat scheduled), 212 WebAppData (class in aiogram.types.web app data), 214 VideoChatStarted (class WebAppInfo (class in aiogram.types.web app info), 215 in aiogram.types.video chat started), 213 WebAppInitData (class in aiogram.utils.web app), 560 VideoNote (class in aiogram.types.video note), 213 WebAppUser (class in aiogram.utils.web app), 561 VIOLET (aiogram.enums.topic_icon_color.TopicIconColor WebhookInfo (class in aiogram.types.webhook_info), attribute), 470 286 VND (aiogram.enums.currency.Currency attribute), 464 width (aiogram.methods.send_animation.SendAnimation VOICE (aiogram.enums.content_type.ContentType atattribute), 361 tribute), 459 (aiogram.methods.send_video.SendVideo width at-VOICE (aiogram.enums.inline_query_result_type.InlineQueryResultTypebute), 390 attribute), 465 width (aiogram.types.animation.Animation attribute), 17 (aiogram.methods.send_voice.SendVoice width(aiogram.types.input_media_animation.InputMediaAnimation voice attribute), 395 attribute), 129 $\verb|voice|(aiogram.types.external_reply_info.ExternalReplyInfwidth(aiogram.types.input_media_video.InputMediaVideo)||$ attribute), 118 attribute), 133 voice (aiogram.types.message.Message attribute), 147 width (aiogram.types.photo_size.PhotoSize attribute), Voice (class in aiogram.types.voice), 214 197 voice_duration (aiogram.types.inline_query_result_voiceshidthe Quogramstoffesticker.Sticker attribute), 265 attribute), 256 width (aiogram.types.video.Video attribute), 211 voice_file_id(aiogram.types.inline query result cachedrimmer.IndiverCountersealthtCountersealthCountersealthC attribute), 237 tribute). 123 voice_url (aiogram.types.inline_query_result_voice.Inline@imm@Resolutitie@iogram.types.giveaway_completed.GiveawayCompleted attribute), 255 attribute), 123 voter_chat (aiogram.types.poll_answer.PollAnswer atwinner_count (aiogram.types.giveaway_winners.GiveawayWinners tribute), 199 attribute), 124 voter_count (aiogram.types.poll_option.PollOption atwinners (aiogram.types.giveaway_winners.GiveawayWinners tribute), 199 attribute), 124 winners_selection_date W (aiogram.types.giveaway.Giveaway attribute), was_refunded (aiogram.types.giveaway_winners.GiveawayWinners 123 winners_selection_date attribute), 125 watcher (aiogram.types.proximity_alert_triggered.ProximityAlertTriggingatam.types.giveaway_winners.GiveawayWinners attribute), 124 attribute), 200 $\texttt{WEB_APP}\ (aiogram.enums.menu_button_type.MenuButtonTy \texttt{WEap_local_file}\ (aiogram.client.telegram.TelegramAPIServer)$ attribute), 13 attribute), 467 web_app (aiogram.types.inline_keyboard_button.InlineKeybWBardButtonESS_ALLOWED (aiogram.enums.content_type.ContentType attribute), 126 web_app (aiogram.types.inline_query_results_button.InlineQueryResattsButton 460 write_access_allowed attribute), 256 attribute), (aiogram.types.message.Message web_app (aiogram.types.keyboard_button.KeyboardButton 148 attribute), 135 web_app (aiogram.types.menu_button.MenuButton at- WriteAccessAllowed (class in aiogram.types.write_access_allowed), 215 tribute), 142 $web_app\,(aiogram.types.menu_button_web_app.MenuButtonWebApp$ attribute), 143 WEB_APP_DATA (aiogram.enums.content_type.ContentType x_shift (aiogram.types.mask_position.MaskPosition atattribute), 461 tribute), 264 web_app_data (aiogram.types.message.Message attribute), 149

Y

y_shift (aiogram.types.mask_position.MaskPosition attribute), 264 year (aiogram.types.birthdate.Birthdate attribute), 19 YELLOW (aiogram.enums.topic_icon_color.TopicIconColor attribute), 470 YER (aiogram.enums.currency.Currency attribute), 464

Ζ

ZAR (aiogram.enums.currency.Currency attribute), 464